A Fully Polynomial-Time Approximation Scheme for Feasibility Analysis in Static-Priority Systems with Arbitrary Relative Deadlines*

Nathan Fisher and Sanjoy Baruah
Department of Computer Science
The University of North Carolina at Chapel Hill
{fishern, baruah}@cs.unc.edu

Abstract

Current feasibility tests for the static-priority scheduling on uniprocessors of periodic task systems run in pseudo-polynomial time. We present a fully polynomial-time approximation scheme (FPTAS) for feasibility analysis in static-priority systems with arbitrary relative deadlines. This test is an approximation with respect to the amount of a processor's capacity that must be "sacrificed" for the test to become exact. We show that an arbitrary level of accuracy, ϵ, may be chosen for the approximation scheme, and present a run-time bound that is polynomial in terms of ϵ and the number of tasks, n.

Keywords: *Real-time scheduling; Uniprocessor systems; Static-priority systems; Feasibility analysis; Fully polynomial-time approximation schemes.*

1 Introduction

An exact feasibility test for the preemptive uniprocessor scheduling of sets of periodic tasks, each with its deadline parameter equal to its period, was introduced by Lehoczky, Sha, and Ding [5]. The test determines whether a set of tasks is feasible using the *rate monotonic algorithm* [8]. The response time of each task is calculated, and checked against its deadline. Audsley *et al*. [2] developed a feasibility test for sets of tasks in which deadlines are less than periods, and which are scheduled using the *deadline monotonic algorithm* [7]. Lehoczky [6] provided a more general feasibility test for periodic task systems where the relation between deadlines and periods may be arbitrary.

In each of the aforementioned feasibility tests, the running time of the test is dependent on the values of the parameters of the tasks in the task system. Thus, these are *pseudo-polynomial time* tests. Albers and Slomka [1] present a *fully polynomial-time approximation scheme (FPTAS)* for feasibility of a sporadic task system using a dynamic-priority scheduling algorithm. The feasibility test accepts as input the specifications of a task system and a constant ϵ, $0 < \epsilon < 1$, and is an approximation scheme in the following sense:

> If the test returns "feasible", then the task set is guaranteed to be feasible on the processor for which it had been specified. If the test returns "infeasible", the task set is guaranteed to be infeasible *on a slower processor*, of computing capacity $(1 - \epsilon)$ times the computing capacity of the processor for which the task system had been specified.

In this paper, we extend the results of Albers and Slomka to the domain of static-priority scheduling with arbitrary relative deadlines. That is, we present an FPTAS for static-priority feasibility analysis that makes a performance guarantee similar to the one above: for any specified value of ϵ, the FPTAS correctly identifies, in time polynomial in the number of tasks in the task system, all task systems that are static-priority feasible (with respect to a given priority assignment) on a processor that has $(1-\epsilon)$ times the computing capacity of the processor for which the task system is specified. We have previously shown that such an FPTAS exists for static-priority systems when relative deadlines are bounded by periods [4].

Since many static-priority feasibility-analysis algorithms (in particular, those based upon iterative convergence of response-time equations) have been observed to converge extremely rapidly in practice, it may be argued that such an FPTAS is not particularly useful.

*Supported in part by the National Science Foundation (Grant Nos. ITR-0082866, CCR-0204312, and CCR-0309825).

However, the presence or otherwise of such an FPTAS is interesting from a theoretical perspective as part of the ongoing debate concerning the relative merits of static-priority and dynamic-priority scheduling: since an FPTAS was recently obtained for dynamic-priority uniprocessor feasibility-analysis, it is of interest to know whether static-priority feasibility-analysis could be approximated as efficiently as dynamic-priority analysis for arbitrary relative deadlines. The FPTAS presented in this paper answers this question in the affirmative.

The running time of current exact feasibility tests for static-priority task systems depends on the ratio between the largest and smallest period. Therefore, the complexity of current feasibility tests for task systems with widely-varying periods may prohibit their use in automatic synthesis tools. The running time of the approximation proposed in this paper is completely independent of tasks' periods, and depends only on the number of tasks and the accuracy constant, ϵ. Thus, the approximation offers a reduction in complexity for many task sets, and its predictable worst-case run-time guarantees a quick estimate for automatic synthesis tools exploring a real-time system design space.

The remainder of this paper is organized as follows. We formally define our task model in the next section. We briefly summarize (Section 3.1) how the *request-bound function* abstraction, which plays a crucial role in the various static-priority feasibility tests mentioned above [5, 2, 6], can be approximated by a function that is easily computed, and which satisfies the property that its value "closely" tracks the exact value of the request-bound function. We review the FPTAS and some results from [4] in Section 3.2. We give an approximate test for a task system with arbitrary relative deadlines in Section 4. We prove the correctness of the approximation test for arbitrary deadlines in Section 5. Finally, in Section 6, we formally state the main result of this paper, the existence of an FPTAS for feasibility in static-priority systems.

2 Task Model

We consider both *periodic* and *sporadic* task models. A *task* $\tau_i = (e_i, d_i, p_i)$ is characterized by a *worst-case execution requirement* e_i and *(relative) deadline* d_i. In the periodic task model, the *period* p_i represents the exact interval between the arrival of jobs of τ_i. In the sporadic task model, p_i represents the *minimum inter-arrival separation* between jobs of τ_i. Each job has a worst-case execution requirement equal to e_i and a deadline that occurs d_i time-units after its arrival time. A task system τ is composed of tasks $\tau_1, \ldots, \tau_n$, where n is the number of tasks in the system. A periodic task system is *synchronous* if the first job of every task is released at the same time.

A *feasibility test* is a necessary and sufficient set of conditions for determining whether a given task system will meet all of its deadlines. In the remainder of this paper, we study approximation schemes for feasibility of both sporadic and synchronous periodic task systems that are to be executed on a preemptive uniprocessor platform.

