Approximating Response Times of Static-Priority Tasks with Release Jitters

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

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

Abstract

We consider static-priority tasks with constrained-deadlines that are subjected to release jitters. We define an approximate worst-case response time analysis and we propose a polynomial time algorithm. For that purpose, we extend the Fully Polynomial Time Approximation Scheme (FPTAS) presented in [2] to take into account release jitters; this feasibility test is then used to define a polynomial time algorithm that computes approximate worst-case response times of tasks. Nevertheless, the approximate worst-case response time values have not be proved to have any bounded error in comparison with worst-case response times.

1 Introduction

Guaranteeing that tasks will always meet their deadlines is a major issue in the design of hard-real time systems. A real-time system is said feasible if no deadline miss can occur at run-time. We next consider periodic tasks scheduled by a preemptive static-priority scheduler upon a uniprocessor platform. We consider tasks having constrained-deadlines (i.e., deadlines are less than or equal to task periods) and subjected to release jitters. Such a task model allows to analyze hard real-time distributed systems [11].

The feasibility problem consists on 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 [4, 6, 8]. Sufficient feasibility conditions are known and can be checked in polynomial time. But, when such a test returns ”not feasible”, this can be a rather pessimistic decision. Recently, approximate feasibility algorithms have been designed to reduce the gap between both approaches. According to an accuracy parameter \( \epsilon \), they check, in polynomial time, if a task set is:

- feasible (upon a unit speed processor).
- infeasible upon a \((1-\epsilon)\)-speed processor. That is, “we must effectively ignore \( \epsilon \) of the processor capacity for the test to become exact” [2]. So, the pessimism introduced by the feasibility test is kept bounded by a constant.

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. According to a accuracy parameter \( \epsilon \), we define approximate worst-case response times as follow:

Definition 1 Let \( \epsilon \) be a constant and \( R^*_i \) be the worst-case response time of a task \( \tau_i \); then the approximate worst-case response times \( R^{\epsilon}_i \) satisfies: \[ R^*_i \leq R^{\epsilon}_i \leq (1+\epsilon)R^*_i. \]

We first define a preliminary result for computing worst-case response time while performing a processor demand analysis (e.g., [6]), then we extend the FPTAS presented in [2] with release jitters. These results are then combined to define for computing approximate worst-case response times. Nevertheless, the computed approximate worst-case response time values are not guaranteed to be closed to worst-case response times (i.e., with a bounded error).

2 Task model and exact analysis

2.1 Task model

A task \( \tau_i, 1 \leq i \leq n \), is defined by a worst-case execution requirement \( C_i \), a relative deadline \( D_i \) and a period between two successive releases \( T_i \). 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 [5].

In order to model delay due to input data communications of tasks, we also consider that jobs are subjected to release jitters. A release jitter \( J_i \) of a task \( \tau_i \) is a interval of time after the release of a job in which it waits for its input data. When release jitters are considered in the task model, then 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 analyzed processor by processor,

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: \( \tau_1 \) is the highest priority task and \( \tau_n \) is the lowest priority one.

2.2 Known results

2.2.1 Request Bound and Workload Functions

The request bound function of a task \( \tau_i \) 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 [11] for details):

\[
\text{RBF}(\tau_i, t) \overset{\text{def}}{=} \left\lceil \frac{t + J_i}{T_i} \right\rceil C_i \tag{1}
\]

\[
W_i(t) \overset{\text{def}}{=} C_i + \sum_{j=1}^{i-1} \text{RBF}(\tau_j, t) \tag{2}
\]

Notice that deadline failures of \( \tau_i \) (if any) occur necessarily in an interval of time where only tasks with a priority higher of equal to \( i \) are running. Such an interval of time is defined as a level-\( i \) busy period [6]. Using these functions, two distinct (but linked) exact feasibility tests can be defined. We recall both results that will be reused in the remainder.

2.2.2 Processor Demand Analysis

The processor demand approach checks that the processor capacity is always less than or equal to the processor capacity required by task executions. In [6] is presented a processor demand schedulability test for constrained-deadline systems (but the test was extended for arbitrary deadline systems in [5]). It can be also easily extended to tasks subjected to release jitters as stated in the following result:

**Theorem 1** [6] A static-priority system with release jitters is feasible iff \( \max_{i=1..n} \left\{ \min_{t \in S_i} \frac{W_i(t)}{t} \right\} \leq 1 \), where \( S_i \) is the set of scheduling points defined as follows: \( S_i \overset{\text{def}}{=} \{ aT_j - J_j \mid j = 1..i, a = 1..\left\lfloor \frac{D_i + J_i}{T_i} \right\rfloor \} \cup \{ D_i \} \).

