Approximation Algorithms for Feasibility Analysis in Real-Time Static-Priority Systems *

Nathan Fisher‡ and Sanjoy Baruah

Department of Computer Science, The University of North Carolina, Chapel Hill, NC 27599 USA

Abstract. Feasibility tests determine whether it is possible for a given real-time system to always meet all of its timing constraints on a specified processing platform. Current feasibility tests for the uniprocessor static-priority scheduling of sporadic task systems run in pseudo-polynomial time. We present a fully polynomial-time approximation scheme (FPTAS) for feasibility analysis in static-priority systems. This test is an approximation in the sense that that there is a quantifiable trade-off between the fraction of the processor's capacity that must be left unused, and the running time of the feasibility test.

Keywords: Real-time scheduling; Uniprocessor systems; Static-priority systems; Feasibility analysis.

1. Introduction

Over the years, the *sporadic task model* (Mok, 1983; Leung and Whitehead, 1982) has proven remarkably useful for the modelling of recurring processes that occur in hard-real-time systems. In this model, a *sporadic task* $\tau_i = (e_i, d_i, p_i)$ is characterized by a *worst-case execution requirement* e_i, a *(relative) deadline* d_i, and a *minimum inter-arrival*

‡ Correspondence to: Nathan Fisher, Department of Computer Science, The University of North Carolina, Chapel Hill, NC 27599 USA. E-mail: fishern@cs.unc.edu

* Supported in part by the National Science Foundation (Grant Nos. ITR-0082866, CCR-0204312, and CCR-0309825). Preliminary versions of some of these results appeared as Fisher, N., and Baruah, S. 2005. 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 5-7). and Fisher, N., and Baruah, S. 2005. A fully polynomial-time approximation scheme for feasibility analysis in static-priority systems with arbitrary relative deadlines. In *Proceedings of Euromicro Conference on Real-Time Systems* (Palma de Mallorca, Spain, June 6-8).

separation p_i, which is, for historical reasons, also referred to as the *period* of the task. Such a sporadic task generates a potentially infinite sequence of jobs, with successive job-arrivals separated by at least p_i time units. 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 *sporadic task system* is comprised of several such sporadic tasks.

In this paper, we are concerned with *preemptive* scheduling on *uniprocessor* platforms. That is, we assume that there is a single shared processor upon which all the jobs in the system must execute, and that a job executing on this processor may be interrupted at any instant in time, and its execution resumed later, at no cost or penalty. Without loss of generality, we also assume that the processor has an execution rate of unity (equivalently, that all task parameters are normalized to the processor speed).

Many run-time scheduling algorithms operate as follows: at each instant in time, they assign a priority to each job that is awaiting execution, and choose for execution the highest-priority waiting job. In *static-priority* scheduling algorithms for sporadic task systems, each task is assigned a distinct priority, and all jobs of a task execute at the task's priority. In *dynamic-priority* scheduling algorithms, by contrast, different jobs generated by the same task may have different priorities (and indeed, the priority of an individual job may change during run-time).

A sporadic task system is said to be *dynamic-priority feasible* (respectively, *static-priority feasible*) upon a specified processing platform if and only if it can be scheduled on the platform by a dynamic-priority (resp., static-priority) scheduling algorithm such that all jobs that may be generated by this task system always meet their deadlines. For sporadic task systems comprised of tasks that all have their relative deadline parameter equal to their periods, dynamic-priority feasibility analysis can be performed in time linear in the representation of the task system (Liu and Layland, 1973); for arbitrary sporadic task systems, however, all dynamic-priority feasibility analysis tests known take time exponential in the representation of the task system.

A static-priority feasibility test for systems of sporadic tasks, each with its relative deadline parameter equal to its period, was presented in (Lehoczky *et al.*, 1989). In (Audsley *et al.*, 1993), a feasibility test for sets of tasks in which the relative deadline of each task is less than or equal to its period is presented. In both of these static-priority feasibility tests, the running time of the test is polynomial in the values of the parameters of the tasks in the task system, i.e., these are pseudo-polynomial time tests. (Lehoczky, 1990) provides a more general feasibility test for sporadic task systems where the relation between deadlines and periods may be arbitrary; the run-time of this test is not known to be better than exponential. (These known results concerning the computational complexity of uniprocessor feasibility-analysis are summarized in Table III.)

In (Albers and Slomka, 2004), a *fully polynomial-time approximation scheme (FPTAS)* was presented for dynamic-priority feasibility analysis of sporadic task systems. This FPTAS 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.

This research. In this paper, we extend the results of Albers and Slomka to the domain of static-priority scheduling. 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.

Significance of this research. The presence or otherwise of an FPTAS for static-priority feasibility analysis 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. The FPTAS presented in this paper answers this question in the affirmative.

Since the running time of current exact feasibility tests for static-priority task systems depends on the ratio between the largest and smallest period, the computational 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.

Organization. The remainder of this paper is organized as follows. We provide some background on real-time uniprocessor scheduling and formally define our task model in Section 2. 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 (Lehoczky *et al.*, 1989; Audsley *et al.*, 1993; Lehoczky, 1990), 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 give an FPTAS for task systems where the relative deadline of each task is at most period in Section 3.2 and prove the FPTAS correct (Section 3.4). 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 4.2. Finally, we give a brief review of approximation schemes

and formally state the main result of this paper, the existence of an FPTAS for feasibility in static-priority systems, in Section 5.

2. Model and Previous Work

2.1. Task and Processor Model

As stated in the introduction, we restrict our attention in this paper to *uniprocessor* platforms, and to *preemptive* scheduling – a job executing on the processor may be interrupted at any instant in time, and its execution resumed later, at no cost or penalty. We consider the *sporadic* task model. 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. 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.

In some of the initial work on real-time scheduling (see, e.g., (Liu and Layland, 1973)), it is assumed that all tasks have their relative deadlines equal to their period parameters (i.e. $d_i = p_i$ for $1 \leq i \leq n$). Such systems are sometimes referred to in the real-time scheduling literature as **implicit-deadline** systems. Later work (e.g., (Leung and Whitehead, 1982)) relaxed the equality constraint between relative deadlines and periods, and considered tasks with relative deadlines bounded by periods (i.e. $d_i \leq p_i$ for $1 \leq i \leq n$). Such task systems with bounded relative deadlines are sometimes referred to as **constrained** task systems. The most general model, the **arbitrary** task system, imposes no constraints between relative deadlines and periods. Table I summarizes the task models.

2.2. Real-Time Uniprocessor Scheduling Algorithms

A job is said to be *active* at a specified time-instant in a schedule, if it has not yet completed execution and its deadline has not yet elapsed. As

Table I. Sporadic Task Models

Task Model	**Relative Deadline Constraint**
Implicit-deadline	Each task's relative deadline is equal to its period parameter (i.e. $d_i = p_i$).
Constrained	Each task's relative deadline is at most its period parameter (i.e. $d_i \leq p_i$).
Arbitrary	No constraint is placed on the relationship between relative deadlines and periods

mentioned in the introduction, most scheduling algorithms used in real-time systems assign each active job a *priority* at each instant in time, and select for execution the active job with the highest priority. (such scheduling algorithms are sometimes called *priority-driven* scheduling algorithms). In *dynamic-priority* scheduling algorithms, different jobs of the same task may be assigned different priorities, and the priority of a job may change during run-time. In *static-priority* algorithms, each task is assigned a distinct priority, and all jobs of a task execute at the task's priority.

The **earliest deadline first** scheduling algorithm (EDF) is an example of a dynamic-priority scheduling algorithm. EDF assigns the highest priority (and hence, the processor) at each time instant to the job awaiting execution that has the earliest deadline. Two well-studied static-priority algorithms for scheduling sporadic task systems are the **rate monotonic** algorithm (RM), and the **deadline monotonic** algorithm (DM). RM (Liu and Layland, 1973) assigns each task a priority equal to the inverse of its period, while DM (Leung and Whitehead, 1982) assigns each task a priority equal to the inverse of its relative deadline (in either case ties may be broken arbitrarily). These algorithms are summarized in Table II.

For a given task system and processor, **static-priority** (respectively, **dynamic-priority**) **feasibility-analysis** is the process of determining whether the task system can be scheduled on the processor by a static-priority (resp., dynamic-priority) scheduling algorithm such that all

Table II. Uniprocessor scheduling algorithms

Algorithm		**Priority assignment rule**
Earliest-deadline first	(EDF)	Job with earlier deadline has higher priority
Rate-monotonic	(RM)	Task with smaller value of period parameter has higher priority
Deadline-monotonic	(DM)	Task with smaller value of relative-deadline parameter has higher priority

deadlines are met, for all possible sequences of job arrivals that may be generated by the task system.

A scheduling algorithm A is said to be dynamic-priority (resp., static-priority) **optimal** for a particular class of task systems if algorithm A will always meet all deadlines for all dynamic-priority feasible (respectively, static-priority feasible) task systems that fall within this particular class of task systems. The following results are known concerning optimality of scheduling algorithms (this information, too, is summarized in Table III):

- EDF is an optimal dynamic-priority algorithm for sporadic task systems (Liu and Layland, 1973; Dertouzos, 1974).

- RM is an optimal static-priority algorithm for implicit-deadline sporadic task systems, i.e., for task systems in which each task has its relative deadline parameter equal to its period parameter (Liu and Layland, 1973).

- DM is an optimal static-priority algorithm for constrained sporadic task systems, i.e., for for task systems in which each task has its relative deadline parameter no larger than its period parameter (Leung and Whitehead, 1982).

- To the best of our knowledge, there is currently no known optimal static-priority scheduling algorithm for arbitrary task systems.

