Polynomial Time Approximate Schedulability Tests for Fixed-Priority Real-Time Tasks: some numerical experimentations

Pascal Richard
Laboratoire d'Informatique Scientifique et Industrielle
ENSMA
1, avenue Clément Ader
Téléport 2 - BP 40109
86961 Futuroscope Cedex (France)
pascal.richard@univ-poitiers.fr

Abstract

Efficient schedulability tests are required for analyzing large task systems or for designing on-line admission controllers. We next focus on periodic fixed-priority tasks. For fixed-priority tasks with constrained deadlines (i.e., deadlines are less than or equal to periods), no exact polynomial time feasibility test is known. We propose several polynomial time algorithms with performance guarantees (with an ijnput accuracy parameter) and compare them with known exact feasibility tests (running in pseudo-polynomial time) and a fully polynomial time approximation scheme (FPTAS). Our main objective is to define the capabilities of such algorithms according to the system workload and an accuracy parameter defining the quality of results to compute.

1 Introduction

Large real-time systems are emerging in many applications, including industrial automation, defense and telecommunication. For these systems, the exact workload cannot be predicted and there are significant runtime uncertainties due to the controlled environment or system resources states. In many case, best effort strategies are required to admit or reject works. After the admission control, all admitted tasks must meet their timing requirements. Many admission controllers are dedicated to improve some Quality of Service (QoS) metrics or a benefit function. They are usually based on two on-line algorithms: an admission controller that checks if a new task can be accepted without any consequence on already admitted tasks and a scheduler that chooses the next task to run among uncompleted admitted tasks. In many systems, tasks are assumed to be periodic, but their first release time is not predictable (i.e., tasks are released over time) and can be killed due to system mode changes. Due to such a dynamic arrival of works, these real-time systems must cope with temporary overloaded conditions (using an admission controller to regulate admitted workload while ensuring that task deadlines will be met). Polynomial time schedulability tests are necessary to define efficient admission controllers.

Checking the feasibility of a task system is usually a hard computational problem, that cannot be solved in polynomial time in the number of tasks. Exact feasibility tests are known for periodic fixed-priority tasks [10, 11] and run in pseudo-polynomial time. Furthermore, their execution times can vary from one execution to another according to the task parameters [13, 6]. Nevertheless, there are two ways for defining efficient schedulability tests that consists on:

- improving initial values of an exact feasibility test as in [13] or [6]. But, the worst-case computational complexity of such tests is still pseudo-polynomial,
- defining an approximate schedulability test running in polynomial time as in [7, 1, 9].

Next, we focus on the second promising way. Approximation algorithms perform a compromise between computational effort to decide the feasibility of task systems and the quality of taken decisions. If the approximate algorithm concludes that a task system is feasible, then it will be true at run-time for all possible behaviors of these tasks. But, if the answer is negative, then we cannot conclude that the task system will be infeasible at run-time.

The paper is organized as follows: Section 2 presents the task model considered in the remaining of the paper. Section 3 presents known exact feasibility tests and approximate feasibility tests for periodic fixed-priority tasks. Section 4 presents some computational complexity results and new polynomial time feasibility tests for fixed-priority real-time tasks. Section 5 presents experimental results based on numerical simulations.

2 The task model

We consider uniprocessor real-time systems running periodic tasks. A periodic task τ_i defines a set of jobs. Every periodic task is known and implemented in the software architecture. Thus, job parameters are always known before starting the system. Every task τ_i is defined by three parameters and denoted $\tau_i(C_i, D_i, T_i)$. C_i is the worst-case execution requirement of τ_i, D_i its the relative deadline (the time window between its release and its completion), and T_i its period between two successive releases. We assume that deadlines are constrained: $D_i \le T_i, 1 \le i \le n$, where n is the number of tasks in the system.

Every job generated by a periodic task is scheduled using a fixed-priority. At any time, the highest priority job is run among available ones. According to such a basic dispatching policy, the optimal priority assignment can be performed off-line using the Deadline Monotonic [2] priority ordering. We assume that task priorities are known before starting the system (i.e. priority assignment is done off-line) and tasks are indexed using the priority ordering, thus τ_1 is the highest priority task.

3 Review of feasibility tests for preemptive fixed-priority task systems

Three main approaches are used to define schedulability tests: analyzing the system utilization factor (i.e., $\sum_{i=1}^{n} C_i/T_i$), analyzing the processor demand or analyzing worst-case response times of tasks. For fixed-priority tasks, tests are known for checking a sufficient schedulability condition of tasks having deadlines equal to periods such as [12]. A necessary and sufficient schedulability condition can be computed in pseudo-polynomial time for systems having constrained-deadlines using a processor demand analysis or by computing worst-case response times of tasks. But, no polynomial time algorithm nor NP-hardness result are currently known for the feasibility problem related to the studied task model. Next, we only present results and schedulability tests that will be used in the remainder of the paper.