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.

2.2.3 Response Time Analysis

An alternative approach to check the feasibility of a static-priority task set is to compute the worst-case response time \( R_i^* \). The worst-case response time of \( \tau_i \) is formally defined as:

**Definition 2** The worst-case response time of a task \( \tau_i \) is:

\[
R_i^* \overset{\text{def}}{=} (\min\{t \in (0, D_i) \mid W_i(t) = t\}) + J_i
\]

An exact algorithm is known [4] (e.g., for a recursive definition of the following method). Using successive approximations starting from a lower bound of \( R_i^* \), we can compute to the smallest fixed-point of \( W_i(t) = t \) with the following iterative process: \( W_i^{(0)} = \sum_{j=1}^{i} C_j \), \( W_i^{(k+1)} = C_i + \sum_{j=1}^{i-1} \text{RBF}(\tau_j, W_i^{(k)}) \). Computations stop for the smallest integer \( k \) such that: \( W_i^{(k+1)} = W_i^{(k)} \).

These approaches are all based on the analysis of the cumulative processor demand [9]. But, as far as we know, no direct link has been presented between these approaches. 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 [10, 1] 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.

2.3 A preliminary result

We show that worst-case response times of tasks can be easily computed using a Time Demand Analysis (i.e., Theorem 1), for every feasible task set (and only for them). For a feasible task \( \tau_i \), it is sufficient to check the following testing set [6]: \( S_i = \{ aT_j - J_j \mid j = 1..i, a = 1..\left\lfloor \frac{D_i + J_i}{T_i} \right\rfloor \} \cup \{ D_i \} \).

We first define the critical scheduling point that allows to compute the worst-case response time of \( \tau_i \) (under the assumption that the task \( \tau_i \) will meet its deadline at execution time).

**Definition 3** The critical scheduling point for a feasible task \( \tau_i \) is: \( t^* \overset{\text{def}}{=} \min\{t \in S_i \mid W_i(t) \leq t\} \).

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

**Theorem 2** The worst-case response time of a task \( \tau_i \), such that \( W_i(t^*) \leq t^* \) is exactly \( R_i^* = W_i(t^*) + J_i \).

**Proof:** Since we assume that \( W_i(t^*) \leq t^* \), then \( \tau_i \) is feasible. Let \( S_i = \{ t_{i1}, t_{i2}, \ldots, t_{i \ell} \} \) with \( t_{i1} < t_{i2} < \cdots < t_{i \ell} < \cdots < t_{id} = D_i \). By Definition 3, 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 \( t < t_i \leq t_{i \ell} \) and \( W_i(t_{ij}) \leq t_{ij} \).

Since \( W_i(t) \) is non-decreasing between subsequent scheduling points \( \{ t_{ia}, t_{ia+1} \} \), \( a \leq a < \ell - 1 \), 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 \( \tau_i \) is then defined as \( W_i(t^*) + J_i \).

Thus, for all feasible tasks, it is quite easy to compute their worst-case response times. But, for an infeasible task \( \tau_i \) (e.g., \( R_i > D_i \)), there is not scheduling point \( t \in T_i \) 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 \( \sum_{j=1}^{i-1} \frac{D_j + J_j}{T_j} \), 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.

3 A FPTAS for feasibility analysis of task

3.1 Approximating Request Bound Function

For synchronous task systems without release jitters, the worst-case activation scenario for the tasks occurs when they are simultaneously released [7]. When tasks are subject to release jitters, then the worst-case processor workload occurs when tasks are simultaneously available after \( J_i \) units of time (i.e., when their input data are available). If we assume that tasks become simultaneously available by time \( 0 \), then the worst-case workload in a processor busy period is defined by the release at time \( -J_i \). According to such a scenario, the total execution time requested at time \( t \) by a task \( \tau_i \) is defined by [11]:

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

Thus, up to \((k-1)T_i\) no approximation is performed to evaluate the total execution requirement of \( \tau_i \), and after that it is approximated by a linear function with a slope equal to the utilization factor of \( \tau_i \).

3.2 Approximation scheme

In [11] is shown that a static-priority task system with release jitters 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 using the principles of the well-known exact feasibility test presented for the rate monotonic scheduling algorithm in [6].