Recently, the relative merits of dynamic- and static-priority scheduling have been widely debated in the real-time systems community —

Table III. Known results concerning **(i)** the computational complexity of, and **(ii)** optimal schedulers for, uniprocessor feasibility analysis for sporadic task systems.

Task Model:	**Cxty. of Feas. analysis**		**Opt. scheduler**	
	static-pri	dynamic-pri	static-pri	dynamic-pri
implicit-deadline	pseudo-poly	linear	RM	EDF
constrained	pseudo-poly	exponential	DM	EDF
arbitrary	exponential	exponential	?	EDF

see, e.g., (Buttazzo, 2005)[1]. Our research in this paper can be seen as part of this wider debate, in the following sense. The results in (Albers and Slomka, 2004) show that it is possible to determine in polynomial time, with bounded error, the dynamic-priority feasibility of an arbitrary sporadic task system. In the context of the debate on dynamic- and static-priority systems, it is interesting to consider whether a similar approximation exists for static-priority systems, or whether approximate feasibility determination is one area where dynamic-priority scheduling is superior to static-priority scheduling. In the remainder of the paper, we show that an FPTAS for feasibility analysis exists for the static-priority scheduling of sporadic tasks on a preemptive uniprocessor. The results of our work imply that equivalent approximations exist for dynamic- and static-priority systems when considering an inexact notion of infeasibility (i.e. the algorithm may return "infeasible" only if it is guaranteed to be infeasible on a sufficiently smaller capacity processor).

[1] Indeed, a panel discussion (Mossé *et al.*, 2004) was recently organized during a prominent international symposium devoted to real-time systems, with the objective of addressing the shortcoming that "there is not a general agreement in the real-time community about the suitability of these two scheduling approaches for future real-time systems," and hoping to start "a constructive [...] discussion aimed at clarifying some critical points and drawing some missing research lines."

3. Constrained Task Systems

In this section, we derive an approximate feasibility test for constrained-deadline static-priority systems; i.e., sporadic task systems where each task's relative deadline is no larger than 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 (Lehoczky *et al.*, 1989; Audsley *et al.*, 1993; Lehoczky, 1990)). 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. 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

For a sporadic task τ_i, the maximum amount of execution time that τ_i can request over an interval $(0, t]$ occurs when τ_i releases a job at time zero, and successive jobs every p_i time units. The *synchronous arrival sequence* occurs when all tasks of a sporadic task system release jobs at the same time instant and subsequent jobs as soon as permissible. The total execution time requested by a task τ_i in a synchronous arrival sequence 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 $(0, t]$:

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

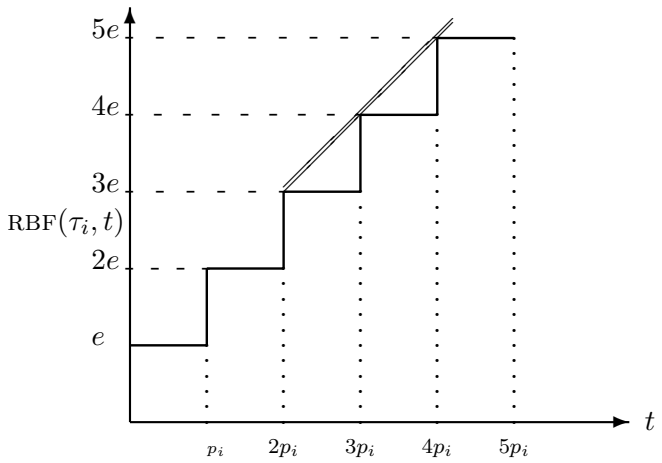

Figure 1. The step function denotes a plot of $\text{RBF}(\tau_i, t)$ as a function of t. The double line represents the function $\delta(\tau_i, t)$, approximating $\text{RBF}(\tau_i, t)$, where $k = 3$; for $t \leq 2p_i$, $\delta(\tau_i, t) \equiv \text{RBF}(\tau_i, t)$.

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.

3.1.1. *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:

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

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

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

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

3.2.1. *Exact Test*

For constrained static-priority task systems, (Liu and Layland, 1973) 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. Note that this sequence of job arrivals is the synchronous arrival sequence. In a constrained, sporadic task system, it is necessary and sufficient to only check the response time of the first job of each task in a synchronous arrival sequence. If the response time of the first job of task τ_i is at most its relative deadline, then all jobs generated by τ_i are guaranteed to always meet their deadlines; else, they cannot be so guaranteed. 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 $\mathbf{T}_{H_i}$ be the set of tasks with priority greater than τ_i. Then, the cumulative request-bound function is defined as:

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

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.*, 1991) presented an exact static-priority feasibility test for sporadic task systems: all jobs of task τ_i are guaranteed to always meet their deadlines 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 (observe that this theorem uses the fact that DM is a static-priority optimal scheduling algorithm for constrained sporadic task systems):

Theorem 1 (from (Audsley *et al.*, 1991**))** *In a sporadic task system, task τ_i always meets all deadlines when scheduled by* DM *if and only if $\exists t \in (0, d_i]$ such that $W_{i,1}(t) \leq t$.*

3.2.2. *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:

$$\widehat{W}_{i,1}(t) \stackrel{\text{def}}{=} e_i + \sum_{\tau_j \in \mathbf{T}_{H_i}} \delta(\tau_j, t) \tag{5}$$

In Section 3.4, we will see that the value of $\widehat{W}_{i,1}(t)$ is always at least that of $W_i(t)$. If we use the approximate cumulative request-bound function, $\widehat{W}_{i,1}(t)$, to find a fixed point as in Theorem 1, it is possible for the approximate function's smallest fixed point to exceed the exact function's smallest fixed point; therefore, the test is no longer necessary and sufficient. Instead, we will have sufficient test for static-priority feasibility, as the following theorem states (proved in Section 3.4):

Theorem 2 *In a sporadic task system, task τ_i always meets all deadlines when scheduled by* DM *if* $\exists t \in (0, d_i]$ *such that* $\widehat{W}_{i,1}(t) \leq t$.

Using the approximate cumulative request-bound function also no longer gives an exact check for infeasibility. Instead, if we cannot find a $t \in (0, d_i]$ such that $\widehat{W}_{i,1}(t) \leq t$, then τ_i may miss a deadline on a lower capacity processor. In fact, we can quantify the amount by which the smaller capacity processor must be slower; the following theorem (also proved in Section 3.4) derives this capacity:

Theorem 3 *If* $\forall t \in (0, d_i]$, $\widehat{W}_{i,1}(t) > t$, *then there exist job-arrival sequences for τ in which τ_i misses some deadlines when scheduled by* 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 static-priority feasibility of constrained sporadic task system τ. The test is:

> If for all tasks, $\tau_i \in \tau$, there is a time $t \leq d_i$ such that $\widehat{W}_{i,1}(t) \leq t$, then τ *is static-priority feasible.* Otherwise, τ is guaranteed to be infeasible on a processor of $(1 - \epsilon)$ capacity.

Section 3.5 will show that the complexity of the approximate feasibility test for constrained task systems is polynomial-time in terms of the number of tasks and the accuracy parameter, ϵ.

3.3. Example

We now consider an example system τ, to illustrate both the exact and approximate feasibility tests. Let τ be comprised of the following tasks:

$$\begin{aligned} \tau_1 &= (1, 3, 3) \\ \tau_2 &= (2, 5, 5) \\ \tau_3 &= (2, 12, 12) \end{aligned}$$

Let us first demonstrate the exact feasibility test. We must calculate the smallest fixed points of the the cumulative request-bound function

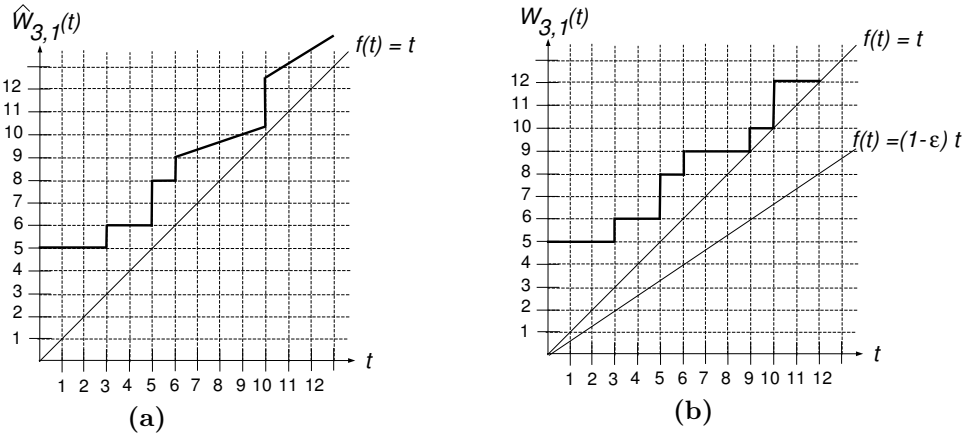

Figure 2. The two graphs above illustrate the *cumulative request-bound functions* of task τ_3 for task system τ where $\tau_1 = (1, 3, 3)$, $\tau_2 = (2, 5, 5)$, and $\tau_3 = (2, 12, 12)$. Graph **(a)** shows the approximate cumulative request-bound function, $\widehat{W}_{3,1}(t)$ where $\epsilon = \frac{1}{3}$ (i.e. $k = 2$) . Since there does not exist a $t \in (0, 12)$ such that $\widehat{W}_{3,1}(t) \leq t$, the approximate feasibility test declares τ *is infeasible*. Graph **(b)** shows the exact cumulative request-bound function, $W_{3,1}(t)$. Notice, that τ is, in fact, infeasible on a uniprocessor of $(1 - \frac{1}{3})$ capacity; however, τ is feasible on the unit capacity uniprocessor.

