A necessary and sometimes sufficient condition for the feasibility of sets of sporadic hard-deadline tasks

Theodore P. Baker*
Department of Computer Science
Florida State University
Tallahassee, FL 32306-4530
e-mail: baker@cs.fsu.edu

Michele Cirinei†
Scuola Superiore Sant'Anna
Pisa, Italy
e-mail: cirinei@gandalf.sssup.it

Abstract

This report describes a necessary condition for feasibility of scheduling a set of sporadic hard-deadline tasks on identical multiprocessor platforms, which is also a sufficient condition if there is only a single processor. The key contribution is the characterization of the maximum, over all time intervals of a given length, of the amount of computation that must be completed to meet all deadlines, and a method of computing this function efficiently to any desired degree of accuracy. Empirical data are provided to verify that the new infeasibility test can be computed efficiently and is an improvement over previously known checks for infeasibility.

1 Introduction

This report describes a necessary condition for feasibility of scheduling a set of independent sporadic hard-deadline tasks. The key contribution is the characterization of the maximum, over all time intervals of a given length, of the amount of computation that must be completed to meet all deadlines, and a method of computing this function efficiently.

A *sporadic task set* τ is a collection of sporadic tasks $\{\tau_1, \tau_2, \ldots, \tau_n\}$. Each sporadic task $\tau_i = (e_i, d_i, p_i)$ generates a potentially infinite sequence of *jobs*, characterized by a *maximum (worst case) execution time requirement* e_i, a maximum response time (relative deadline) d_i, and a *minimum release time separation* (period) p_i.

A *release time sequence* r_i for a sporadic task τ_i is a finite or infinite sequence of times $r_{i,1} < r_{i,2} < \cdots$ such that $r_{i,j+1} - r_{i,j} \geq p_i$, for $j = 1, 2, \ldots$. A *release time assignment* r for a task set is a mapping of release time sequences r_i to tasks τ_i, one for each of the tasks in τ.

A sporadic task set is *feasible* on m processors if, for each release time assignment r there is a schedule for m processors that meets all the task deadlines, *i.e.*, such that τ_i executes for e_i units of time in each interval $[r_{i,j}, r_{i,j} + d_i)$, for $i = 1, \ldots, n$ and $j = 1, 2, \ldots$. It is assumed that $e_i \leq \min(d_i, p_i)$ for every task τ_i, since otherwise a task system would be trivially infeasible.

A task system is *schedulable* according to a given scheduling policy if the policy produces a schedule that meets all the task deadlines.

Several sufficient tests have been derived for the schedulability of a sporadic task set on a multiprocessor using a given scheduling policy, such as global preemptive scheduling based on fixed task priorities (FTP) or deadlines (EDF) [2, 3, 4, 10, 11, 13, 15]. For example, it can be shown that a set of independent periodic tasks with deadline equal to period will not miss any deadlines if it is scheduled by a global EDF policy on m processors, provided the sum of the processor utilizations does not exceed $m(1 - u_{\max}) + u_{\max}$, where $u_{\max}$ is the maximum single-task processor utilization.

One difficulty in evaluating and comparing the efficacy of such schedulability tests has been distinguishing the causes of failure. That is, when one of these schedulability tests is unable to verify that a particular task set is schedulable there are three possible explanations:

1. The problem is with the task set, which is *not feasible*, *i.e.*, not able to be scheduled by any policy.
2. The problem is with the scheduling policy. The task set is *not schedulable* by the given policy, even

*This material is based upon work supported in part by the National Science Foundation under Grant No. 0509131, and a DURIP grant from the Army Research Office.

†Visiting at the Florida State University, supported by PARADES.

though the task set is feasible.

3. The problem is with the test, which is *not able to verify* the fact that the task set is schedulable by the given policy.

To the best of our knowledge, there are no known algorithms other than "brute force" enumeration that can distiguish the above three cases.[1]x

The following facts are fairly well known:

1. A sporadic task set $\tau = \{\tau_1, \ldots, \tau_n\}$ is feasible on m processors (with ideal processor sharing) if $\lambda_{sum}(\tau) \leq m$, where

$$\lambda_{sum}(\tau) \stackrel{\text{def}}{=} \sum_{i=1}^{n} \lambda_i = \sum_{i=1}^{n} \frac{e_i}{\min(p_i, d_i)}$$

2. A sporadic task set is not feasible on m processors if $u_{sum}(\tau) > m$, where

$$u_{sum}(\tau) \stackrel{\text{def}}{=} \sum_{i=1}^{n} u_i = \sum_{i=1}^{n} \frac{e_i}{p_i}$$

Baruah and Fisher[7] showed that $\delta_{sum}(\tau) \leq m$ is a necessary condition for the feasibility of the task set τ on a platform with m unit-capacity processors, where

$$\delta_{sum}(\tau) \stackrel{\text{def}}{=} \operatorname*{lub}_{t>0} \frac{\sum_{i=1}^{n} \text{DBF}(\tau_i, t)}{t}$$

and

$$\text{DBF}(\tau_i, t) \stackrel{\text{def}}{=} \max(0, (\lfloor \frac{t - d_i}{p_i} \rfloor + 1)e_i)$$

Fisher, Baker, and Baruah[12] showed how to approximate the load bound function $\delta_{sum}(\tau)$ efficiently, in polynomial time, and that the criterion $\delta_{sum} > m$ was significantly more effective than the criterion $u_{sum} > m$ as a test of infeasibility when tested on several large collections of pseudo-randomly generated task sets.