The cumulative request bound function at time \( t \) is defined by:

\[
W_i(t) \triangleq C_i + \sum_{j=1}^{i-1} RBF(\tau_j, t).
\]

A task \( \tau_i \) is feasible (with a constrained relative deadline) iff, there exists a time \( t, 0 \leq t \leq D_i \), 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 its value of \( C_i \) for every scheduling point in the following set \( S_i = \{ t = bT_a - J_a : a = 1 \ldots i, b = 1 \ldots \left( \frac{D_a + 1}{J_a} \right) \} \}. \)

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:

\[
\hat{W}_i(t) \triangleq C_i + \sum_{j=1}^{i-1} \hat{\delta}(\tau_j, t).
\]

The key point to ensure the correctness is:

\[
\delta(\tau_i, t)/RBF(\tau_i, t) \leq (1 + \epsilon). \]

This key point 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:

\[
\frac{\delta(\tau_i, t)}{RBF(\tau_i, t)} \leq (1 + \frac{1}{2}) \frac{\delta(\tau_i, t)}{RBF(\tau_i, t)} \text{ where } k = \left[ \frac{1}{\epsilon} \right] - 1.
\]

**Proof:** We first prove the first inequality: for all \( t \in [0,(k-1)T_i - J_i] \), \( \delta(\tau_i, t) = RBF(\tau_i, t) \).

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

As a consequence:

\[
\delta(\tau_i, t) \geq \left( \frac{t + J_i}{T_i} \right) C_i = RBF(\tau_i, t).
\]

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

\[
RBF(\tau_i, t) \geq kC_i \quad (4)
\]

Furthermore, \( \delta(\tau_i, t) - RBF(\tau_i, t) \leq C_i \); this is obvious if \( t \in [0,(k-1)T_i - J_i] \) since \( \delta(\tau_i, t) = RBF(\tau_i, t) \), and if \( t > (k-1)T_i - J_i \) then:

\[
\delta(\tau_i, t) - RBF(\tau_i, t) = C_i + (t + J_i) \left( \frac{C_i}{T_i} \right).
\]

As a consequence:

\[
\delta(\tau_i, t) \leq RBF(\tau_i, t) + C_i \text{ and using inequality (4), we obtain the result: } \delta(\tau_i, t) \leq (1 + \frac{1}{2}) \frac{\delta(\tau_i, t)}{RBF(\tau_i, t)}.
\]
Using the same approach presented in [2, 3], we can establish the correctness of approximation.

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

**Proof:** Directly follows from Theorem 3

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

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

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

To conclude the correctness of the approximation scheme, we need to prove that scheduling points are sufficient.

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

**Proof:** Let \( t_1 \) and \( t_2 \) be two adjacent points in \( \hat{S}_i \) (i.e., \( t_1 < t < t_2 \)). Since \( \hat{W}_i(t_1) > t_1 \), \( \hat{W}_i(t_2) > t_2 \) and the fact that \( \hat{W}_i(t) \) is a non-decreasing step left-continuous function we conclude that \( \forall t \in (t_1, t_2), \hat{W}_i(t) > t \). The property follows.

### 4 Approximate Response Time Analysis with release jitters

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 \( \tau_i \) such that there exists a time \( t \) satisfying \( \hat{W}_i(t) \leq t \), then an approximate worst-case response time is defined by:

\[
t^* \overset{\text{def}}{=} \min \left( t \in \hat{S}_i \mid \hat{W}_i(t) \leq t \right) \quad \text{and} \quad \hat{R}_i^* \overset{\text{def}}{=} \hat{W}_i(t^*) + J_i.
\]

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

**Theorem 7** For every task \( \tau_i \) such that there exists a time \( t \) satisfying \( \hat{W}_i(t) \leq t \), then: \( R_i^* \leq \hat{R}_i^* \)

**Proof:** Let \( t \) be a scheduling point such that \( \hat{W}_i(t) \leq t \). From the approximate feasibility test, we verify that \( \tau_i \) 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 \( \hat{R}_i^* = \hat{W}_i(t) + J_i \), then, it follows from properties of the approximate feasibility test that \( R_i^* \leq \hat{R}_i^* \).

It can be shown that this method does not lead to an approximation algorithm (i.e., with the expected bounded error presented in Definition 1).

### 5 Conclusion

The existence of an approximate scheme (or weakly an approximation algorithm) to solve that problem is still an interesting open issue.

**Acknowledgments** The authors are grateful to Nathan Fisher for detecting an error in the previous version of the paper.

### References