(if they exist). The fixed points are: $W_{1,1}(1) = 1$, $W_{2,1}(3) = 3$, and $W_{3,1}(9) = 9$. Since, for all τ_i, there exists a t such that $W_{i,1}(t) \leq t$ and $t \in (0, d_i]$, then τ is feasible on a uniprocessor. Figure 2(b) shows the cumulative request-bound function for τ_3. Observe from the graph that you can quickly identify the smallest fixed point of $W_{3,1}(t)$.

For the approximate test, $\widehat{W}_{1,1}(1) = 1$, and $\widehat{W}_{2,1}(3) = 3$. However, it is easy to see from Figure 2(a) that there does not exist a $t \in (0, 12]$ such that $\widehat{W}_{3,1}(t) \leq t$. Therefore, by Theorem 3, τ is infeasible on a uniprocessor of $(1 - \epsilon)$ capacity. The fact that $W_{3,1}(t)$ does not fall below the line $f(t) = (1 - \epsilon)t$ in Figure 2(b) illustrates the infeasibility of τ on this smaller capacity processor.

3.4. Correctness of Approximate Feasibility Test for Constrained Task Systems

In this section, we prove Theorems 2 and 3, thereby proving the correctness of the approximate feasibility test. However, before proving

the theorems, it will be useful to prove properties of the request-bound functions. Using the request-bound function properties, we will be able to prove lemmas about the approximate cumulative request-bound function. From these lemmas and properties, Theorems 2 and 3 will follow.

We prove several properties about $\delta(\tau_i, t)$, which together show that $\delta(\tau_i, t)$ closely tracks $\text{RBF}(\tau_i, t)$. The first property states that our approximation always exceeds or equals RBF.

Property 1 $\forall t \geq 0$, $\delta(\tau_i, t) \geq \text{RBF}(\tau_i, t)$.

Proof: For all $t \in (0, (k-1)p_i]$, $\delta(\tau_i, t) = \text{RBF}(\tau_i, t)$, by definition. For $t > (k-1)p_i$, $\delta(\tau_i, t) = e_i + \frac{te_i}{p_i} = (\frac{t}{p_i} + 1)e_i \geq \lceil \frac{t}{p_i} \rceil e_i = \text{RBF}(\tau_i, t)$ □

The second property states that if the approximation strictly exceeds the RBF at time t, then we have calculated at least $k-1$ steps of the RBF. We may observe this visually in Figure 1, because the linear approximation does not begin until after the $k-1$ step. Formally,

Property 2 *If* $\delta(\tau_i, t) > \text{RBF}(\tau_i, t)$, *then* $\text{RBF}(\tau_i, t) \geq ke_i$.

Proof: $\delta(\tau_i, t) > \text{RBF}(\tau_i, t)$ implies that $t > (k-1)p_i$. Thus, $\text{RBF}(\tau_i, t) = \lceil \frac{t}{p_i} \rceil e_i > \left\lceil \frac{(k-1)p_i}{p_i} \right\rceil e_i = (k-1)e_i$. Since RBF increases by e_i at each step, $\text{RBF}(\tau_i, t) \geq ke_i$. □

The third property reflects the fact that the approximation never exceeds the RBF by more than one step size, e_i. Again, visually this is easy to see, since after $k-1$ steps, $\delta(\tau_i, t)$ is a linear interpolation of the "edges" of the steps.

Property 3 $\forall t \geq 0$, $\delta(\tau_i, t) - \text{RBF}(\tau_i, t) \leq e_i$.

Proof: For all $t \in (0, (k-1)p_i]$, $\delta(\tau_i, t) - \text{RBF}(\tau_i, t) = 0 \leq e_i$. For $t > (k-1)p_i$, $\delta(\tau_i, t) - \text{RBF}(\tau_i, t) = \frac{te_i}{p_i} + e_i - \left\lceil \frac{t}{p_i} \right\rceil e_i \leq \frac{te_i}{p_i} + e_i - \frac{te_i}{p_i} = e_i$. □

Property 4, below, bounds the ratio of the value of the approximate function at time t to the value of the exact function:

Property 4 $\forall t \geq 0$, $\text{RBF}(\tau_i, t) \leq \delta(\tau_i, t) \leq (1 + \frac{1}{k})\text{RBF}(\tau_i, t)$.

Proof: If $t \leq (k-1)p_i$, then $\delta(\tau_i, t) = \text{RBF}(\tau_i, t)$. Since $k \geq 1$, for all $0 < \epsilon < 1$, the inequality $(1 + \frac{1}{k}) > 1$ holds. Therefore, $\text{RBF}(\tau_i, t) \leq \delta(\tau_i, t) \leq (1 + \frac{1}{k})\text{RBF}(\tau_i, t)$.

Otherwise, if $t > (k-1)p_i$, then $\text{RBF}(\tau_i, t) \leq \delta(\tau_i, t)$ follows from Property 1. By Property 3, $\delta(\tau_i, t) \leq \text{RBF}(\tau_i, t) + e_i$. Then by Property 2, $\text{RBF}(\tau_i, t) + e_i \leq \text{RBF}(\tau_i, t) + \frac{\text{RBF}(\tau_i, t)}{k}$. This implies, $\forall t \geq 0$, $\delta(\tau_i, t) \leq (1 + \frac{1}{k})\text{RBF}(\tau_i, t)$. □

The following lemmas describe the implications of using the approximation function δ in the cumulative request-bound function. Informally, 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 *If $\widehat{W}_{i,\ell}(t) \leq t$, then $W_{i,\ell}(t) \leq t$.*

Proof:

By Property 1, $\sum_{\tau_j \in \mathbf{T}_{H_i}} \delta(\tau_j, t) \geq \sum_{\tau_j \in \mathbf{T}_{H_i}} \text{RBF}(\tau_j, t)$. Thus, $\ell e_i + \sum_{\tau_j \in \mathbf{T}_{H_i}} \delta(\tau_j, t) \leq t \Rightarrow \ell e_i + \sum_{\tau_j \in \mathbf{T}_{H_i}} \text{RBF}(\tau_j, t) \leq t$. The Lemma follows from the definitions of $W_{i,\ell}(t)$ and $\widehat{W}_{i,\ell}(t)$. □

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 *If $\widehat{W}_{i,\ell}(t) > t$, then $W_{i,\ell}(t) > \frac{k}{1+k}(t)$.*

Proof:

$$
\begin{aligned}
&\widehat{W}_{i,\ell}(t) = \ell e_i + \textstyle\sum_{\tau_j \in \mathbf{T}_{H_i}} \delta(\tau_j, t) > t \\
&\Rightarrow \ell e_i + \textstyle\sum_{\tau_j \in \mathbf{T}_{H_i}} (1 + \frac{1}{k})\text{RBF}(\tau_j, t) > t \qquad \text{(by Property 4)} \\
&\Rightarrow (1 + \tfrac{1}{k})(\ell e_i + \textstyle\sum_{\tau_j \in \mathbf{T}_{H_i}} \text{RBF}(\tau_j, t)) > t \\
&\Rightarrow \ell e_i + \textstyle\sum_{\tau_j \in \mathbf{T}_{H_i}} \text{RBF}(\tau_j, t) > \frac{k}{1+k}(t)
\end{aligned}
$$

The Lemma follows from definition of $W_{i,\ell}(t)$. □

We are now prepared to prove Theorems 2 and 3, originally stated in Section 3.

Proof of Theorem 2 Assume there exists a $t_0 \in (0, d_i]$ such that $\widehat{W}_{i,1}(t_0) \leq t_0$. Then, by Lemma 1, $W_{i,1}(t_0) \leq t_0$. From Theorem 1, τ_i always meets all deadlines using DM. □

Proof of Theorem 3 The proof is by contradiction. Assume that for all $t \in (0, d_i]$, $\widehat{W}_{i,1}(t) > t$, but τ_i is still feasible on a processor of $(1-\epsilon)$ capacity. Notice that $1 - \frac{k}{k+1} \leq \epsilon$. By Theorem 1, $\exists t_0 \in (0, d_i]$ such that $W_{i,1}(t_0) \leq (1-\epsilon)t_0 \leq (\frac{k}{k+1})t_0$. But, $\widehat{W}_{i,1}(t_0) > t_0 \Rightarrow W_{i,1}(t_0) > (\frac{k}{k+1})t_0$ by Lemma 2. This is a contradiction; therefore, τ_i is infeasible on a processor of capacity $(1-\epsilon)$. □

3.5. Testing Set

We now turn our attention to the most significant benefit of the approximate feasibility test: its polynomial time complexity. In order to prove that the test runs in polynomial time, we show that the number of points at which Equation (5) must be evaluated is bounded by a polynomial in terms of n and ϵ.

As a basis of comparison, let us first consider the testing set for the exact feasibility test. It is known (Audsley *et al.*, 1993) that the time-instants at which the condition of Theorem 1 must be evaluated is:

$$S_i \stackrel{\text{def}}{=} \left\{ t = bp_a : a = 1, \ldots, i; b = 1, \ldots, \left\lfloor \frac{d_i}{p_a} \right\rfloor \right\} \tag{6}$$

Informally, these are the only values of t where $W_j(t)$ changes value, for some j, $1 \leq j \leq i$. The number of points in this set may be as large as:

$$\sum_{j=1}^{i} \left\lfloor \frac{d_i}{p_j} \right\rfloor \tag{7}$$

Since the number of points that must be checked is dependent on the task periods, this represents a pseudo-polynomial feasibility test.