3.1 Exact algorithms

For a given task τ_i, the scenario leading to its worst-case response time R_i is achieved when task τ_i is released at a *critical instant* (i.e., simultaneously with all higher priority tasks) [12]. The processor demand analysis is based on the total execution time required by a task τ_i and can be expressed as of function of time. In a periodic synchronous task system, the total execution time requested by task τ_i is (request bound function):

$$rbf_i(t) = \left\lceil \frac{t}{T_i} \right\rceil C_i \tag{1}$$

The *cumulative request bound function* allows to compute the worst-case response time of task τ_i:

$$W_i(t) = C_i + \sum_{j=1}^{i-1} rbf_i(t) \tag{2}$$

For task τ_i, its exact worst-case response time R_i^* is the minimal solution to the equation:

$$W_i(R_i^*) = R_i^* \tag{3}$$

Joseph and Pandya [10] proposed a recursive algorithm to solve the previous equation. But an iterative algorithm can be defined using successive approximation of response times in order to reach the smallest fixed-point of Equation 3 (this lead to simple recursion formula). The feasibility test consists on: first, computing worst-case response times of all tasks, and second, checking that $R_i^* \le D_i, 1 \le i \le n$. The corresponding algorithm is pseudo-polynomial and the number of iterations before reaching the smallest fixed-point widely varies from one task system to another and is highly dependent on task parameters [13, 6].

Lehoczky *et al.* [11] provided a processor demand analysis for checking task feasibility that will lead in practice to a different feasibility test. Their main result is stated hereafter:

Theorem 1 *[11] In a synchronous task system, task τ_i is feasible if, and only if, there exists a time $t \in (0, D_i]$ such that $W_i(t) \le t$.*

Such a result defines an alternative way to check feasibility of a task system, without explicitly computing worst-case response times. The cumulative request bound function (defined in Equation 2) only changes for a finite set of values (i.e., when tasks are released). Thus, the number of time instants to check the feasibility of task τ_i in Theorem 1 is defined by the following testing set (for constrained-deadline task systems):

$$S_i = \{bT_j | j = 1 \dots i, b = 1 \dots \lfloor D_i/T_j \rfloor\} \cup \{D_i\} \tag{4}$$

Thus, checking task τ_i feasibility requires to verify if:

$$\min_{t \in S_i} \left(\frac{W_i(t)}{t} \right) \le 1 \tag{5}$$

As a consequence if one instant $t \in S_i$ satisfies $W_i(t) \le t$ then τ_i is feasible and no more time instant has to be checked to decide the feasibility of τ_i. According to Theorem 1, a practical implementation of such a test usually requires to check only a subset of S_i. But, the computational complexity of this algorithm depends on the ratio: D_i/T_j. As a consequence, the algorithm runs in pseudo-polynomial time. In [5], an improvement of this test is presented, but this algorithm is still running in pseudo-polynomial time.

In practice, the algorithms proposed by [10] and [11] can lead to a quite different number of iterations. But, their pseudo-polynomial complexities are not acceptable to define an on-line admission controller and furthermore, the numbers of iterations are too dependent on task parameters.

3.2 Approximation algorithms

An approximation algorithm is a polynomial time algorithm that is used to solve efficiently NP-hard (optimization) problems. There exist several ways to define a solution in polynomial time with performance guarantees in comparison with an exact algorithm (always computing the optimal value of an optimized function). Let A be an approximation algorithm and OPT be an exact algorithm. For any instance, vales returned by A or OPT for a given instance I are respectively denoted $A(I)$ and $OPT(I)$. The (relative) performance guarantee of the algorithm A is defined by a *ratio* $A(I)/OPT(I)$ while considering any possible instance I of a given optimization problem. The competitive ratio of A is thus defined by: $r_A = \inf_{any I} A(I)/OPT(I)$, where I is a instance of the considered problem. Thus, the ratio defines the worst-case performance guarantee while considering all possible instances of the optimization problem. An approximation algorithm is a polynomial time algorithm having a ratio bounded by a constant. Note that an algorithm A is optimal (i.e., always leads to the optimal value of the optimized objective function) if, and only if, $r_A = 1$.

A approximation scheme is a parametric approximation algorithm (thus running in polynomial time) that takes an input problem instance and an error bound $0 < \epsilon < 1$. The error bound defines an *accuracy* input parameter. The ratio of an approximation scheme must be defined as follows: $r_A \leq 1 + \epsilon$. A Polynomial-Time Approximation Scheme (PTAS) is an algorithm that runs in polynomial time in the length of the input. A fully polynomial time algorithm (FPTAS) is a PTAS that satisfies an additional condition: it is also polynomial in $1/\epsilon$. That is the best result that can be achieved to solve an NP-hard problem. Only few optimization problems admit FPTAS.