Static-Priority Scheduling Algorithms

In static-priority systems, each task is assigned a distinct priority, and all jobs of a task execute at the task's priority. More formally, a job is said to be *active* at a specified time-instant in a schedule, if it has remaining execution time and has not missed its deadline. When a scheduling algorithm is invoked at time t, it will select the job with highest priority out of the set of active jobs at time t. Two well-studied static-priority scheduling algorithms are *rate monotonic*(RM) and *deadline monotonic*(DM). RM, introduced in [8], assigns each task a priority equal to the inverse of its period. DM, first presented in [7], assigns each task a priority equal to the inverse of its relative deadline. We will assume throughout this paper that tasks are indexed according to their assigned priority (i.e. for $1 \leq i < n$, τ_i has higher priority than τ_{i+1}).

A task system τ is *feasible* with respect to static-priority systems if there exists a task priority assignment such that when τ is scheduled according to this priority assignment, all deadlines are met. A static-priority scheduling algorithm A is *optimal over all static-priority algorithms*, if for every feasible task system τ, A produces a schedule in which all deadlines are met. RM is known to be optimal for static-priority algorithms when deadlines are equal to periods. DM is optimal for static-priority algorithms when deadlines are less than or equal to periods. Shih *et al*. [9] give a modified rate monotonic algorithm that is optimal for special cases when deadlines can exceed periods; however, to the best of our knowledge, there is currently no known optimal static-priority scheduling algorithm for arbitrary relative deadlines.

3 Bounded Relative Deadlines

In this section, we present the approximate feasibility test for static-priority systems, developed in [4], where each task's relative deadline is constrained to be at most its period. We begin in Section 3.1 by defining a *request-bound function* (RBF) that bounds the amount of execution time requested by a task (similarly defined in [5, 2, 6]). An approximation to the RBF is defined such that the deviation from the RBF is bounded.

In Section 3.2, we define both exact and approximate *cumulative request-bound functions* based, respectively, on the exact and approximate request-bound functions for a task τ_i. The functions describe the cumulative execution requests over a time interval for task τ_i and all tasks of higher priority. Lehoczky *et al.* [5] showed that in a sporadic or synchronous periodic system, the *smallest* fixed point of the exact cumulative request-bound function for task τ_i is the time at which the processor can satisfy τ_i's request. We assume that DM is used to schedule the tasks. If the smallest fixed point of task τ_i's cumulative request-bound function is no larger than its relative deadline, then τ_i will always meet its deadline. If the smallest fixed point exceeds τ_i's deadline, then we cannot guarantee τ_i will meet all deadlines; hence, τ is not feasible.

3.1 Request-Bound Function

In a periodic synchronous task system, the total execution time requested by a task τ_i can be expressed as a function of time. Every time a task τ_i releases a job, e_i additional units of processor time are requested. The following function provides an upper bound on the total execution time requested by task τ_i over time interval $[0, t]$:

$$\text{RBF}(\tau_i, t) \stackrel{\text{def}}{=} \left\lceil \frac{t}{p_i} \right\rceil e_i \qquad (1)$$

In a sporadic task system, $\text{RBF}(\tau_i, t)$ represents the total execution time requested by τ_i in its worst-case phasing. Figure 1 shows an example of a RBF. Notice that the "step" function, $\text{RBF}(\tau_i, t)$ increases by e_i units every p_i time units.

Approximating the RBF

The function $\text{RBF}(\tau_i, t)$ has a discontinuity every p_i time units. We call these discontinuities *steps*. We define an approximation that computes the first $(k - 1)$ steps of $\text{RBF}(\tau_i, t)$ exactly (where k is a constant, defined below), and is a linear approximation of $\text{RBF}(\tau_i, t)$, thereafter.

We choose a constant k based on our given "accuracy" constant ϵ, $0 < \epsilon < 1$. For the remainder of the paper, assume the integer constant k is defined as follows:

$$k \stackrel{\text{def}}{=} \lceil 1/\epsilon \rceil - 1 \qquad (2)$$

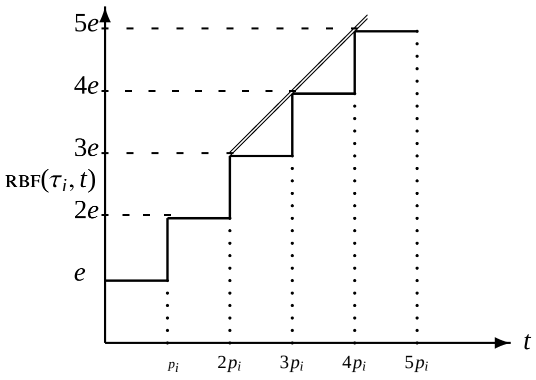

Figure 1. The step function denotes a plot of $\text{RBF}(\tau_i, t)$ as a function of t. The double line represents the function $\delta(\tau_i, t)$, approximating $\text{RBF}(\tau_i, t)$, where $k = 3$; for $t \leq 2p_i$, $\delta(\tau_i, t) \equiv \text{RBF}(\tau_i, t)$.

We now define the following function $\delta(\tau_i, t)$ which closely approximates the function $\text{RBF}(\tau_i, t)$:

$$\delta(\tau_i, t) = \begin{cases} \text{RBF}(\tau_i, t)\,, & \text{for } t \leq (k-1)p_i \\ e_i + \frac{te_i}{p_i}\,, & \text{for } t > (k-1)p_i \end{cases} \qquad (3)$$

Figure 1 shows that $\delta(\tau_i, t)$ is exactly $\text{RBF}(\tau_i, t)$ up to $t = (k-1)p_i$, (in this example, $k = 3$) and then is a linear approximation for $t > (k-1)p_i$ that "bounds" $\text{RBF}(\tau_i, t)$ from above.

3.2 Description of Feasibility Test

Exact Test

For static-priority task systems with relative deadlines bounded by periods, Liu and Layland [8] showed that the *worst-case* response time for a job of task τ_i occurs when all tasks of priority greater than τ_i release a job simultaneously with τ_i. If a task τ_i releases a job J simultaneously with all higher priority tasks and each higher priority task τ_j releases subsequent jobs at the earliest legal opportunity (i.e. the inter-arrival separation between jobs of higher-priority task τ_j is *exactly* p_j), then J has the largest response time of any job of task τ_i. In a sporadic or synchronous periodic task system with relative deadlines bounded by periods, it is necessary and sufficient to only check the response time of the first job of each task. If the response time of the first job of task τ_i is at most its relative deadline, then τ_i is schedulable; else, it is not schedulable. A task system τ is feasible on a uniprocessor *if and only if* the first job of each task τ_i has a worst-case response time at most d_i.