In the remainder of this section, we will show that if we were to use the approximation $\widehat{W}_{i,1}(t)$ in place of $W_i(t)$, we would only need test the condition of Theorem 2 at *polynomially* many points. In particular,

we will show that the set of points that must be checked is:

$$\widehat{S}_i \stackrel{\text{def}}{=} \{t = bp_a : a = 1, \ldots, i-1; b = 1, \ldots, k-1\} \cup \{t = d_i\} \qquad (8)$$

3.5.1. *Necessity and Sufficiency of Testing Set*
In the following argument, we will show that $\widehat{S}_i$ contains a sufficient number of points to determine the existence (or absence) of a time t such that $\widehat{W}_{i,1}(t) \leq t$. Our proof obligation is to show: if for all $t \in \widehat{S}_i$, $\widehat{W}_{i,1}(t) > t$, then for all $t \in (0, d_i]$, $\widehat{W}_{i,1}(t) > t$.

We call two elements t_1 and t_2 ($t_1 < t_2$) in set $\widehat{S}_i$ *adjacent*, if no t satisfying $t_1 < t < t_2$ is in $\widehat{S}_i$. In order to fulfill our proof obligation, we will show that for any pair of adjacent points in $\widehat{S}_i$, if the value of $\widehat{W}_{i,1}$ at the adjacent points lies above $f(t) = t$, then the value of $\widehat{W}_{i,1}$ at all points in between the adjacent points are above $f(t) = t$.

Define a linear function, $g_{(t_1,t_2),\ell}(t)$ over the interval (t_1, t_2) as follows,

$$g_{(t_1,t_2),\ell}(t) \stackrel{\text{def}}{=} \frac{\widehat{W}_{i,\ell}(t_2) - \widehat{W}_{i,\ell}(t_1)}{t_2 - t_1}(t - t_1) + \widehat{W}_{i,\ell}(t_1), \ \forall t \in (t_1, t_2) \qquad (9)$$

Intuitively, $g_{(t_1,t_2),\ell}(t)$ is a linear interpolation of points $(t_1, \widehat{W}_{i,\ell}(t_1))$ and $(t_2, \widehat{W}_{i,\ell}(t_2))$. For example, in Figure 2a consider the two adjacent points $t_1 = 6$ and $t_2 = 10$. $g_{(t_1,t_2),1}(t)$ is the line defined by points $(6, 8)$ and $(10, 10\frac{1}{3})$.

The following claim shows that between any two adjacent points, the linear function, $g_{(t_1,t_2),\ell}(t)$ is less than the approximation function. So, $g_{(t_1,t_2),\ell}(t)$ bounds the approximation function from below over the interval (t_1, t_2). Let $r_i(t)$ be the total execution of all tasks τ_j, of priority τ_i or greater, that release jobs at time t when $t \leq (k-1)p_j$. Informally, $r_i(t)$ is the "step"-size of $\widehat{W}_{i,\ell}(t)$. Formally,

$$r_i(t) \stackrel{\text{def}}{=} \sum_{\{\tau_j \in \mathbf{T}_{H_i} : (p_j \text{divides } t) \wedge (t \leq (k-1)p_j)\}} e_j. \qquad (10)$$

Claim 1 *For adjacent elements,* $t_1,\ t_2 \in \widehat{S}_i\ (t_1 < t_2)$, $\Big(g_{(t_1,t_2),\ell}(t) < \widehat{W}_{i,\ell}(t),\ \forall t \in (t_1,t_2)\Big)$.

Proof: We know that $g_{(t_1,t_2),\ell}(t_1) = \widehat{W}_{i,\ell}(t_1)$ and $g_{(t_1,t_2),\ell}(t_2) = \widehat{W}_{i,\ell}(t_2)$, by definition of $g_{(t_1,t_2),\ell}(t)$. Observe that the "height" of the step for $\widehat{W}_{i,\ell}(t)$ is $r_i(t) > 0$ at every point t in $\widehat{S}_i$. Also, note that $\widehat{W}_{i,\ell}(t)$ is linear in interval (t_1,t_2). Then, for $t \in (t_1,t_2)$, $\widehat{W}_{i,\ell}(t)$ is a line that passes through $(t_1, \widehat{W}_{i,\ell}(t_1) + r_i(t_1))$ and $(t_2, \widehat{W}_{i,\ell}(t_2))$. Thus for t in interval (t_1,t_2), $\widehat{W}_{i,\ell}(t) = \frac{\widehat{W}_{i,\ell}(t_2) - \widehat{W}_{i,\ell}(t_1) - r_i(t_1)}{t_2 - t_1}(t - t_1) + \widehat{W}_{i,\ell}(t_1) + r_i(t_1)$. Notice that $\widehat{W}_{i,\ell}(t) - g_{(t_1,t_2),\ell}(t) = r_i(t_1)\left(\frac{t_2 - t}{t_2 - t_1}\right) > 0$. This implies, $\widehat{W}_{i,\ell}(t) > g_{(t_1,t_2),\ell}(t)$. □

The next lemma shows that for any adjacent elements of $\widehat{S}_i$, if the value of $\widehat{W}_{i,\ell}$ lies above the line $f(t) = t$, then the value of $\widehat{W}_{i,\ell}$ of all points in between the adjacent elements must lie above $f(t) = t$.

Lemma 3 *For adjacent elements,* $t_1,\ t_2 \in \widehat{S}_i$, *if* $\widehat{W}_{i,\ell}(t_1) > t_1$ *and* $\widehat{W}_{i,\ell}(t_2) > t_2$, *then* $\widehat{W}_{i,\ell}(t) > t$, $\forall t \in (t_1,t_2)$.

Proof: Notice that $g_{(t_1,t_2),\ell}(t_1) = \widehat{W}_{i,\ell}(t_1) > t_1$ and $g_{(t_1,t_2),\ell}(t_2) = \widehat{W}_{i,\ell}(t_2) > t_2$. Now consider function $g_{(t_1,t_2),\ell}(t)$ over the interval (t_1,t_2). Assume that there exists $t' \in (t_1,t_2)$, such that $g_{(t_1,t_2),\ell}(t') \le t'$. Since, $g_{(t_1,t_2),\ell}(t)$ is linear the slope from t_1 to t' must equal the slope from t' to t_2. But,

$$\begin{aligned} & \widehat{W}_{i,\ell}(t') \le t' \\ \Rightarrow\ & \tfrac{\widehat{W}_{i,\ell}(t') - t_1}{t' - t_1} \le 1 \\ \Rightarrow\ & \tfrac{\widehat{W}_{i,\ell}(t') - \widehat{W}_{i,\ell}(t_1)}{t' - t_1} < 1, \end{aligned}$$

and

$$\begin{aligned} & -\widehat{W}_{i,\ell}(t') \ge -t' \\ \Rightarrow\ & \tfrac{t_2 - \widehat{W}_{i,\ell}(t')}{t_2 - t'} \ge 1 \\ \Rightarrow\ & \tfrac{\widehat{W}_{i,\ell}(t_2) - \widehat{W}_{i,\ell}(t')}{t_2 - t'} > 1. \end{aligned}$$

The slopes are unequal and we have reached a contradiction. Our assumption that there exists a t' such that $g_{(t_1,t_2),\ell}(t') \le t'$ is incorrect. Therefore, $\forall t \in (t_1,t_2)$, $g_{(t_1,t_2),\ell}(t) > t$. By Claim 1, $\widehat{W}_{i,\ell}(t) > t$, $\forall t \in (t_1,t_2)$. □

The following theorem proves $\widehat{S}_i$ is a necessary and sufficient testing set for our approximation.

Theorem 4 *There exists $t \in (0, d_i]$ such that $\widehat{W}_{i,1}(t) \leq t$ if and only if there exists $t' \in \widehat{S}_i$ such that $\widehat{W}_{i,1}(t') \leq t'$.*

Proof: The "if" direction is obvious since $\widehat{S}_i \subset (0, d_i]$. Therefore, we will concentrate on the "only if" direction of the proof. If there exists a $t \in (0, p_{min})$ such that $\widehat{W}_{i,1}(t) \leq t$ (where p_{min} is the smallest period), then $\widehat{W}_{i,1}(p_{min}) \leq p_{min}$ and $p_{min} \in \widehat{S}_i$. So, assume there exists a $t \in [p_{min}, d_i]$ such that $\widehat{W}_{i,1}(t) \leq t$, but $\nexists t' \in \widehat{S}_i$ such that $\widehat{W}_{i,1}(t') \leq t'$. Since $\widehat{S}_i \subset [p_{min}, d_i]$, there exists adjacent elements t_1 and t_2 of $\widehat{S}_i$ such that $t \in (t_1, t_2)$. But, by Lemma 3, for all $t \in (t_1, t_2)$, $\widehat{W}_{i,1}(t) > t$. This a contradiction; therefore, our assumption that there did not exist a $t' \in \widehat{S}_i$ such that $\widehat{W}_{i,1}(t') \leq t'$ is incorrect. □

3.5.2. *Computational Complexity*

It is easy to see that the number of items in the testing set for τ_i is at most:

$$1 + (i-1)(k-1) \tag{11}$$

In a naive implementation of approximate feasibility-analysis, the condition in Theorem 2 would be evaluated at most $\Sigma_{i=1}^{n}(1+(i-1)(k-1))$ times, which is $\mathrm{O}(n^2 k)$. A smarter implementation is to consider only the points of $\widehat{S}_n$. It is possible to use a "heap-of-heaps" data structure (see, e.g, (Mok, 1988)) to determine the order of the job-releases in the test set, $\widehat{S}_n$, in $\mathrm{O}(\log n)$ time. For each distinct time $t \in \widehat{S}_n$, it is also possible to determine the set of tasks that have their execution requests satisfied by time t in $\mathrm{O}(\log n)$ time. Therefore, we can reduce the overall time complexity to $\mathrm{O}(nk \log n)$.