Since few years, approximation algorithms gain a great interest in the real-time research community. To the best of our knowledge, no approximation algorithm has been proposed to calculate approximate response times of tasks with performance guarantees (we shall provide such a result in the next section). Nevertheless, checking feasibility is not an optimization problem, but only a *decision problem*. As a consequence, approximation algorithm principles cannot be exploited without revisiting their definition. In fact, several frameworks have been proposed to reuse approximation algorithm concepts and thus defining several approaches to perform *approximate schedulability analysis*:

- Chakraborty *et al.* [7] proposed approximation scheme that always provide the good answer if the task system is not schedulable, but can give a wrong answer in the other case with a bounded error ϵ (i.e., it returns not schedulable whereas the task system is feasible).
- Based on the results obtained by [1] for EDF, Fisher and Baruah [9] proposed another definition: if a task system is stated as infeasible then it is really not feasible on a slower processor (with speed $1 - \epsilon$).

Even if these two frameworks are different, the performance guarantee of an approximation algorithm is obtained by bounding the error on the exact value of the function $rbf(t)$ and its approximate version. We only present the function proposed in [9] that is directly linked (and will be reused) to the problem we cope with in this paper.

The function $rbf_i(t)$ is a non-decreasing step function. The number of steps is not bounded by any polynomial function in the size of task parameters. One way to define an polynomial-time approximation scheme is to consider a limited number k of steps (polynomially bounded in the number of tasks in the system) and then to use a linear function to define an upper bound of $rbf_i(t)$. The number of steps that will be considered while computing the approximate request bound function is defined as follow:

$$k = \lceil 1/\epsilon \rceil - 1 \tag{6}$$

Then, the approximate demand bound function $\overline{rbf}_i(t)$ is defined by considering the first k steps of $rbf_i(t)$:

$$\begin{aligned} \overline{rbf}_i(t) &= rbf_i(t) && \text{if} \quad t \leq (k-1)T_i \\ &= C_i + t\frac{C_i}{T_i} && \text{otherwise} \end{aligned}$$

Then, the *approximate cumulative request bound function* is defined by:

$$\overline{W}_i(t) = C_i + \sum_{j=1}^{i-1} \overline{rbf}_j(t) \tag{7}$$

To complete the test, Fisher and Baruah use exactly the same principle than those proposed by Lehoczky et al. [11] but defining a testing set, but having a polynomial number of entries according to the input task system size and the accuracy parameter (ϵ):

$$\overline{S}_i = \{bT_j | j = 1 \dots i-1, b = 1 \dots k\} \cup \{D_i\} \tag{8}$$

where k is defined in Equation 6. A basic implementation of this approximate schedulability test leads to an $O(n^2/\epsilon)$ algorithm [9]. Clearly, if ϵ is closed to 0, then the number of iterations performed by the algorithm is quite huge and should not be acceptable into an on-line admission controller (even if it is a polynomial time algorithm from a theoretical point of view). Thus, numerical experimentations are necessarily required according to the application in order to define a *good* value for ϵ for the considered task systems. Note that such an approximation scheme has been extended to task systems with arbitrary deadlines in [8] (i.e., periods and deadlines are not related).

4 New Algorithms

We first present some computation complexity results, and then, we propose three new polynomial time algorithms for checking the feasibility of fixed-priority tasks with constrained-deadlines.

4.1 Computational complexity of feasibility problems

The computation complexity theory classifies decision problems according to their internal complexity. Checking feasibility of a task system is obviously a decision problem. Nevertheless, no computational complexity result is known for the feasibility problem related to the studied task model. This decision problem is not known $\mathcal{NP}$-hard, nor belonging to $\mathcal{NP}$. Before defining approximation algorithms, we first state a computational complexity result for fixed-priority tasks, then we recall that verifying that tasks scheduled under EDF (Earliest Deadline First) leads to a very different class of problems in the computational complexity theory, unless $\mathcal{P}$ equals $\mathcal{NP}$.

Theorem 2 *Checking deadlines for synchronous fixed-priority tasks having constrained-deadlines is a decision problem belonging to $\mathcal{NP}$.*

Proof: In order to show the problem to be in $\mathcal{NP}$, we have to prove that a task set can be decided feasible using a polynomial time non-deterministic algorithm. If the non-deterministic part of such an algorithm "guesses" a scheduling point t in the testing set defined in Equation 4 for checking the feasibility of a task τ_i, then a necessary and sufficient condition according to Theorem 1 is: $W_i(t) \leq t$. Such a test is done in polynomial time since the Equation 2 is computable in linear time. Repeating this principle for every task leads to a polynomial time test (using a non deterministic algorithm). Thus, the considered feasibility problem belongs to $\mathcal{NP}$. □

Note that the same feasibility problem will be in co-$\mathcal{NP}$ if we consider an EDF scheduler (thus, one can checked in polynomial time for a given date t that a task system is infeasible, but checking that a task system is feasible requires more than a polynomial amount of time [3]).