In order to determine the response-time for the first job of task τ_i, we must consider execution requests of τ_i and all jobs of tasks which may preempt τ_i. We define the following *cumulative request-bound function* based on RBF. Let $\mathbf{T}_{-i}$ be the set of tasks with priority greater than τ_i. Then, the cumulative request-bound function is defined as:

$$W_{i,\ell}(t) \stackrel{\text{def}}{=} \ell e_i + \sum_{\tau_j \in \mathbf{T}_{-i}} \text{RBF}(\tau_j, t) \quad (4)$$

The cumulative request-bound function $W_{i,\ell}(t)$ is simply the total execution requests of all tasks of higher priority than τ_i over the interval $(0, t]$, and the execution request of the first ℓ jobs of τ_i. When deadlines do not exceed periods, we are concerned only with $W_{i,1}(t)$ which is the cumulative request-bound function for the first job of τ_i.

Audsley *et al.* [3] presented an exact feasibility test for task τ_i using DM: a task is feasible if and only if there exists a fixed point, t, of $W_{i,1}(t)$ such that t occurs before τ_i's deadline . The following theorem restates their test:

Theorem 1 (from [3]) *In a synchronous periodic (or sporadic) task system, task τ_i is feasible using* DM *if and only if $\exists t \in (0, d_i]$ such that $W_{i,1}(t) \le t$.*

■

Approximate Test

The goal of using a linear approximation in $\delta(\tau_i, t)$ is to bound the number of steps in the approximation function. Since $\delta(\tau_i, t)$ has at most $k - 1$ steps for all τ_i, a *superposition* of $\delta(\tau_i, t)$'s (i.e. a summation of a number of different δ functions) will have a polynomially bounded number of steps in terms of k and the number of functions in the superposition. The following equation defines a superposition which we will use as the approximate cumulative request-bound function for the approximate feasibility test:

$$\widehat{W}_{i,\ell}(t) \stackrel{\text{def}}{=} \ell e_i + \sum_{\tau_j \in \mathbf{T}_{-i}} \delta(\tau_j, t) \quad (5)$$

The following lemmas describe the implications of using the approximation function δ in the request-bound function. Informally, the first Lemma 1 states that if the approximate cumulative request-bound function is below line $f(t) = t$, then the exact cumulative request-bound function must be below as well.

Lemma 1 (from [4]) *If $\widehat{W}_{i,\ell}(t) \le t$, then $W_{i,\ell}(t) \le t$.*

Lemma 2 states that if the approximate cumulative request-bound function lies above $f(t) = t$, then the exact cumulative request-bound function must lie above the line $f(t) = \frac{k}{1+k}(t)$. Formally stated:

Lemma 2 (from [4]) *If $\widehat{W}_{i,\ell}(t) > t$, then $W_{i,\ell}(t) > \frac{k}{1+k}(t)$.*

The value of $\widehat{W}_{i,\ell}(t)$ is always at least that of $W_{i,\ell}(t)$. Therefore, if we use the approximate request-bound function, $\widehat{W}_{i,\ell}(t)$ to find a fixed-point as in Theorem 1, the test is no longer necessary and sufficient. Instead, we will have a sufficient test for feasibility tests, as the following theorem states:

Theorem 2 (from [4]) *A synchronous periodic (or sporadic) task system, task τ_i is feasible using* DM *if $\exists t \in (0, d_i]$ such that $\widehat{W}_{i,1}(t) \le t$.*

■

Using the approximate response time function also no longer gives an exact check for infeasibility. Instead, if we cannot find a $t \in (0, d_i]$ such that $\widehat{W}_{i,1}(t) \le t$, then τ_i is infeasible on a lower capacity processor. In fact, we can quantify a smaller capacity processor for which this approximate feasibility test would become exact. The following theorem quantifies this capacity:

Theorem 3 (from [4]) *If $\forall t \in (0, d_i]$, $\widehat{W}_{i,1}(t) > t$, then τ_i is infeasible using* DM *on a processor of $(1 - \epsilon)$ capacity.*

■

The preceding theorem states we must effectively ignore $(1 - \epsilon)$ of the processor capacity for the test to become exact.

Together, theorems 2 and 3 provide an approximate test for feasibility of task system τ. The time complexity of the test is $O(n^2 k)$.

4 Arbitrary Relative Deadlines

When deadlines can exceed periods, Lehoczky [6] shows that it is no longer sufficient to check the response-times of only the first job of each task. Instead, it is potentially necessary to check the response-time of all jobs in the *level-i busy interval* for each task τ_i . A level-i busy interval is a time interval $[a, b]$ where only jobs of $\mathbf{T}_i = \mathbf{T}_{-i} \cup \{\tau_i\}$ are executing continuously and the following is true:

1. A job of $\mathbf{T}_i$ is released at time a.
2. All jobs of $\mathbf{T}_i$ released prior to a have completed by time a.
3. b is the first time instant such that all jobs of $\mathbf{T}_i$ released in the interval $[a, b)$ have completed.

It may be tempting to try extending our results to a task system with arbitrary deadlines by applying the approximate feasibility test presented in this paper to each job of τ_i in the level-i busy interval. Unfortunately, if this approach is used the approximate feasibility test is no longer polynomial in terms of n and ϵ. Applying the approximate feasibility test to each job of τ_i in the level-i busy interval results in a *pseudo-polynomial* time test. The reasons that the test is pseudo-polynomial are the following:

- The length of the level-i busy interval does not depend on n, but on the p_i and e_i terms; therefore, the level-i busy interval contains a pseudo-polynomial number of jobs of τ_i. Applying the approximate feasibility test of the previous section would require running the test a pseudo-polynomial number of times.
- The number of jobs of a task τ_i that are active at each time instant t (i.e. t lies between the job's release time and absolute deadline) could be $\Theta(d_i/p_i)$. Again, this is not polynomial in terms of n and ϵ. Therefore, at each point in the testing set, we may have to perform a computation for a pseudo-polynomial number of active jobs to check if any of them have missed their deadline.