4. Arbitrary Task Systems

When deadlines can exceed periods, (Lehoczky, 1990) 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 $\mathbf{T}_i = \mathbf{T}_{H_i} \cup \{\tau_i\}$ 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 an arbitrary task system 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 possibly *exponential* time test. The reason that the test is not polynomial-time is the following:

- The length of the level-i busy interval does not depend on n, but on the p_i and e_i terms. The exact length of the level-i busy interval is the solution to the following equation:

$$t = \sum_{j=1}^{i} \left\lceil \frac{t}{p_i} \right\rceil e_i \tag{12}$$

  Therefore, the level-i busy interval contains possibly exponential number of jobs of τ_i. Applying the approximate feasibility test of the previous section would require running the test a exponential 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 an exponential (or even pseudo-polynomial) number of checks are not required. We construct an FPTAS for feasibility analysis in static-priority, arbitrary task systems given an arbitrary priority assignment. Furthermore, our FPTAS for arbitrary task systems achieves the same asymptotic time complexity as the FPTAS for constrained task systems. 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. 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 of the correctness of the feasibility approximation in Section 4.2. We derive the running time of approximation in Section 4.3.

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:

$$\widetilde{W}_i(t) \stackrel{\text{def}}{=} \delta(\tau_i, t) + \sum_{\tau_j \in \mathbf{T}_{H_i}} \delta(\tau_j, t) \tag{13}$$

$\widetilde{W}_i(t)$ 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 3.

The next lemma describes the time interval where $\widetilde{W}_i(t)$ is guaranteed to be at least as large as $\widehat{W}_{i,\ell}(t)$.

Lemma 4 *If $t > (\ell-1)p_i$, then $\widetilde{W}_i(t) \geq \widehat{W}_{i,\ell}(t)$.*

Proof: We will prove the contrapositive. Assume that $\widetilde{W}_i(t) < \widehat{W}_{i,\ell}(t)$. Then,

$$\begin{array}{l} \delta(\tau_i,t) + \sum_{\tau_j \in \mathbf{T}_{H_i}} \delta(\tau_j,t) < \ell e_i + \sum_{\tau_j \in \mathbf{T}_{H_i}} \delta(\tau_j,t) \\ \Rightarrow \delta(\tau_i,t) < \ell e_i \end{array}$$

If $t > (k-1)p_i$, then

$$e_i + \tfrac{te_i}{p_i} < \ell e_i \;\Rightarrow t < (\ell-1)p_i \;\Rightarrow t \leq (\ell-1)p_i$$

Otherwise, if $t \leq (k-1)p_i$, then

$$\left\lceil \tfrac{t}{p_i} \right\rceil e_i < \ell e_i \;\Rightarrow \tfrac{t}{p_i} + 1 \leq \ell \;\Rightarrow t \leq (\ell-1)p_i$$

Since in either case $t \leq (\ell-1)p_i$, we have proved the contrapositive. □

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. Using this function, we will avoid checking potentially $\Theta(d_i/p_i)$ number of jobs for completion at each point in the approximate test.

$$Z_i(t) \stackrel{\text{def}}{=} \max\left(\left\lceil \frac{\widetilde{W}_i(t) - t}{e_i} \right\rceil, 0\right) \tag{14}$$

Observe that for each $\ell > 1$, $\widehat{W}_{i,\ell+1}(t) - \widehat{W}_{i,\ell}(t) = e_i$. So each of the approximate cumulative request-bound functions is separated by e_i units for consecutive jobs of τ_i. $Z_i(t)$ determines the number of jobs $\ell \leq \lceil \frac{t}{p_i} \rceil$ such that $\widehat{W}_{i,\ell}(t) > t$. Therefore, 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

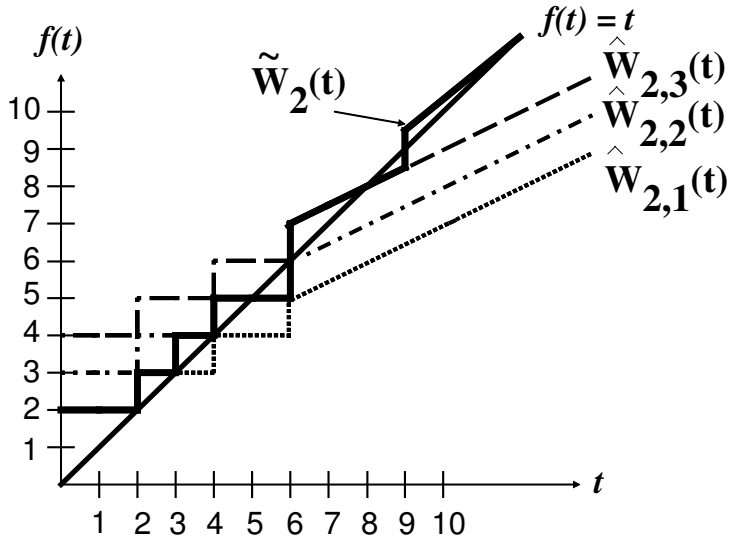

Figure 3. Examples of $\widetilde{W}_2(t)$ and $\widehat{W}_{i,\ell}(t)$ functions for system with $\tau_1 = (p_1, e_1, d_1) = (2, 1, 2)$ and $\tau_2 = (p_2, e_2, d_2) = (3, 1, 4)$, and $k = 4$. The approximation function $\widehat{W}$ for the first three jobs of task τ_2 is shown along with the $\widetilde{W}$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

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.

The following lemma states this formally:

Lemma 5 *If* $Z_i(t) \leq \left\lceil \frac{t}{p_i} \right\rceil - 1$, *then* $\forall \ell \in \{1, \ldots, \left\lceil \frac{t}{p_i} \right\rceil - Z_i(t)\}$, $W_{i,\ell}(t) \leq t$.

Proof: Let $b = \left\lceil \frac{t}{p_i} \right\rceil - Z_i(t)$. Then,

$$\begin{aligned}
\widehat{W}_{i,b}(t) &= \left(\left\lceil \frac{t}{p_i} \right\rceil - Z_i(t)\right) e_i + \textstyle\sum_{\tau_j \in \mathbf{T}_{H_i}} \delta(\tau_j, t) && \text{(by definition of } \widehat{W}) \\
&\leq -Z_i(t) e_i + (\delta(\tau_i, t) + \textstyle\sum_{\tau_j \in \mathbf{T}_{H_i}} \delta(\tau_j, t)) && \text{(by Property 1))} \\
&= \widetilde{W}_i(t) - Z_i(t) e_i && \text{(by definition of } \widetilde{W})
\end{aligned}$$

If $Z_i(t) > 0$, then

$$\begin{aligned}\widehat{W}_{i,b}(t) &\leq \widetilde{W}_i(t) - \left\lceil \frac{\widetilde{W}_i(t)-t}{e_i} \right\rceil e_i \text{ (by definition of } Z_i(t)) \\ &\leq \widetilde{W}_i(t) - \widetilde{W}_i(t) + t \\ &\leq t.\end{aligned}$$

Otherwise, if $Z_i(t) = 0$ then $\widehat{W}_{i,b}(t) \leq \widetilde{W}_i(t)$ and $\left\lceil \frac{\widetilde{W}_i(t)-t}{e_i} \right\rceil \leq 0$. This implies that $\widetilde{W}_i(t) \leq t$. Thus, $\widehat{W}_{i,b}(t) \leq t$. Since, $\forall a < b$, $\widehat{W}_{i,a}(t) \leq \widehat{W}_{i,b}(t)$, then $W_{i,\ell}(t) \leq t$, $\forall \ell \in \{1, \ldots, \left\lceil \frac{t}{p_i} \right\rceil - Z_i(t)\}$, by Lemma 1. □

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 6 *If $t \in ((\ell-1)p_i, (\ell-1)p_i + d_i]$ and $\ell > \left\lceil \frac{t}{p_i} \right\rceil - Z_i(t)$, then $\widehat{W}_{i,\ell}(t) > t$.*

Proof: Let $a = \left\lceil \frac{t}{p_i} \right\rceil - Z_i(t) + 1$. Assume that $\widehat{W}_{i,a}(t) \leq t$. Then,

$$\begin{aligned}\widehat{W}_{i,a}(t) &= ae_i + \textstyle\sum_{\tau_j \in \mathbf{T}_{H_i}} \left\lceil \frac{t}{p_j} \right\rceil e_j \leq t \\ &\Rightarrow \left\lceil \frac{t}{p_i} \right\rceil e_i - Z_i(t)e_i + e_i + \textstyle\sum_{\tau_j \in \mathbf{T}_{H_i}} \left\lceil \frac{t}{p_j} \right\rceil e_j \leq t \\ &\Rightarrow \widehat{W}_{i,a-1}(t) \leq t - e_i \\ &\Rightarrow \frac{\widetilde{W}_i(t)-\widehat{W}_{i,a-1}(t)}{e_i} \geq \frac{\widetilde{W}_i(t)-t+e_i}{e_i} > \left\lceil \frac{\widetilde{W}_i(t)-t}{e_i} \right\rceil - 1 + 1 \\ &\Rightarrow \frac{\widetilde{W}_i(t)-\widehat{W}_{i,a-1}(t)}{e_i} > \left\lceil \frac{\widetilde{W}_i(t)-t}{e_i} \right\rceil\end{aligned}$$

But, notice that $\frac{\widetilde{W}_i(t)-\widehat{W}_{i,a-1}(t)}{e_i} = Z_i(t)$. Since $\widetilde{W}_i(t) - \widehat{W}_{i,a-1}(t) \geq 0$ (otherwise, $\widetilde{W}_i(t) < \widehat{W}_{i,a}(t)$, which implies $t \leq (\ell-1)p_i$ from Lemma 4 and contradicts the assumption on t), $\frac{\widetilde{W}_i(t)-\widehat{W}_{i,a-1}(t)}{e_i} = \left\lceil \frac{\widetilde{W}_i(t)-t}{e_i} \right\rceil$. This is a contradiction of the above statement $\frac{\widetilde{W}_i(t)-\widehat{W}_{i,a-1}(t)}{e_i} > \left\lceil \frac{\widetilde{W}_i(t)-t}{e_i} \right\rceil$. Therefore, $\widehat{W}_{i,a}(t) > t$.