Theorem 3 *[3] Checking deadlines for synchronous tasks having constrained deadlines, to be scheduled under EDF, is a decision problem belonging to co-$\mathcal{NP}$.*

Proof: In order to show the problem to be in $\mathcal{NP}$, we have to prove that a task set can be decided infeasible using a polynomial time non-deterministic algorithm. If one "guesses" a time instant t, then for checking that task τ_i is not schedulable, it is necessary and sufficient to check there exist a time instant t such that : $dbf(t) > t$ (see [3] for the definition of the demand bound function $dbf(t)$). This is done in polynomial time since $dbf(t)$ is computable in linear time. Thus, a non-deterministic algorithm can check the infeasibility of a task system in polynomial time. Thus, the considered feasibility problem belongs to co-$\mathcal{NP}$. □

Algorithm 1: Linear Time Approximation Algorithm

```
Data    : $n, (\tau_1, \ldots, \tau_n)$
$\tilde{R}_1 = C_1$;
$r = s = 0$;
for i=2..n do
    $r = r + C_{i-1}/T_{i-1}$;
    $s = s + C_{i-1}$;
    $\tilde{R}_i = (s + C_i)/(1 - r)$;
end
return $(\tilde{R}_1, \ldots, \tilde{R}_n)$ ;
```

Next, we present several polynomial time algorithms to check the feasibility of fixed-priority tasks with constrained deadlines.

4.2 A linear time approximation

Consider the workload function stated in Equation 2 and let R_i^* be the exact worst-case response time of τ_i. In order compute an approximate worst-case response time (i.e., an upper bound), one can relax the integral values of t/T_j while computing the interference of any higher priority task τ_j. That is to say:

$$R_i^* \leq C_i + \sum_{j=1}^{i-1} \left(1 + \frac{R_i^*}{T_j}\right) C_j$$

For obtaining a lower bound of the worst-case response time of τ_i:

$$R_i^* \geq C_i + \sum_{j=1}^{i-1} \frac{R_i^*}{T_j} C_j$$

Using the two previous inequations, we obtain:

$$\frac{C_i}{1 - \sum_{j=1}^{i-1} \frac{C_j}{T_j}} \leq R_i^* \leq \frac{\sum_{j=1}^{i} C_j}{1 - \sum_{j=1}^{i-1} \frac{C_j}{T_j}} \qquad (9)$$

We use such an upper bound to approximate the worst-case response time of τ_i. Then, these upper bounds of tasks can be used to define a linear time feasibility algorithm (i.e., running in $O(n)$, where n is the number of tasks) that computes response time upper bounds $\tilde{R}_i$ as presented in Algorithm 1.

We first establish a negative result concerning the performance guarantees of Algorithm 1 while considering any possible task systems with constrained-deadline. Then, we shall show that under a simple assumption that Algorithm 1 has a bounded performance guarantee.

Theorem 4 *Let R_i^* be the exact worst-case response of τ_i and $\tilde{R}_i$ be the upper bound computed by Algorithm 1, then the ratio $\overline{R}_i/R_i^*$ is not bounded (i.e., Algorithm 1 has no performance guarantee).*

Proof : Consider the following task system with two tasks: $\tau_1(1-\epsilon, 1, 1)$ and $\tau_2(K\epsilon, K, K)$, where ϵ satisfies $0 < \epsilon < 1$ and K is an arbitrary integer number such that $K > 1$. Note that periods are proportional, thus a necessary and sufficient condition for the task system to be schedulable under the Rate Monotonic scheduling rule is $C_1/T_1 + C_2/T_2 \leq 1$. The utilization factor is:

$$U = \frac{C_1}{T_1} + \frac{C_2}{T_2} = (1-\epsilon) + \frac{K\epsilon}{K} = 1$$

Thus, the task system is schedulable under Rate Monotonic and the exact worst-case response times and those obtained by Algorithm 1 are:

$$\begin{aligned} R_1^* &= \overline{R}_1 = 1-\epsilon \\ R_2^* &= K \qquad\qquad \overline{R}_2 = \frac{1+(K-1)\epsilon}{\epsilon} \end{aligned}$$

Thus, the worst-case performance guarantee of Algorithm 1 is obtained while considering τ_2:

$$\lim_{\epsilon\to 0} \frac{\overline{R}_2}{R_2^*} = \lim_{\epsilon\to 0} \frac{1}{K\epsilon} + \frac{(K-1)}{K} = \lim_{\epsilon\to 0} \frac{1}{\epsilon} = \infty$$

$\square$

A similar result can be achieved for the performance guarantee of the lower bound $\tilde{R}_i$ that is defined by: $\max_{i=1\dots n} \frac{R_i^*}{\tilde{R}_i}$.