In this section, we show that pseudo-polynomial time checks are not required. We construct an FPTAS for feasibility analysis in static-priority systems with arbitrary relative deadlines given an arbitrary priority assignment. Furthermore, our FPTAS for arbitrary relative deadlines achieves the same asymptotic time complexity as the FPTAS for bounded relative deadlines. We define an algorithm that determines (according to the approximation functions defined in Section 3.2) the set of jobs of τ_i that complete prior to or at time $t = \max_{j \in \{1,\ldots,i\}}\{(k-1)p_j\}$ (the point after which the approximation becomes a linear function), and meet their deadlines, in Section 4.1.1.

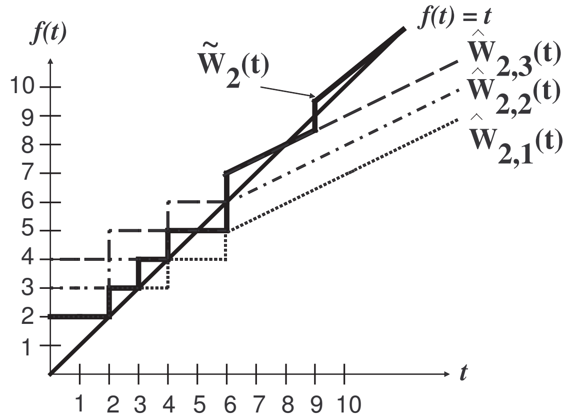

Figure 2. Examples of $\widetilde{W}_2(t)$ and $\widehat{W}_{i,\ell}(t)$ functions for system with $\tau_1 = (p_1, e_1, d_1) = (2, 1, 2)$ and $\tau_2 = (p_2, e_2, d_2) = (3, 1, 4)$, and $k = 4$. The approximation function $\widehat{W}$ for the first three jobs of task τ_2 is shown along with the $\widetilde{W}$ function.

Section 4.1.2 describes a test to approximate the set of jobs that complete after time $\max_{j \in \{1,\ldots,i\}}\{(k-1)p_j\}$ and meet their deadline. We provide a proof sketch for the correctness of the feasibility approximation in Section 5. We derive the running time of approximation in Section 5.1. For the lemmas and theorems of the following section, we provide a proof sketch and intuition. The extended version of this paper will include full proofs for each of these lemmas and theorems.

4.1 Feasibility Test

4.1.1 Jobs with completion time prior or equal to $\max_{j \in \{1,\ldots,i\}}\{(k-1)p_j\}$

We define the following function used to determine the set of jobs that have their execution requests satisfied by time t:

$$\widetilde{W}_i(t) \stackrel{\text{def}}{=} \delta(\tau_i, t) + \sum_{\tau_j \in \mathbf{T}_{-i}} \delta(\tau_j, t) \tag{6}$$

$\widetilde{W}_i(t)$ represents a "close" approximate to the cumulative requests of task τ_i and all higher priority tasks with respect to *any* given time t. In comparison, $\widehat{W}_{i,\ell}(t)$ is only a "close" approximation when t lies in the interval $((\ell-1)p_i, \ell p_i]$. An example of $\widetilde{W}_i(t)$ and $\widehat{W}_{i,\ell}(t)$ functions is illustrated in Figure 2.

Notice that the number of *active* jobs at time t could be $\Theta(d_i/p_i)$. The next function is used to identify the index of the most recently released job of τ_i to have its execution request satisfied by time t.

$$Z_i(t) \overset{\text{def}}{=} \max\left(\left\lceil \frac{\widetilde{W}_i(t) - t}{e_i} \right\rceil, 0\right) \tag{7}$$

The index of the most recently released job to have its execution request satisfied by time t is $\left\lceil \frac{t}{p_i} \right\rceil - Z_i(t)$ (according to our approximation). By finding the index of the most recently released job of τ_i to complete execution with respect to time t, we can determine in constant time the set of active jobs that have their execution requests satisfied at or prior to t.

Lemma 3 *If $\left\lceil \frac{t}{p_i} \right\rceil - Z_i(t) \geq 1$ and $t \leq (k-1)p_i$, then for $\ell \in \{1, \ldots, \left\lceil \frac{t}{p_i} \right\rceil - Z_i(t)\}$ the ℓ^{th} job has its request satisfied by t (i.e. $W_{i,\ell}(t) \leq t$).*

Proof Sketch: Let $b = \left\lceil \frac{t}{p_i} \right\rceil - Z_i(t)$. There are two cases:

1. $Z_i(t) > 0$
2. $Z_i(t) = 0$

In both cases, we can derive the inequality $\widehat{W}_{i,b}(t) \leq t$, and the lemma follows. ■

The next lemma shows: if the ℓ^{th} job of τ_i is active at time t, and its index exceeds $\left\lceil \frac{t}{p_i} \right\rceil - Z_i$, then the ℓ^{th} job does not complete before or at time t. Using this lemma we can determine the set of jobs of τ_i that do not have their execution request satisfied by time t.