The present paper derives an improvement on the above load bound function, which allows the detection of a strictly larger range of infeasible task sets, including the example below.

Example 1 Consider the task system below.

i	p_i	e_i	d_i	$\text{DBF}(\tau_i, 1)$
1	4	2	2	0
2	2	1	1	1
3	2	1	1	1

□

[1]For example, [5] takes exponential time and only distinguishes (2) from (3).

This task set is infeasible on $m = 2$ processors, but $\delta_{sum}(\tau) = 2$. The problem is that $\text{DBF}(\tau_1, 1) = 0$ under-estimates the real demand of task τ_1 in the interval $[0, 1)$. Task τ_1 must execute for one unit of time in the interval$[0, 1)$ in order to meet its deadline at 2. The effective combined demand of the three tasks over this interval should be 3, not 2. The phenomenon observed above was recognized previously by Johnson and Maddison [14]), who used the term "throwforward" to describe the amount of execution time that a task with later deadline, like τ_1, must complete before an earlier deadline of another task, like τ_2 and τ_3.

The new load bound function $m\ell(\tau)$ defined here is similar to $\delta_{sum}(\tau)$, but differs by using $\text{DBF}(\tau_i, t) + \max(0, t - (jp_i + d_i - e_i))$ instead of $\text{DBF}(\tau_i, t)$. The additional term corrects for cases of under-estimation of the actual worst-case computational load of an interval like the example above, by taking into account the throwforward of jobs whose deadlines may occur past the end of the interval.

This paper makes the following contributions:

1. It shows how to recognize a significant number of infeasible task sets, by computing a new load-bound function and determining whether the load bound exceeds the available number of processors.
2. It shows that the new load bound retains the property of the $\delta_{sum}(\tau)$ load bound of Baruah, Mok, and Rosier[9] that $m\ell(\tau) \leq 1$ is a necessary and sufficient condition for single-processor feasibility, and a necessary and sufficient test of single-processor EDF schedulability.
3. It provides empirical evidence of the degree of improvement in ability to detect infeasible task sets using the new load-bound function, as compared to the previously-defined load bound function $\delta_{sum}(\tau)$.
4. It provides an algorithm for computing the new load bound function to any specified degree of accuracy within polynomial time.
5. It provides empirical evidence that the new algorithm can be computed at least as efficiently as the best previously known algorithm for computing $\delta_{sum}(\tau)$.

2 Time Model

For mathematical convenience, points and durations in real time are modeled by real numbers. However, in an actual system time is not infinitely divisible. The times of event occurrences and durations between them cannot be determined more precisely than one tick of

the system's most precise clock. Therefore, any time value t involved in scheduling is assumed to be a non-negative integer value and is viewed as representing the entire interval

$$[t, t+1) \stackrel{\text{def}}{=} \{x \in \mathbb{R} | t \leq x < t+1\}$$

The notation $[a, b)$ is used for time intervals as a reminder that the interval includes all of the clock tick starting at a but does not include the clock tick starting at b.

These conventions allow the use of mathematical induction on clock ticks for proofs, avoid potential confusion around end-points, and prevent impractical schedulability results that rely on being able to slice time at arbitrary points.

3 The maxmin load

Given a sporadic task τ_i and a possibly infinite sequence $r_{i,1}, r_{i,2}, \ldots$ of release times that are compatible with the sporadic arrival constraint p_i, the *minimum demand* of τ_i in any specific time interval is defined to be the minimum amount of time that τ_i must execute within that interval in order to meet all its deadlines.

Note that this definition of the minimum demand of a task does not presume any specific scheduling policy, and it takes into account release times and deadlines both inside and outside the interval. In the latter respect this definition of minimum demand is different from the definition of demand on which the definition of δ_{sum} above is based; in δ_{sum} only tasks with deadlines and release times inside the interval are considered.