Theorem 5 *Let R_i^* be the exact worst-case response of τ_i and $\tilde{R}_i$ be the lower bound computed in Equation 9, then the ratio $R_i^*/\tilde{R}_i$ is not bounded (i.e., the lower bound has no performance guarantee).*

Proof : Consider the following task system with two tasks: $\tau_1(K, 2K, 2K)$ and $\tau_2(\epsilon, 2K, 2K)$, where ϵ satisfies: $0 < \epsilon < 1$ and K is an arbitrary number such that $K > 1$. Note that periods are equal. Thus, it is quite easy to see that the Rate Monotonic scheduling algorithm leads to a feasible schedule.

The exact worst-case response time for task τ_2 is $R_2^* = K + \epsilon$. And the lower bound defined by Equation 9 is:

$$\tilde{R}_2 = \frac{\epsilon}{1 - \frac{K}{2K}} = 2\epsilon$$

As a consequence, we verify:

$$\lim_{\epsilon\to 0} \frac{R_2^*}{\tilde{R}_2} = \lim_{\epsilon\to 0} \frac{K+\epsilon}{2\epsilon} = \infty$$

As a consequence, such a lower bound has no performance guarantee. $\square$

We now prove that if task parameters satisfy a simple condition, then Algorithm 1 is an approximation algorithm.

Theorem 6 *If we assume that there exists a constant K such that $K > \sum_i (C_i)/\min_i C_i$ for any task system, then Algorithm 1 has a performance guarantee not greater than K.*

Proof: Starting from equation 9:

$$\frac{C_i}{1 - \sum_{j=1}^{i-1} \frac{C_j}{T_j}} \leq R_i^* \leq \frac{\sum_{j=1}^{i} C_j}{1 - \sum_{j=1}^{i-1} \frac{C_j}{T_j}}$$

Thus,

$$\frac{\overline{R}_i}{R_i^*} \leq \frac{\sum_{j=1}^{i} C_j}{C_i} \leq K$$

Thus, under the assumption, Algorithm 1 is a K-approximation. $\square$

Thus, one can hope that Algorithm 1 is quite interesting for evaluating task systems having small tasks with similar worst-case execution times. For such systems, Algorithm 1 provides an efficient $O(n)$-time approximation algorithm for computing worst-case response times of tasks. But, when there are high variations on task lengths, then the algorithm cannot be efficient, since the constant K can be a huge number.

Using a similar argument, we define an assumption such that the lower bound defined in Equation 9 has a performance guarantee in comparison with exact worst-case response times of tasks.

Theorem 7 *If we assume that there exists a constant K such that $K > \sum_i (C_i)/\min_i C_i$ for any task system, the lower bound of the worst-case response time defined in Equation 9 has a performance guarantee not greater than K.*

Proof: As in the previous proof, starting from equation 9 we directly obtain:

$$\frac{R_i^*}{\tilde{R}_i} \leq \frac{\sum_{j=1}^{i} C_j}{C_i} \leq K$$

$\square$

We investigate next sections, two new approximation algorithms requiring more computational efforts (i.e., that are not running in linear time).

4.3 A deterministic approximation algorithm

The algorithm proposed by Joseph and Pandya [10] is based on computing the smallest fixed-point of Equation 3. The algorithm runs in pseudo-polynomial time since the number of iterations is not known to be bounded by a polynomial number in the task system size.

A simple way to achieve a bounded number of iterations is to stop computations at most after k iterations. If the smallest fixed-point is reached before k iterations then the algorithm returns the exact worst-case response times.

Algorithm 2: Deterministic Approximation Algorithm

```
Data    : $n, (\tau_1, \ldots, \tau_n)$, $\epsilon$
$(\overline{R}_1, \ldots, \overline{R}_n)$=Algorithm1$(n, (\tau_1, \ldots, \tau_n)$;
$k = \lceil \frac{1}{\epsilon} \rceil - 1$;
$r = s = 0$;
for $i$=2..$n$ do
    $l = 0$;
    $t = C_i$;
    while $(t < W_i(t)$ and $l < k$ and $t \le D_i)$ do
        $l = l + 1$;
        $t = W_i(t)$;
    end
    if $t = W_i(t)$ then
        $\overline{R}_i = t$;
    end
end
return $(\overline{R}_1, \ldots, \overline{R}_n)$ ;
```