Lemma 4 *If $\ell \geq \left\lceil \frac{t}{p_i} \right\rceil - Z_i(t)$ and $t > (\ell - 1)p_i$, then $\widehat{W}_{i,\ell}(t) > t$.*

Proof Sketch: Suppose that for job $a = \left\lceil \frac{t}{p_i} \right\rceil - Z_i(t) + 1$, $\widehat{W}_{i,a}(t) \leq t$. Then, the difference between $\widetilde{W}_i(t)$ and t is less than $\widetilde{W}_i(t) - \widehat{W}_{i,a-1}(t)$ (since $t \geq \widehat{W}_{i,a}(t) > \widehat{W}_{i,a-1}(t)$). We can show $Z_i(t)$ can be expressed by both Equation (7) and $\frac{\widetilde{W}_i(t) - \widehat{W}_{i,a-1}(t)}{e_i}$. This will imply the following equality $\widetilde{W}_i(t) - \widehat{W}_{i,a-1}(t) = \widetilde{W}_i(t) - t$; however, this contradicts the fact that $t > \widehat{W}_{i,a-1}(t)$ is a strict inequality. ■

We now define, for a given task τ_i, the set of points that must be tested in our approximation as:

$$\widetilde{S}_i \overset{\text{def}}{=} \{t = bp_a : a = 1, \ldots, i; b = 1, \ldots, k-1\} \cup \{0\} \tag{8}$$

We call two elements t_1 and t_2 ($t_1 < t_2$) in set $\widetilde{S}_i$ *adjacent* if no t satisfying $t_1 < t < t_2$ is in $\widetilde{S}_i$. Observe that for any two adjacent elements $t_1, t_2 \in \widetilde{S}_i$, $|t_2 - t_1| \leq p_i$. Therefore, at most one job of τ_i can have its deadline occur between any two adjacent elements of $\widetilde{S}_i$. Using this observation and Lemmas 3 and 4, we can construct an algorithm which determines, for all $t \in \widetilde{S}_i$, which jobs of τ_i have their execution requests satisfied at or prior to time t. Also, we can check in constant time whether the processor meets the execution requests of any job whose deadline has elapsed since the prior adjacent element in $\widetilde{S}_i$. Figure 3 describes this algorithm ApproxFirstStage in greater detail. Section 5 will prove the correctness of ApproxFirstStage.

ApproxFirstStage(τ, i, k):

Step 0: Construct an ordered set $\widetilde{S}_i$ as in Equation (8).

Step 1: Initialize variable *lowest_active* to 1.

Step 2: For each $t_a \in \widetilde{S}_i - \{0\}$:

a) If $t_a > (\textit{lowest\_active} - 1)p_i + d_i$ then:

   i) Let t_{a-1} be the adjacent element prior to t_a in ordered set $\widetilde{S}_i$. Let y be the total execution requirement of all jobs released at time t_a. Determine where the line defined by $(t_{a-1}, \widehat{W}_{i,\,\textit{lowest\_active}}(t_{a-1}) + y)$ and $(t_a, \widehat{W}_{i,\,\textit{lowest\_active}}(t_a))$ intersects with $f(t) = t$. Let this point of intersection be t'.

   ii) If $t' > (\textit{lowest\_active} - 1)p_i + d_i$, then return "$\tau_i$ *is not schedulable*"; otherwise, increment *lowest_active*.

b) Let $x := \max\left(0, \left\lceil \frac{t_a}{p_i} \right\rceil - Z_i(t)\right)$.

c) Let *lowest_active* := $\max(x + 1, \textit{lowest\_active})$.

Step 3: Return *lowest_active*.

Figure 3. The function ApproxFirstStage(τ, i, k) **determines whether all jobs of task τ_i with deadlines less than $\max_{j \in \{1,\ldots,i\}}\{(k-1)p_j\}$ are schedulable. If no deadlines are missed up to time $\max_{j \in \{1,\ldots,i\}}\{(k-1)p_j\}$** ApproxFirstStage **returns the lowest indexed job whose execution request is not satisfied by time $\max_{j \in \{1,\ldots,i\}}\{(k-1)p_j\}$. Otherwise, it returns τ_i** *not schedulable***.**

4.1.2 Jobs with completion time after $\max_{j\in\{1,\ldots,i\}}\{(k-1)p_j\}$

Next, we describe a constant time test for the set of jobs of task τ_i that have deadlines after time $\max_{j\in\{1,\ldots,i\}}\{(k-1)p_j\}$ and ApproxFirstStage does not determine that their demand is satisfied prior to or at time $\max_{j\in\{1,\ldots,i\}}\{(k-1)p_j\}$. Notice from the definition of $\widehat{W}_{i,\ell}$,

$$\forall \ell \in \mathbb{N}, \forall t \in \left(\max_{j\in\{1,\ldots,i\}}\{(k-1)p_j\}, \infty\right) :: \left(\widehat{W}_{i,\ell}(t) = \ell e_i + \sum_{\tau_j \in \mathbf{T}_{-i}} \left(e_j + \frac{te_j}{p_j}\right)\right) \quad (9)$$

Let us assume that ApproxFirstStage(τ, i, k) returns h. This means that h is the lowest indexed job of τ_i that according to the approximation has not had its execution request satisfied by time $\max_{j\in\{1,\ldots,i\}}\{(k-1)p_j\}$. Then for all $\ell \in \mathbb{N}(\ell \geq h)$, we can solve equation (9) to find point t_ℓ at which $\widehat{W}_{i,\ell}(t_\ell) = t_\ell$.

$$t_\ell \stackrel{\text{def}}{=} \frac{\ell e_i}{1 - U_{-i}} + \frac{\sum_{\tau_j \in \mathbf{T}_{-i}} e_j}{1 - U_{-i}} \quad (10)$$

where $U_{-i} \stackrel{\text{def}}{=} \sum_{\tau_j \in \mathbf{T}_{-i}} \frac{e_j}{p_j}$. From Lemma 1, we know that $W_{i,\ell}(t_\ell) \leq t_\ell$. Intuitively, t_ℓ represents the time at which the approximation determines that the processor can satisfy the execution requests of the ℓ^{th} job of task τ_i. Therefore, if

$$t_h \leq (h-1)p_i + d_i, \quad (11)$$

then the h^{th} job of τ_i meets its deadline. Otherwise, we declare τ_i to be *not schedulable*.