Then $\forall \ell \geq a$, $\widehat{W}_{i,\ell}(t) \geq \widehat{W}_{i,a}(t)$ which implies $\widehat{W}_{i,\ell}(t) > t$ and the lemma follows. □

ApproxFirstStage(τ, i, k):

Step 0: Construct an ordered set $\widetilde{S}_i$ as in Equation (15).

Step 1: Initialize variable *lowest_active* to 1.

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

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

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

ii) If $t' > (lowest\_active - 1)p_i + d_i$, then return "$\tau_i$ *does not always meet all deadlines*"; 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 $lowest\_active := \max(x + 1, lowest\_active)$.

Step 3: Return *lowest_active*.

Figure 4. 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 *does not always meet all deadlines*".

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

$$\widetilde{S}_i \stackrel{\text{def}}{=} \{t = bp_a : a = 1, \ldots, i; b = 1, \ldots, k-1\} \cup \{0\} \qquad (15)$$

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 5 and 6, 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 4 describes this algorithm

ApproxFirstStage in greater detail. Section 4.2 will prove the correctness of ApproxFirstStage.

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 $\widehat{W}_{i,\ell}$,

$$\begin{array}{c}\forall \ell \in \mathbb{N}, \forall t \in \left(\max_{j\in\{1,\ldots,i\}}\{(k-1)p_j\}, \infty\right) :: \\ \left(\widehat{W}_{i,\ell}(t) = \ell e_i + \sum_{\tau_j\in\mathbf{T}_{H_i}} \left(e_j + \frac{te_j}{p_j}\right)\right)\end{array} \tag{16}$$

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 (16) to find point t_ℓ at which $\widehat{W}_{i,\ell}(t_\ell) = t_\ell$.

$$t_\ell \stackrel{\text{def}}{=} \frac{\ell e_i}{1-U_{H_i}} + \frac{\sum_{\tau_j\in\mathbf{T}_{H_i}} e_j}{1-U_{H_i}} \tag{17}$$

where $U_{H_i} \stackrel{\text{def}}{=} \sum_{\tau_j\in\mathbf{T}_{H_i}} \frac{e_j}{p_j}$. 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

$$t_h \leq (h-1)p_i + d_i, \tag{18}$$

then the h^{th} job of τ_i meets its deadline in the synchronous arrival sequence. Otherwise, we declare that τ_i *will not always meet its deadlines.*

If Inequality (18) 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

$$\Delta_\ell \stackrel{\text{def}}{=} [(\ell-1)p_i + d_i] - \left\lceil \frac{\ell e_i}{1-U_{H_i}} + \frac{\sum_{\tau_j \in \mathbf{T}_{H_i}} e_j}{1-U_{H_i}} \right\rceil. \tag{19}$$

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 7 $\forall \ell \in \mathbb{N}(\ell \geq h)$, $\Delta_\ell = \Delta_h - (\ell - h)\left(\frac{e_i}{1-U_{H_i}} - p_i\right)$.

Proof: The proof is by induction on ℓ.

Base Case: Let $\ell = h$. The following statement is vacuously true: $\Delta_h = \Delta_h - (h-h)\left(\frac{e_i}{1-U_{H_i}} - p_i\right)$.

Inductive Step: Assume that the lemma holds for all $b \in \mathbb{N}$ where $h \leq b \leq \ell - 1$. Then,

$$\begin{aligned}
\Delta_\ell &= (\ell-1)p_i + d_i - \tfrac{\ell e_i}{1-U_{H_i}} - \tfrac{\sum_{\tau_j \in \mathbf{T}_{H_i}} e_j}{1-U_{H_i}} \quad \text{(by definition)} \\
&= \left[(\ell-2)p_i + d_i - \tfrac{(\ell-1)e_i}{1-U_{H_i}} - \tfrac{\sum_{\tau_j \in \mathbf{T}_{H_i}} e_j}{1-U_{H_i}}\right] - \left(\tfrac{e_i}{1-U_{H_i}} - p_i\right) \\
&= \Delta_{\ell-1} - \left(\tfrac{e_i}{1-U_{H_i}} - p_i\right) \\
&= \Delta_h - (\ell-h-1)\left(\tfrac{e_i}{1-U_{H_i}} - p_i\right) - \left(\tfrac{e_i}{1-U_{H_i}} - p_i\right) \\
&\quad \text{(by inductive hypothesis)} \\
&= \Delta_h - (\ell-h)\left(\tfrac{e_i}{1-U_{H_i}} - p_i\right)
\end{aligned}$$

□

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

Lemma 8 *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* $\frac{e_i}{1-U_{H_i}} > p_i$.

Proof: We will prove the *only if* part, first. Assume that $t_\ell > (\ell - 1)p_i + d_i$ and $\ell > h$. By equation (18),

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

Observe, by Equation (17), $t_\ell - t_h = (\ell-h)\frac{e_i}{1-U_{H_i}}$. So $(\ell-h)\frac{e_i}{1-U_{H_i}} > (\ell-h)p_i$, which implies $\frac{e_i}{1-U_{H_i}} > p_i$.

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

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

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

$$\begin{aligned} \Delta_\ell &= \Delta_h - (\ell-h)\left(\tfrac{e_i}{1-U_{H_i}} - p_i\right) \quad \text{(from Lemma 7)} \\ &= \Delta_h - \left(\left\lceil \frac{\Delta_h}{\left(\frac{e_i}{1-U_{H_i}}\right)-p_i} \right\rceil + 1\right)\left(\tfrac{e_i}{1-U_{H_i}} - p_i\right) \ \text{(from definition of } \ell) \\ &\leq \Delta_h - \left(\frac{\Delta_h}{\left(\frac{e_i}{1-U_{H_i}}\right)-p_i} + 1\right)\left(\tfrac{e_i}{1-U_{H_i}} - p_i\right) \\ &= \Delta_h - \Delta_h - \left(\tfrac{e_i}{1-U_{H_i}} - p_i\right) \\ &< 0 \quad \text{(from assumption)} \end{aligned}$$

$\Delta_\ell < 0$ 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 determine whether the h^{th} job and all subsequent jobs of task τ_i always meet their deadlines (according to the approximation) by testing inequality (18) and checking that $\frac{e_i}{1-U_{H_i}} \leq p_i$. Figure 5 gives the pseudo-code for the algorithm ApproxSecondStage. Finally, Figure 6 describes the full approximation scheme for feasibility of sporadic static-priority task systems with respect to a given priority assignment.

ApproxSecondStage(τ, i, k, h):

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

Step 1: If $t_h > (h-1)p_i + d_i$, return τ_i *does not always meet all deadlines.*

Step 2: If $\frac{e_i}{1-U_{H_i}} > p_i$, return τ_i *does not always meet all deadlines.*

Step 3: Return τ_i *always meets all deadlines.*

Figure 5. 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 *always meets all deadlines*, else it returns τ_i *does not always meet all deadlines.*

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 does not always meet all deadlines", then set $h :=$ ApproxFirstStage(τ, i, k). Else return τ *is infeasible.*

b) If ApproxSecondStage(τ, i, k, h) returns "τ_i does not always meet all deadlines", then return τ *is infeasible.*

Step 2: Return τ *is feasible.*

Figure 6. 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.

4.2. Proof of Correctness for Arbitrary Task Systems

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 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 prove a claim about the connection between the total system utilization and the slope of the $\widehat{W}$ function. This claim is used to prove an important invariant of ApproxFirstStage (Lemma 9). 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 10) that Approx will return "τ is feasible" *if and only if* the approximate cumulative request-bound function for each jobs of each task has a fixed point prior to each job's deadline. We then invoke a result from (Lehoczky, 1990) to arrive at the stated goal, above.