Otherwise, it returns the upper bound presented in the previous section (i.e., using Algorithm 1):

$$\overline{R}_i = \frac{\sum_{j=1}^{i} C_j}{1 - \sum_{j=1}^{i-1} \frac{C_j}{T_j}}$$

The number k is a parameter that must be based on an *accuracy* constant: $\epsilon, 0 < \epsilon < 1$. As Fisher and Baruah, we define it as follows[1]:

$$k = \left\lceil \frac{1}{\epsilon} \right\rceil - 1$$

Algorithm 2 presents the pseudo-code of the deterministic approximation algorithm. In order to improve the algorithm efficiency, we first run Algorithm 1 that defines initial values of approximate worst-case response times. The algorithm runs in $O(\frac{n^2}{\epsilon})$ since the workload $W_i(t)$ is computed in $O(n)$.

As a direct consequence of the result presented in Theorem 4, we can establish that Algorithm 2 is an approximation algorithm under the following condition: there is a constant K such that $K > \sum_i (C_i)/\min_i C_i$ for any task system.

Theorem 8 *If we assume that there exists a constant K such that $K > \sum_i (C_i)/\min_i C_i$ for any task system, then Algorithm 2 has a performance guarantee not greater than K.*

4.4 A randomized approximation scheme

The last proposed algorithm is based on the Lehoczky, Sha and Ding's feasibility test. This algorithm checks the processor demand using a testing set S_i for any task τ_i. The size of such a set is not known to have a polynomial number of items. The feasibility test enumerates the testing set and stops when a time t that verifies $W_i(t)/t \le 1$. The worst-case behavior of such a test is achieved when all items in the testing set have been checked. The number of iterations for analyzing task τ_i is at most $\sum_{j=1}^{i} \left\lfloor \frac{D_i}{T_j} \right\rfloor$. From the implementation point of view, the order in which items in S_i's are enumerated is not important.

[1]We same the same definition of k in order to allow comparisons of algorithms in the Section dedicated to numerical experimentations.

Algorithm 3: Randomized Approximation Scheme

```
Data    : $n, (\tau_1, \ldots, \tau_n)$, $\epsilon$
$k = \lceil \frac{1}{\epsilon} \rceil - 1$;
$Feasible$=True;
$i = 1$;
while $i \le n$ and $Feasible$ do
    $f_i$=False;
    $j = 1$;
    while $j \le k$ and not $f_i$ do
        Choose randomly a time $t \in S_i$;
        if $\overline{W}_i(t) \le t$ then
            $f_i$=True;
        end
        $j = j + 1$;
    end
    $i = i + 1$;
    $Feasible = f_i$;
end
return Feasible ;
```

A simple way to define an approximation scheme based on the Lehoczky, Sha and Ding's exact feasibility test is to limit the size S_i while checking the feasibility of task τ_i. Once again, we fix such a number using an accuracy constant $\epsilon, 0 < \epsilon < 1$ as follows: $k = \lceil \frac{1}{\epsilon} \rceil - 1$.

In order to ensure the algorithm to be an approximation scheme we also have to use the approximate workload $\overline{W}_i(t)$ (i.e., Equation 7) rather than the exact workload $W_i(t)$ (i.e., Equation 2). If such a function is not used, we cannot ensure that the algorithm has competitive ratio bounded by a constant (i.e., to ensure that is an approximation algorithm). As a consequence, Algorithm 3 is a simple randomized version of the algorithm proposed in [9].

We define a randomized approximation scheme by enumerating randomly at most k items in each S_i with the same probability (i.e., a uniform law). While considering such items if no of them leads to a positive answer, then we state the task system to be infeasible. The corresponding algorithm has a computational complexity of $O(\frac{n^2}{\epsilon})$. If ϵ tends to 0, then the randomized approximation scheme has the same behavior than Lehoczky, Sha and Ding's exact feasibility test.

Algorithm Names	Authors
LSD89	Lehoczky, Sha and Ding, 1989
JP86	Joseph and Pandya, 1986
FB05	Fisher and Baruah, 2005
UB	Section 4.2
DET	Section 4.3
RAND	Section 4.4

Table 1. Algorithm name abbreviations used in the paper

5 Numerical results

We first describe the simulation environment and then numerical results.

5.1 Experimentation environment

We compared all presented methods (see Table 1 for the complete list). Task systems are randomly generated in order to achieved a given processor workload. The maximum worst-case execution time is fixed to 100 units of time and deadlines are constrained for all tasks (i.e., $D_i \leq T_i, 1 \leq i \leq n$). The simulator parameters are:

- the processor workload are 0.5 and 0.9,
- the number of tasks are between 2 and 50 tasks in every task systems,
- considered epsilon values are from 0.01 to 0.46 with a step 0.05 (Note that if $\epsilon \geq 0.5$, then $k = \lceil \frac{1}{\epsilon} \rceil - 1$ is always equal to 1).