If Inequality (11) is true, we must then determine whether all subsequent jobs of τ_i after h meet their deadlines. This is equivalent to determining whether $\forall \ell \in \mathbb{N}(\ell > h)$, $t_\ell \leq (\ell-1)p_i + d_i$. Define

$$\Delta_\ell \stackrel{\text{def}}{=} [(\ell-1)p_i + d_i] - \left[\frac{\ell e_i}{1 - U_{-i}} + \frac{\sum_{\tau_j \in \mathbf{T}_{-i}} e_j}{1 - U_{-i}}\right]. \quad (12)$$

Δ_ℓ represents the difference between t_ℓ and the deadline for the ℓ^{th} job of task τ_i. The following lemma quantifies this difference in terms of Δ_h.

Lemma 5 *$\forall \ell (\in \mathbb{N}) \geq h$, $\Delta_\ell = \Delta_h - (\ell - h)\left(\frac{e_i}{1-U_{-i}} - p_i\right)$.*

Proof Sketch: The proof is easily shown by induction on ℓ. ■

Using the previous lemma, we can show that if the h^{th} job of task τ_i meets its deadline, then all subsequent jobs of τ_i will meet their deadlines *if and only if* $\frac{e_i}{1-U_{-i}} > p_i$. The next lemma formalizes this statement.

Lemma 6 *Given that $\Delta_h \geq 0$, then $\exists \ell \in \mathbb{N}$ $(\ell > h)$ such that $t_\ell > (\ell-1)p_i + d_i$ if and only if $\frac{e_i}{1-U_{-i}} > p_i$.*

Proof: We will prove the *only if* part, first. Assume that $t_\ell > (\ell-1)p_i + d_i$ and $\ell > h$. Notice from equation (10), $t_\ell - t_h = (\ell - h)\frac{e_i}{1-U_{-i}}$. By equation (11),

$$\begin{aligned} t_\ell - t_h &\geq t_\ell - [(h-1)p_i + d_i] \\ &> (\ell-1)p_i + d_i - (h-1)p_i - d_i \\ &\quad \text{(from the assumption on } t_\ell) \\ &= (\ell - h)p_i. \end{aligned}$$

Observe $t_\ell - t_h = (\ell-h)\frac{e_i}{1-U_{-i}}$. So $(\ell-h)\frac{e_i}{1-U_{-i}} > (\ell-h)p_i$, which implies $\frac{e_i}{1-U_{-i}} > p_i$.

Now proving the *if* direction, assume that $\frac{e_i}{1-U_{-i}} > p_i$. Define ℓ as follows:

$$\ell = \left\lceil \frac{\Delta_h}{\left(\frac{e_i}{1-U_{-i}}\right) - p_i} \right\rceil + 1 + h$$

Obviously, $\ell > h$. We will show that for the ℓ^{th} job of task τ_i, $t_\ell > (\ell-1)p_i + d_i$.

$$\begin{aligned} \Delta_\ell &= \Delta_h - (\ell - h)\left(\tfrac{e_i}{1-U_{-i}} - p_i\right) \quad \text{(from Lemma 5)} \\ &= \Delta_h - \left(\left\lceil \frac{\Delta_h}{\left(\frac{e_i}{1-U_{-i}}\right) - p_i} \right\rceil + 1\right)\left(\tfrac{e_i}{1-U_{-i}} - p_i\right) \quad \text{(from definition of } \ell) \\ &\leq \Delta_h - \left(\frac{\Delta_h}{\left(\frac{e_i}{1-U_{-i}}\right) - p_i} + 1\right)\left(\tfrac{e_i}{1-U_{-i}} - p_i\right) \\ &= \Delta_h - \Delta_h - \left(\tfrac{e_i}{1-U_{-i}} - p_i\right) \\ &< 0 \quad \text{(from assumption)} \end{aligned}$$

$\Delta_\ell < 0$ implies $(\ell-1)p_i + d_i - t_\ell < 0$. Thus, $t_\ell > (\ell-1)p_i + d_i$.
■

We have shown that we can check the approximate feasibility of the h^{th} job and all subsequent jobs of task τ_i by testing inequality (11) and checking that $\frac{e_i}{1-U_{-i}} \leq p_i$. Figure 4 gives the pseudo-code for the algorithm ApproxSecondStage. Finally, Figure 5 describes the full approximation scheme for feasibility of synchronous periodic or sporadic static-priority task systems with respect to a given priority assignment.

5 Proof of Correctness for Arbitrary Deadlines

In this section, we will give a proof sketch of correctness for Approx. The goal is to show that:

If Approx(τ, ϵ) returns "feasible", then the task

ApproxSecondStage(τ, i, k, h):

Step 0: Set $t_h := \frac{he_i}{1-U_{-i}} + \frac{\sum_{\tau_j \in \mathbf{T}_{-i}} e_j}{1-U_{-i}}$.

Step 1: If $t_h > (h-1)p_i + d_i$, return τ_i *is not schedulable*.

Step 2: If $\frac{e_i}{1-U_{-i}} > p_i$, return τ_i *is not schedulable*.

Step 3: Return τ_i *is schedulable*.

Figure 4. The function ApproxSecondStage(τ, i, k, h) **determines whether the h^{th} job and all subsequent jobs of task τ_i are schedulable. If so, it returns τ_i *schedulable*, else it returns τ_i *not schedulable*.**

set is guaranteed to be feasible on the processor for which it had been specified. If Approx(τ, ϵ) returns "infeasible", the task set is guaranteed to be infeasible *on a slower processor*, of computing capacity $(1-\epsilon)$ times the computing capacity of the processor for which the task system had been specified.

This section is organized as follows: We first specifiy an invariant of ApproxFirstStage. The invariant quantifies the set of jobs of task τ_i that are guaranteed, according to ApproxFirstStage, to have their execution request met prior to their respective deadlines. We show (Lemma 9) that Approx will return "τ is feasible" *if and only if* the approximate cumulative request-bound function for all jobs of all tasks is satisfied prior to each job's deadline. We then invoke a result from Lehoczky [6] to arrive at the stated goal, above.

The next lemma states the invariant that identifies the set of jobs that are schedulable according to each iteration of ApproxFirstStage. Let $lowest\_active_a$ be the value of *lowest_active* prior to the a^{th} iteration of the *for* loop of ApproxFirstStage.