First, let us prove a claim that will be useful in proving the invariant of ApproxFirstStage. In words, the claim states that if the total system utilization is at most one, then the slope of the linear interpolation of $\widehat{W}_{i,\ell}(t)$ is at most one. Formally,

Claim 2 *For any adjacent t_1,$t_2 \in \widetilde{S}_i$ (where $t_1 < t_2$), if the system utilization $U = \sum_{j=1}^{n} \frac{e_j}{p_j} \leq 1$, then $\frac{\widehat{W}_{i,\ell}(t_2) - \widehat{W}_{i,\ell}(t_1) - r_i(t_1)}{t_2 - t_1} \leq 1$.*

Proof: We prove the contrapositive of the claim. Assume that

$$\frac{\widehat{W}_{i,\ell}(t_2) - \widehat{W}_{i,\ell}(t_1) - r_i(t_1)}{t_2 - t_1} > 1.$$

Then,

$$\begin{aligned} t_2 - t_1 &< \widehat{W}_{i,\ell}(t_2) - \widehat{W}_{i,\ell}(t_1) - r_i(t_1) \\ &= \textstyle\sum_{\tau_j \in \mathbf{T}_{H_i}} [\delta(\tau_j, t_2) - \delta(\tau_j, t_1)] - r_i(t_1) \\ &= \textstyle\sum_{\{\tau_j \in \mathbf{T}_{H_i} : t_2 > (k-1)p_j\}} [\delta(\tau_j, t_2) - \delta(\tau_j, t_1)] \\ &\quad + \textstyle\sum_{\{\tau_j \in \mathbf{T}_{H_i} : (t_2 \leq (k-1)p_j) \wedge (p_j | t_1)\}} [\delta(\tau_j, t_2) - \delta(\tau_j, t_1)] \\ &\quad + \textstyle\sum_{\{\tau_j \in \mathbf{T}_{H_i} : (t_2 \leq (k-1)p_j) \wedge (p_j \nmid t_1)\}} [\delta(\tau_j, t_2) - \delta(\tau_j, t_1)] - r_i(t_1) \end{aligned}$$

If $t_2 \leq (k-1)p_j$ and p_j divides t_1, a job of τ_j is released at time t_1; since t_1 and t_2 are adjacent, $\delta(\tau_j, t_2) - \delta(\tau_j, t_1) = e_j$. Otherwise, if $t_2 \leq (k -$

$1)p_j$ and p_j does not divide t_1, no job of τ_j is released at t_1; therefore, $\delta(\tau_j, t_2) - \delta(\tau_j, t_1) = 0$. Observe, $\sum_{\{\tau_j \in \mathbf{T}_{H_i} : (t_2 \leq (k-1)p_j) \wedge (p_j | t_1)\}} [\delta(\tau_j, t_2) - \delta(\tau_j, t_1)] - r_i(t_1) \leq 0$. Therefore,

$$\begin{aligned}\widehat{W}_{i,\ell}(t_2) - \widehat{W}_{i,\ell}(t_1) - r_i(t_1) &< \sum_{\{\tau_j \in \mathbf{T}_{H_i} : t_2 > (k-1)p_j\}} [\delta(\tau_j, t_2) - \delta(\tau_j, t_1)] \\ &\leq \sum_{\{\tau_j \in \mathbf{T}_{H_i} : t_2 > (k-1)p_j\}} (t_2 - t_1)\frac{e_j}{p_j} \\ &\leq (t_2 - t_1)U\end{aligned}$$

$$\Rightarrow \frac{\widehat{W}_{i,\ell}(t_2) - \widehat{W}_{i,\ell}(t_1) - r_i(t_1)}{t_2 - t_1} < U$$

The above inequality implies that the total system utilization is greater than one. □

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

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

Proof: The proof is by induction on a. Let t_a be the a^{th} element in the ordered set $\widetilde{S}_i$ (t_0 represents element 0).

Base Case: Let $a = 1$. Since $lowest\_active_a$ is initialized to 1 prior to first iteration, the condition is vacuously true.

Inductive Step: Assume that the condition is true after the first $a-1$ iterations of the *for* loop, and prior to the a^{th} iteration where $1 < a < |\widetilde{S}_i| - 1$. We must show then that the following is true:

$$\begin{aligned}&\forall \ell \in \{lowest\_active_a, \ldots, lowest\_active_{a+1} - 1\}, \\ &\quad \exists t \in ((\ell - 1)p_i, (\ell - 1)p_i + d_i] : \ \widehat{W}_{i,\ell}(t) \leq t\end{aligned}$$

Assume that ApproxFirstStage does not terminate during the a^{th} iteration. If $lowest\_active_{a+1} = lowest\_active_a$, the above condi-

tion is vacuously true by the inductive step. Therefore, assume that $lowest\_active_{a+1} > lowest\_active_a$. This implies that $\left\lceil \frac{t_a}{p_i} \right\rceil - Z_i(t_a) > 0$. We know from Lemma 5 that

$$\forall \ell \in \{1, \ldots, lowest\_active_{a+1} - 1\},\ W_{i,\ell}(t_a) \leq t_a$$

Notice at most one job's deadline elapses between t_{a-1} and t_a, the only job whose deadline could elapse is $lowest\_active_a$. Therefore,

$$\begin{aligned} &\forall \ell \in \{lowest\_active_a + 1, \ldots, lowest\_active_{a+1} - 1\}, \\ &\quad \exists t \in ((\ell - 1)p_i, (\ell - 1)p_i + d_i] :\ \widehat{W}_{i,\ell}(t) \leq t \end{aligned}$$

We need to only check if $lowest\_active_a$ met its deadline. It now remains to be shown that

$$\begin{aligned} &\exists t \in ((lowest\_active_a - 1)p_i, (lowest\_active_a - 1)p_i + d_i] \\ &\quad :: (\widehat{W}_{i,lowest\_active_a}(t) \leq t) \end{aligned} \tag{20}$$

If $(lowest\_active_a - 1)p_i + d_i \geq t_a$, then Equation (20) is true. Otherwise, we execute Step 2.*a.i*. Since ApproxFirstStage does not terminate in the a^{th} step, there exists a t', calculated by Step 2.*a.i*, such that $t' \leq (lowest\_active_a - 1)p_i + d_i$ and $\widehat{W}_{i,lowest\_active_a}(t') = t'$. If $t' \geq t_{a-1}$, then $t' > (lowest\_active_a - 1)p_i$ and (20) is true. If $t' < t_{a-1}$, then we know by definition of t' and Claim 2 that $\frac{\widehat{W}_{i,lowest\_active_a}(t_a) - \widehat{W}_{i,lowest\_active_a}(t')}{t_a - t'}$ is equal to

$$\frac{\widehat{W}_{i,lowest\_active_a}(t_a) - \widehat{W}_{i,lowest\_active_a}(t_{a-1}) - r_i(t_{a-1})}{t_a - t_{a-1}} \leq 1.$$

Thus, $\widehat{W}_{i,lowest\_active_a}(t_a) \leq t_a$.

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

□

(Lehoczky, 1990) 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 always meets all deadlines. We will use this result to show that the

task set τ is feasible when $\mathsf{Approx}(\tau, \epsilon)$ returns "τ is feasible," and τ is infeasible on a processor of $(1-\epsilon)$ capacity when $\mathsf{Approx}(\tau, \epsilon)$ returns "τ is infeasible." We restate Lehoczky's results in the following theorem.

Theorem 5 (from (Lehoczky, 1990**))** *A sporadic 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 state a corollary that follows from Lemma 3. The corollary simply shows that Lemma 3 holds for the set $\widetilde{S}_i$. The following corollary will be used in the proof of Lemma 10:

Corollary 1 *For adjacent elements, $t_1,\ t_2 \in \widetilde{S}_i$, if $\widehat{W}_{i,\ell}(t_1) > t_1$ and $\widehat{W}_{i,\ell}(t_2) > t_2$, then $\widehat{W}_{i,\ell}(t) > t$, $\forall t$ 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 10 $\mathsf{Approx}(\tau, \epsilon)$ *returns "τ is feasible"* if and only if

$$\begin{aligned} &\forall \tau_i \in \tau,\ \ell \in \mathbb{N}(\ell > 0) :: \\ &\quad (\exists t \in ((\ell-1)p_i, (\ell-1)p_i + d_i] : \widehat{W}_{i,\ell}(t) \leq t). \end{aligned} \tag{21}$$

Proof: Proving the "only if" direction first, assume that $\mathsf{Approx}(\tau, \epsilon)$ returns "τ is feasible." Consider any task $\tau_i \in \tau$. Let h_i be the value returned from $\mathsf{ApproxFirstStage}(\tau, i, k)$. Notice that h_i is equal to *lowest_active* at the last iteration of $\mathsf{ApproxFirstStage}(\tau, i, k)$. Thus, by Lemma 9,

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

Since, $\mathsf{Approx}(\tau, \epsilon)$ returns "τ is feasible", it must be the case that both,

$$t_{h_i} \leq (h_i - 1)p_i + d_i, \ \textbf{and}\ \frac{e_i}{1 - U_{H_i}} \leq p_i.$$

If either condition is false, ApproxSecondStage(τ, i, k, h_i) would return "τ_i does not always meet all deadlines." This would contradict our assumption that Approx(τ, ϵ) returns "τ is feasible."

Notice, $t_{h_i} \leq (h_i - 1)p_i + d_i$ implies by definition of t_{h_i} that $\exists t \in ((h_i - 1)p_i, h_i - 1)p_i + d_i]$ such that $\widehat{W}_{i,h_i}(t) \leq t$. Also, by Lemma 8,

$$\begin{array}{ll} \frac{e_i}{1-U_{H_i}} \leq p_i & \\ \Rightarrow & \forall \ell (\in \mathbb{N}) > h,\ t_\ell \leq (\ell - 1)p_i + d_i \\ \Rightarrow & \forall \ell (\in \mathbb{N}) > h,\ \exists t \in ((\ell - 1)p_i, (\ell - 1)p_i + d_i] : \widehat{W}_{i,\ell}(t) \leq t \end{array}$$

We can see by combining the set of jobs for which $\exists t \in ((\ell-1)p_i, (\ell-1)p_i + d_i]$ such that $\widehat{W}_{i,\ell}(t) \leq t$, we have proven:

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

We have, thereby, completed the proof of the "only if" direction.