For every value of these parameters, 25 task systems have been randomly generated and all methods have been run and compared. In the following, algorithms will be denoted as indicated in Table 1.

We only focus on two output parameters:

- the number of validated task systems,
- the number of iterations performed by the algorithms, which indicate the number of times that the workload function is computed during the test (i.e., Equation 2).

We are aware that simulation environment can have biasing effects on results [4], nevertheless every simulation results is always valid only within the confine of the stochastic model defined in the simulator. We note that results presented in the next section are valid for our simulation environment, and only for it.

5.2 Simulation Results

Figures 1 and 2 present numerical results for task systems having a processor utilization equal to 0.5. Figure 1 gives the number of validated task systems (i.e., the output status of the test is *feasible*). The algorithm

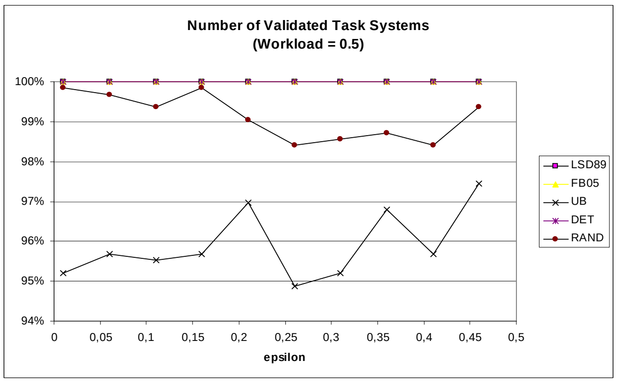

Figure 1. Number of validated task systems: all methods achieved good performances (Workload 0.5)

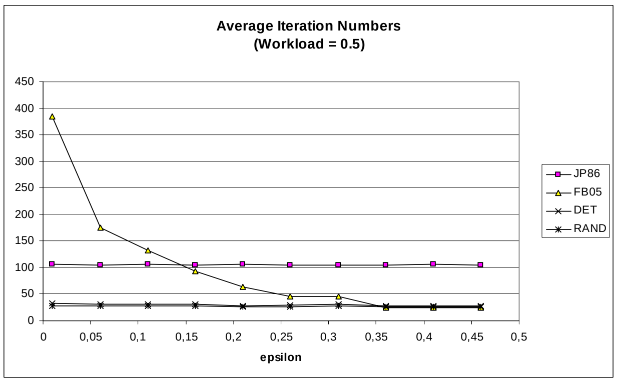

Figure 2. Iteration numbers according to epsilon values (Workload 0.5)

LSD89 is used as a reference. As we can see, all methods achieved good performances. More precisely, Figure 1 shows that LSD89, FB05 and DET have equivalent results. The randomized algorithm has lower performances in comparison with DET. The linear time approximation algorithm (based on the upper bound presented in Section 4.2) leads to acceptable results since in more than 94 percent it achieves a positive result (i.e., the same result than an exact feasibility test).

The average iteration number of JP86 remains constant for every epsilon value, because epsilon is not an input parameter for that algorithm. One can note that algorithm FB05 requires more iterations than JP86 for task sets when small epsilon values are considered. But, when epsilon is up to 0.25, then FB05 needs the same average iteration numbers than the other approximation algorithms and furthermore achieves better results.

Figures 3 and 4 present the same kink of results for a processor workload equal to 0.9. Clearly from Figure 3,

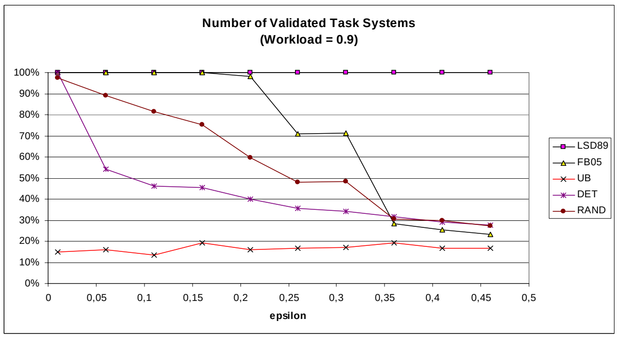

Figure 3. Number of validated task systems: all methods achieved good performances (Workload 0.9)

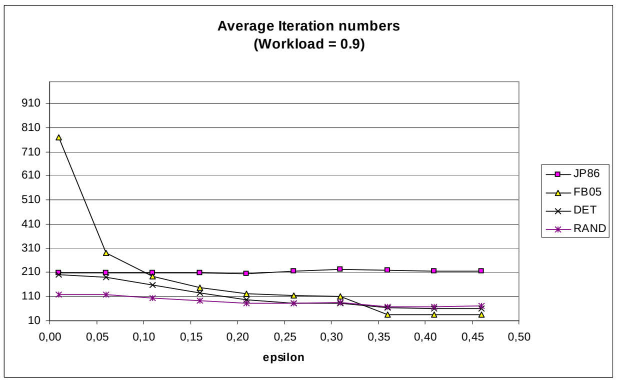

Figure 4. Iteration numbers according to epsilon values (Workload 0.9)

FB05 is more competitive in comparison to other approximation algorithms. But, when epsilon is up to 0.25 then its performance against an exact feasibility test decreases drastically to 30 percent of positive results. One can note that the linear approximation algorithm is not competitive enough when the processor utilization is high.