Lemma 7 *After $a-1$ iterations and prior to the a^{th} iteration of the* for *loop of* ApproxFirstStage*, the following condition holds:*

$$\forall \ell \in \{1, \ldots, lowest\_active_a - 1\} :: \\ (\exists t \in ((\ell-1)p_i, (\ell-1)p_i + d_i] : \widehat{W}_{i,\ell}(t) \leq t)$$

Proof Sketch: We can show the invariant by induction. Observe that the base case is vacuously true. Notice that at each step of the algorithm, the variable *lowest_active* can either increase or remain the same as the previous iteration. If the value of *lowest_active* remains the same, then by the inductive hypothesis the invariant still holds. However, if the value of *lowest_active* increases, the invariant holds due to Lemma 3. ■

Approx(τ, ϵ):

Step 0: Initialize variables h to zero, and $k := \lceil 1/\epsilon \rceil - 1$.

Step 1: For each $\tau_i \in \tau$:

a) If ApproxFirstStage(τ, i, k) does not return "τ_i is not schedulable", then set $h :=$ ApproxFirstStage(τ, i, k). Else return τ *is infeasible*.

b) If ApproxSecondStage(τ, i, k, h) returns "τ_i is not schedulable", then return τ *is infeasible*.

Step 2: Return τ *is feasible*.

Figure 5. The function Approx(τ, ϵ) **determines whether the task system τ is feasible. If** Approx **returns τ *is feasible*, then τ is guaranteed to be feasible on a processor of unit capacity. Otherwise, if** Approx **returns τ *is infeasible*, then τ is guaranteed to be infeasible on a processor of $(1-\epsilon)$ capacity.**

In [6], Lehoczky showed: if for each job j of task τ_i there exists a time t between the release and deadline of j such $W_{i,j}(t) \leq t$, then τ_i is schedulable. We will use this result to show that the task set τ is feasible when Approx(τ, ϵ) returns "τ is feasible," and τ is infeasible on a processor of $(1-\epsilon)$ capacity when Approx(τ, ϵ) returns "τ is infeasible." We restate Lehoczky's results in the following theorem.

Theorem 4 (from [6]) *A sporadic or synchronous periodic task system τ is feasible* if and only if $\forall \tau_i \in \tau$, $\ell(> 0) \in \mathbb{N}$, $\exists t \in ((\ell-1)p_i, (\ell-1)p_i + d_i]$ *such that* $W_{i,\ell}(t) \leq t$.

■

Before proving that Approx correctly identifies the feasible tasks, we will restate the following result from [4] used in the proof of Lemma 9:

Lemma 8 (from [4]) *For adjacent elements, $t_1, t_2 \in \widetilde{S}_i$, if $\widehat{W}_i(t_1) > t_1$ and $\widehat{W}_i(t_2) > t_2$, then $\widehat{W}_i(t) > t$, $\forall t$ in* *interval* (t_1, t_2).

We can now prove that $\mathsf{Approx}(\tau, i, \epsilon)$ will return "τ is feasible" *if and only if* for each task τ_i of τ and for all jobs ℓ of τ_i, there exists a time t between the release of job ℓ and its deadline where $\widehat{W}_{i,\ell}(t) \leq t$. The next lemma formally proves this statement.

Lemma 9 $\mathsf{Approx}(\tau, \epsilon)$ *returns "τ is feasible"* if and only if

$$\begin{aligned}&\forall \tau_i \in \tau,\ \ell \in \mathbb{N}(\ell > 0) :: \\ &\quad (\exists t \in ((\ell-1)p_i, (\ell-1)p_i + d_i] : \widehat{W}_{i,\ell}(t) \leq t).\end{aligned} \tag{13}$$

Proof Sketch: Proving the "only if" direction first, assume that $\mathsf{Approx}(\tau, \epsilon)$ returns "τ is feasible." We can show that for all τ_i, Lemma 7 implies that for all jobs ℓ with deadline prior to $\max_{j \in \{1,\ldots,i\}}\{(k-1)p_j\}$ there exists a $t \in ((\ell-1)p_i, (\ell-1)p_i + d_i]$ such that $\widehat{W}_{i,\ell}(t) \leq t$. Lemma 6 implies that if $\mathsf{ApproxSecondStage}$ returns "τ_i *is schedulable*", then all jobs ℓ with deadlines after $\max_{j \in \{1,\ldots,i\}}\{(k-1)p_j\}$ there exists $t \in ((\ell-1)p_i, (\ell-1)p_i + d_i]$ such that $\widehat{W}_{i,\ell}(t) \leq t$. Together these existential statements imply Equation 13.

For the "if" direction, we can prove the contrapositive. In other words, we will assume that $\mathsf{Approx}(\tau, \epsilon)$ returns "τ is infeasible." It must be the case that some task τ_i of task system τ has been declared "not schedulable" by either $\mathsf{ApproxFirstStage}$ or $\mathsf{ApproxSecondStage}$. There are two cases:

1. $\mathsf{ApproxFirstStage}$ returns "τ_i *is not schedulable*.
2. $\mathsf{ApproxSecondStage}$ returns "τ_i *is not schedulable*.

In either case, we can find a job a of τ_i, such that for all $t \in ((a-1)p_i, (a-1)p_i + d_i]$, $\widehat{W}_{i,a}(t) > t$. Therefore, we have shown the negation of Equation 13. ■

The following theorem proves formally that if Approx declares "τ is feasible", then τ is, in fact, feasible.