Now we will prove "if" direction by proving the contrapositive. In other words, we will assume that Approx(τ, ϵ) returns "τ is infeasible." It must be the case that some task τ_i of task system τ has been declared that "τ_i dos not always meet all deadlines" by either ApproxFirstStage or ApproxSecondStage. There are two cases:

1) ApproxFirstStage *returns "τ_i does not always meet all deadlines"*:
Let $a > 0$ be the iteration of Step 2 in ApproxFirstStage that returns τ_i *does not always meet all deadlines*. By Step 2.*a.i*, t' is defined by the intersection of $f(t) = t$ and the line consisting of points $(t_a, \widehat{W}_{i,lowest\_active_a}(t_a))$ and $(t_{a-1}, \widehat{W}_{i,lowest\_active_a}(t_{a-1}) + r_i(t_{a-1}))$. $t' > (lowest\_active_a - 1)p_i + d_i$ by Step 2.*a.ii*; therefore, for all $t \in (t_{a-1}, (lowest\_active_a - 1)p_i + d_i]$, $\widehat{W}_{i,lowest\_active_a}(t) > t$.

   To prove the remainder of Case 1, we must show that for all $t \in ((lowest\_active_a - 1)p_i, t_{a-1}]$, $\widehat{W}_{i,lowest\_active_a}(t) > t$. Let J_x be the elements of $\widetilde{S}_i$ that intersect with the time interval during which the x^{th} job of task τ_i is active. More formally,

$$J_x \stackrel{\text{def}}{=} \left\{ t \in \widetilde{S}_i \mid (x-1)p_i < t \leq (x-1)p_i + d_i \right\}.$$

Note that each element of $J_{lowest\_active_a}$ represents an iteration of Step 2 of ApproxFirstStage where job $lowest\_active_a$ is active. If for any element $t \in J_{lowest\_active_a}$, $lowest\_active_a \leq \left\lceil \frac{t}{p_i} \right\rceil - Z_i(t)$, by Lemma 5 this would contradict the variable $lowest\_active_a$ at iteration a (since $lowest\_active_a$ is non-decreasing). So, $lowest\_active_a > \left\lceil \frac{t}{p_i} \right\rceil - Z_i(t)$, $\forall t \in J_{lowest\_active_a}$. By Lemma 6 this implies $\forall t \in J_{lowest\_active_a}$, $\widehat{W}_{i,lowest\_active_a}(t) > t$. By Corollary 1, for all $t \in ((lowest\_active_a - 1)p_i, t_{a-1}]$, $\widehat{W}_{i,lowest\_active_a}(t) > t$. We can then conclude that

$$\forall t \in ((lowest\_active_a - 1)p_i, (lowest\_active_a - 1)p_i + d_i] : \\ \widehat{W}_{i,lowest\_active_a}(t) > t.$$

2) ApproxSecondStage *returns "τ_i does not always meet all deadlines"*:
We analyze the two possibilities and show in each case that there exists a job h of task τ_i such that for all t where h is active $\widehat{W}_{i,h}(t) > t$.

a) Step 1 of ApproxSecondStage returns *"τ_i does not always meet all deadlines"*:

Let h be the value returned from ApproxFirstStage. If $h < k$, job h's deadline must be strictly after $\max_{j \in \{1,\ldots,i\}}\{(k-1)p_j\}$ (otherwise, the fact that ApproxFirstStage returned h is contradicted). So, for all $t \in ((h-1)p_i, \max_{j \in \{1,\ldots,i\}}\{(k-1)p_j\}]$, $\widehat{W}_{i,h}(t) > t$. In other words, for the h^{th} job of τ_i, $\widehat{W}_{i,h}(t)$ does not fall below t prior to $\max_{j \in \{1,\ldots,i\}}\{(k-1)p_j\}$. However, since $t_h > (h-1)p_i + d_i$, for all $t \in (\max_{j \in \{1,\ldots,i\}}\{(k-1)p_j\}, (h-1)p_i + d_i]$, $\widehat{W}_{i,h}(t) > t$. Therefore,

$$\forall t \in ((h-1)p_i, (h-1)p_i + d_i], \widehat{W}_{i,h}(t) > t.$$

b) Step 2 of ApproxSecondStage returns *"τ_i does not always meet all deadlines"*:

By Lemma 8 and definition of t_ℓ,

$$\exists \ell (\ell > h) \in \mathbb{N} :: (\forall t \in ((\ell - 1)p_i, (\ell - 1)p_i + d_i], \widehat{W}_{i,\ell}(t) > t).$$

We have shown in each of the above cases that if $\mathsf{Approx}(\tau, \epsilon)$ returns "τ is infeasible", then

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

□

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

Theorem 6 *A sporadic, 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 10,

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

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 5. □

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

Theorem 7 *If for a sporadic 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 10,

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

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 always meets all deadlines on a processor of $(1-\epsilon)$ capacity, Theorem 5 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 6 and 7, Approx is correct.

4.3. 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:

$$1 + i(k-1) \tag{22}$$

This corresponds to the number of iterations that ApproxFirstStage must make for each task, τ_i. In our implementation of approximate feasibility-analysis, the condition in Step 2 of ApproxFirstStage would be executed at most $\Sigma_{i=1}^{n}(1+(i)(k-1))$ times, which is $\mathrm{O}(n^2k)$. (Note that the functions $\widehat{W}_{i,\ell}$, $\widetilde{W}_i$, and Z_i can be precomputed for the points in $\widetilde{S}_i$ in $\mathrm{O}(ik)$ time; the total time to precompute these functions for all tasks of τ is $\mathrm{O}(n^2k)$. Therefore, the evaluation of each of these functions in Approx requires constant time.)

5. A Fully Polynomial-Time Approximation Scheme

5.1. Approximation Schemes

For sporadic task systems, the precise computational complexity of determining if a task set is feasible for a given uniprocessor system is currently unknown. However, since current tests for feasibility in this model require at least pseudo-polynomial time, we have directed our attention at developing approximations that require only polynomial-time. As traditionally defined, an algorithm $\mathcal{A}$ for a given optimization problem is an *approximation scheme* if given input I, and accuracy parameter $\epsilon > 0$ it produces a solution that is a bounded factor (in terms of ϵ) away from the optimal solution. Specifically, if OPT is the optimal value for the given minimization problem, we require the "cost" of the solution produced by $\mathcal{A}$ to not exceed $(1 + \epsilon) \cdot OPT$ (for a maximization problem the solution must not be less $(1 - \epsilon) \cdot OPT$). A *polynomial-time approximation scheme* is an approximation scheme that produces a solution for each *constant* $\epsilon > 0$ in time polynomially bounded in terms of $|I|$. A *fully polynomial-time approximation scheme* is an approximation scheme that produces a solution for arbitrary $\epsilon > 0$ in running time that is polynomially bounded by $|I|$ and $1/\epsilon$. For further general information on approximation schemes see (Vazirani, 2001).

However, the feasibility problem for static-priority scheduling on uniprocessors is not an optimization problem; hence, the notion of approximation employed in this paper is as follows. The $(1 - \epsilon)$ term quantifies the capacity of the processor we must "sacrifice" for the approximation algorithm result to be correct. In other words, if our approximation returns "feasible", the task system is guaranteed to be feasible on the specified uniprocessor. However, if the approximate test returns "infeasible", the task system is only guaranteed to be infeasible on a processor of $(1-\epsilon)$ of the original capacity. We will see in the next subsection the implications of our FPTAS and that its running time is polynomially bounded in terms of n and $1/\epsilon$.

For a given accuracy, ϵ, the running time of the approximate feasibility test 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 sporadic tasks systems in a static-priority system. The following theorem states this formally.

Theorem 8 *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 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.*

□

The synchronous arrival sequence exactly models the job arrival sequence in a *synchronous periodic* task system. In a synchronous periodic task system, the first job of each task arrives at time instant zero, and subsequent jobs of each task τ_i arrive every p_i time units. Therefore, it is useful to note that the results of Theorem 8 trivially extend to the synchronous periodic task model without modification.

6. Summary

It has been shown (Albers and Slomka, 2004) 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 for the implicit-deadline, constrained, and arbitrary task model. 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

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

N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings. Hard Real-Time Scheduling: The Deadline Monotonic Approach. In *Proceedings 8th IEEE Workshop on Real-Time Operating Systems and Software*, pages 127–132, Atlanta, May 1991.

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

G. C. Buttazzo. Rate monotonic vs. EDF: Judgment day. *Real-Time Systems*, 29(1):5–26, 2005.

M. Dertouzos. Control robotics : the procedural control of physical processors. In *Proceedings of the IFIP Congress*, pages 807–813, 1974.

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

J. P. Lehoczky. Fixed priority scheduling of periodic tasks with arbitrary deadlines. In *IEEE Real-Time Systems Symposium*, pages 201–209, December 1990.

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

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

A. K. Mok. *Fundamental Design Problems of Distributed Systems for The Hard-Real-Time Environment.* PhD thesis, Laboratory for Computer Science,

Massachusetts Institute of Technology, 1983. Available as Technical Report No. MIT/LCS/TR-297.

A. Mok. Task management techniques for enforcing ED scheduling on a periodic task set. In *Proc. 5th IEEE Workshop on Real-Time Software and Operating Systems*, pages 42–46, Washington D.C., May 1988.

D. Mossé, T. Baker, S. Baruah, G. Buttazzo, A. Burns, L. Sha, and J. Stankovic. Fixed or dynamic priority? That is the question (panel discussion). In *Proceedings of the Real-Time Systems Symposium*, page 9, Lisbon, Portugal, December 2004. IEEE Computer Society Press.

Vijay V. Vazirani. *Approximation Algorithms*. Springer-Verlag, Berlin-Heidelberg-New York-Barcelona-Hong Kong-London-Milan-Paris-Singapur-Tokyo, 2001.