According to Figure 4, the average iteration numbers have slopes in comparison with numerical results achieved for a processor utilization equal to 0.5. Once again, FB05 becomes interesting for values around 0.25 since it requires the same average number of iterations and achieves better results.

As a conclusion, we must say that FB05 is better than the proposed polynomial time approximation algorithms for epsilon values near to 0.25. For optimizing the processor utilization, we conclude that FB05 is the better algorithm among those proposed here, but the accuracy parameter ϵ must be carefully chosen in order to control the quality of results. When the processor utilization is not high, then admission control can be efficiently done using the linear time approximation algorithm (denoted UB), that has been presented in Section 4.2.

6 Conclusion

Efficient feasibility tests are required for implementing an admission controller for large real-time systems. We focused on feasibility tests with a polynomial time complexity for defining efficient admission controllers. We presented computational complexity results and compared several approximate feasibility tests. We shown the checking the feasibility of tasks with constrained-deadlines belongs to $\mathcal{NP}$ when tasks have fixed-priorities, whereas the same problem with EDF belongs to co-$\mathcal{NP}$. We proposed three simple approximate algorithms and compared them with exact feasibility tests [10, 11] and one existing polynomial time approximation scheme [9].

Numerical results shown that if the processor utilization is not high, then admission control can be efficiently done in linear time. When the processor utilization increases, then we can use the Fisher and Baruah's fully polynomial time approximation scheme. According to our results, it could also interesting to evaluate exact feasibility tests since in many situations they can be as powerful than polynomial time approximation schemes even if their worst-case computational complexities lead to pseudo-polynomial time algorithms. But, there is still a small gap between polynomial time admission control and exact tests based pseudo-polynomial time algorithms.

The fully polynomial-time approximation scheme proposed in [9] is to decide if a given task system is feasible on a unit speed processor. But, it is not the case then the test ensures that the task system is infeasible upon a slower processor (the slowdown is related to the accuracy parameter). Thus, we want to use such techniques in order to define an efficient scheduling algorithm for tasks to be run upon a variable speed processor for power aware computer systems.

We must also conclude that the existence of approximation algorithms (or better approximation schemes) for computing worst-case response times of tasks is still an important open issue. Most of known papers do not cope with any performance guarantee in comparison with exact values of worst-case response time. Thus, we think that for most real-world systems validated with such schedulability tests lead to oversizing the real-time system features.

References

[1] K. Albers and F. Slomka. An event stream driven approximation for the analysis of real-time systems. *proc. Euromicro Int. Conf. on Real-Time Systems (ECRTS'04)*, pages 187–195, 2004.

[2] N. Audsley, A. Burns, M. Richardson, and A. Wellings. Hard real-time scheduling: the deadline monotonic approach. *proc. 8th IEEE Workshop on Real-Time Operating Systems and Software, Atlanta*, pages 127–132, 1991.

[3] S. K. Baruah, R. R. Howell, and L. E. Rosier. Algorithms and complexity concerning the preemptive scheduing of periodic, real-time tasks on one processor. *Real-Time Systems*, 2:301–324, 1990.

[4] E. Bini and G. Buttazzo. Biasing effects in schedulability measures. *Euromicro Int. Conf. on Real-Time Systems (ECRTS'04)*, 2004.

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

[6] 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.

[7] S. Chakraborty, S. Kunzli, and L. Thiele. Approximate schedulability analysis. *proc. 23rd Int. Symposium on Real-Time Systems (RTSS'02)*, 2002.

[8] N. Fisher and S. Baruah. A fully polynomial-time approximation scheme for feasibility analysis in static-priority systems with arbitrary relative deadlines. *proc. Euromicro Int. Conf. on Real-Time Systems (ECRTS'05)*, pages 117–126, July 2005.

[9] N. Fisher and S. Baruah. A polynomial-time approximation scheme for feasibility analysis in static priority systems with bounded relative deadlines. *proc. Real-Time and Embedded Systems (RTS'05), Paris*, 2005.

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

[11] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: exact characterization and average case behavior. *proc. Real-Time System Symposium (RTSS'98)*, pages 166–171, 1989.

[12] 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.

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