Theorem 5 *A sporadic or synchronous periodic task system, τ, is feasible if* $\mathsf{Approx}(\tau, \epsilon)$ *returns "τ* is feasible*" (where $0 < \epsilon < 1$).*

Proof: If $\mathsf{Approx}(\tau, \epsilon)$ returns "τ *is feasible*," then by Lemma 9,

$$\begin{aligned}&\forall \tau_i \in \tau,\ \forall \ell (\in \mathbb{N}) > 0, \\ &\quad \exists t \in ((\ell-1)p_i, (\ell-1)p_i + d_i] : \widehat{W}_{i,\ell}(t) \leq t.\end{aligned}$$

By Lemma 1,

$$\begin{aligned}&\forall \tau_i \in \tau,\ \forall \ell (\in \mathbb{N}) > 0, \\ &\quad \exists t \in ((\ell-1)p_i, (\ell-1)p_i + d_i] : W_{i,\ell}(t) \leq t.\end{aligned}$$

The theorem follows by applying Theorem 4. ■

In the next theorem, we state the implications of $\mathsf{Approx}(\tau, \epsilon)$ returning "τ *is infeasible*":

Theorem 6 *If for a sporadic or synchronous periodic task system, τ, and $\epsilon \in (0, 1)$,* $\mathsf{Approx}(\tau, \epsilon)$ *returns "τ* is infeasible,*" then τ is infeasible on a processor of capacity* $(1 - \epsilon)$.

Proof: The proof is by contradiction. Assume that $\mathsf{Approx}(\tau, \epsilon)$ returns "τ *is infeasible*," and τ is feasible on a processor of capacity $(1 - \epsilon)$. By Lemma 9,

$$\begin{aligned}&\exists \tau_i \in \tau,\ \ell \in \mathbb{N} :: \\ &\quad \forall t \in ((\ell-1)p_i, (\ell-1)p_i + d_i], \widehat{W}_{i,\ell}(t) > t.\end{aligned}$$

This implies from Lemma 2 that

$$\begin{aligned}&\exists \tau_i \in \tau,\ \ell \in \mathbb{N} :: \\ &\quad \forall t \in ((\ell-1)p_i, (\ell-1)p_i + d_i], \left(W_{i,\ell}(t) > \tfrac{k}{k+1} t \geq (1-\epsilon)t\right).\end{aligned}$$

However, if τ_i is feasible on a processor of $(1 - \epsilon)$ capacity, Theorem 4 implies $\forall \tau_i \in \tau,\ \ell(> 0) \in \mathbb{N}, \exists t \in ((\ell-1)p_i, (\ell-1)p_i + d_i]$ such that $W_{i,\ell}(t) \leq (1-\epsilon)t$. This is a contradiction; thus, the theorem is true. ■

Thus, by Theorems 5 and 6, Approx is correct.

5.1 Computational Complexity

The computational complexity of $\mathsf{Approx}(\tau, \epsilon)$ depends entirely on the size of the testing set, $\widetilde{S}_i$. It is easy to see that the size of $\widetilde{S}_i$ is at most:

$$1 + i(k-1) \tag{14}$$

This corresponds to the number of iterations that $\mathsf{ApproxFirstStage}$ must make for each task, τ_i. In our implementation of approximate feasibility-analysis, the condition in Step 2 of $\mathsf{ApproxFirstStage}$ would be executed at most $\Sigma_{i=1}^{n}(1 + (i)(k-1))$ times, which is $O(n^2 k)$.

6 Fully Polynomial-Time Approximation Scheme

For a given accuracy, ϵ, the running time of the approximation algorithm is $O(n^2/\epsilon)$. Thus, these algorithms are members of a family of algorithms that collectively represent a fully polynomial-time approximation scheme for uniprocessor feasibility analysis, with respect to a given priority assignment, for both synchronous period and sporadic tasks systems in a static-priority system. The following theorem states this formally.

Theorem 7 *For any* ϵ *in the range* $(0, 1)$*, there is an algorithm* A_ϵ *that has run-time* $O(n^2/\epsilon)$ *and exhibits the following behavior: On any synchronous periodic or sporadic task system* τ*,*

- if τ *is infeasible on a unit-capacity processor* then *Algorithm* A_ϵ *correctly identifies it as being infeasible;*
- if τ *is feasible on a processor of computing capacity* $(1 - \epsilon)$ then *Algorithm* A_ϵ *correctly identifies it as being feasible;*
- else *Algorithm* A_ϵ *may identify* τ *as being either feasible or infeasible.*

■

7 Summary

It has been shown [1] that there exists a fully polynomial-time approximation scheme (FPTAS) for uniprocessor feasibility analysis of sporadic task sets in dynamic-priority systems. We have constructed a similar FPTAS for static-priority feasibility analysis of uniprocessor synchronous periodic and sporadic task systems with arbitrary relative deadlines. We have, thus, shown that dynamic- and static-priority systems have equivalent approximate feasibility-analysis "tools" available.

The fully polynomial-time approximation tests presented in this paper offer a reduction in complexity for feasibility tests. These approximate feasibility tests may be useful for quick estimates of task system feasibility in automatic system-synthesis tools.

References

[1] Albers, K., and Slomka, F. An event stream driven approximation for the analysis of real-time systems. In *Proceedings of the EuroMicro Conference on Real-Time Systems* (Catania, Sicily, July 2004), IEEE Computer Society Press, pp. 187–195.

[2] Audsley, N., Burns, A., Richardson, M., Tindell, K., and Wellings, A. Applying new scheduling theory to static priority preemptive scheduling. *Software Engineering Journal 8*, 5 (1993), 285–292.

[3] Audsley, N. C., Burns, A., Richardson, M. F., and Wellings, A. J. Hard Real-Time Scheduling: The Deadline Monotonic Approach. In *Proceedings 8th IEEE Workshop on Real-Time Operating Systems and Software* (Atalanta, May 1991).

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

[5] Lehoczky, J., Sha, L., and Ding, Y. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings of the Real-Time Systems Symposium - 1989* (Santa Monica, California, USA, Dec. 1989), IEEE Computer Society Press, pp. 166–171.

[6] Lehoczky, J. P. Fixed priority scheduling of periodic tasks with arbitrary deadlines. In *IEEE Real-Time Systems Symposium* (Dec. 1990), pp. 201–209.

[7] Leung, J., and Whitehead, J. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation 2* (1982), 237–250.

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

[9] Shih, W. K., Liu, J. W. S., and Liu, C. L. Modified rate-monotonic algorithm for scheduling periodic jobs with deferred deadlines. *IEEE Transactions on Software Engineering 19*, 12 (December 1993), 1171–1179.