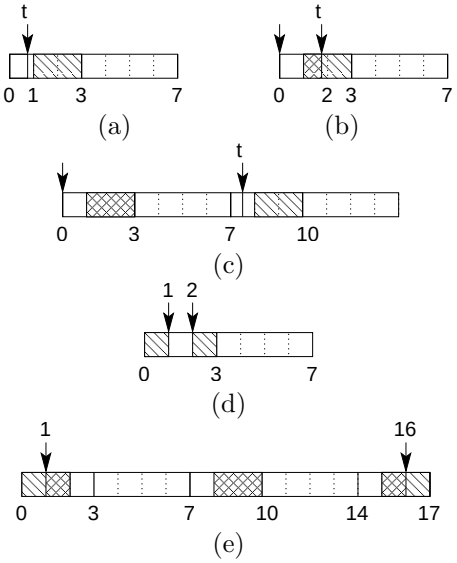

Figure 1. minimum demand examples

Example 2 Consider a task $\tau_i = (2, 3, 7)$ (with $e_i = 2$, $d_i = 3$, $p_i = 7$) and the sequence of release times $0, 7, 14$. Figure 1 shows the minimum demand of τ_i for several intervals. The diagonally shaded areas show the most favorable position of τ_i's execution in the schedule for minimizing the work done in the given interval while still meeting deadlines. The cross-hatched areas indicate the portion of that work that cannot be moved outside the given interval without missing a deadline.

(a) The minimum demand of the interval $[0, t)$ is zero for $t \leq 1$, since it is possible to meet all deadlines and not start any jobs of τ_i before time 1.

(b) The minimum demand of the interval $[0, t)$ is $2 - (3 - t)$ for $1 < t \leq 3$, since the first job of τ_i can execute for at most $3 - t$ time between t and the deadline 3.

(c) The minimum demand of the interval $[0, t)$ is 2 for $3 < t \leq 8$, since execution of the second job does not need to start until time 8 in order to meet the deadline at time 10.

(c) The minimum demand of the interval $[1, 2)$ is zero, since half the execution of the first job can be done before the start of the interval and the other half can be postponed until after the interval.

(e) The minimum demand of the interval $[1, 16)$ is 4, since the first job cannot do more than one unit of execution before time 1, the second job cannot be postponed past 10, and the third job cannot postpone more than one unit of execution past 16.

□

Given a sporadic task τ_i and a time duration t, the *maxmin-demand* $md(\tau_i, t)$ of τ_i for intervals of length t is defined to be the maximum of the minimum demand of $[a, a+t)$, taken over all release time assignments and all interval start times $a \geq 0$.

The *maxmin load* of a set τ of n sporadic tasks is

$$m\ell(\tau) \stackrel{\text{def}}{=} \operatorname*{lub}_{t \geq 0} \sum_{i=1}^{n} md(\tau_i, t)/t$$

From the requirement that $e_i \leq \min(d_i, p_i)$, it is clear that $md(\tau_i, t)/t \leq 1$, and so the above least upper bound is well defined.

For purposes of analysis, it is helpful to think in terms of intervals that start at time zero.

Given a sporadic task τ_i and a time duration t, the *canonical demand* $cd(\tau_i, t)$ of τ_i for intervals of length t is defined to be the minimum demand of the interval $[0, t)$ with periodic releases starting at time zero, *i.e.*, $r_{i,j} = jp_i$, $j = 0, 1, 2, \ldots$.

Theorem 1 (Critical zone) *For any set τ of sporadic tasks and any $t > 0$, $md(\tau_i, t) = cd(\tau_i, t)$.*

proof: Let r' be any release time sequence and $[a, a+t)$ be any interval of length t. Consider any single task τ_i. It is enough to show that the minimum demand of τ_i in $[a, a + t)$ under r' is no greater than the minimum demand of τ_i in $[0, t)$ under the canonical release time sequence $r_{i,j} = jp_i$, $j = 0, 1, 2, \ldots$. This is done by a series of modifications to the release-time sequence r', each of which does not reduce the minimum demand.

Without loss of generality, delete from r' all the release times that do not contribute to the minimum demand of τ_i in $[a, a + t)$, and let $b \stackrel{\text{def}}{=} r'_{i,1}$ be the release time of the first job of τ_i that contributes to that minimum demand.

The next step is to show that the minimum demand of τ_i in $[a, a + t)$ under r' is no greater than the minimum demand with strictly periodic releases starting at b. If there are any release times in r' that are farther apart than p_i, shifting those releases closer to the start of the interval $[a, a + t)$ cannot decrease the minimum demand in the interval. The sporadic minimum separation constraint does not allow releases to be any closer together than p_i. Therefore, it is sufficient to limit consideration to cases where $r'_{i,j} = b + jp_i$.

The next step is to show that the minimum demand in $[a, a + t)$ will not be decreased by shifting all the release times so that the first release occurs at a. If $b \geq a$, it is clear that the minimum demand in $[a, a+t)$ will not be decreased by next shifting all the release times down by $b - a$. Therefore, it only remains to show that if $b < a$ the minimum demand in $[a, a + t)$ will not be decreased by shifting all the release times up by $a - b$, so that the first release occurs at a.

Note that the shift has different effects near the beginning and the end of the interval, and in particular the minimum demand of the task

- near a tends to be increased by the shift;
- near $a + t$ tends to be decreased by the shift.

In order to show that the overall minimum demand in $[a, a+t)$ is not decreased, we consider separately the consequences of the shift at the beginning and at the end of the interval $[a, a + t)$, and we compare the two results.

Since the job of τ_i released at b contributes to the minimum demand of τ_i in $[a, a + t)$, $b + e_i > a$. The next release of τ_i is at $b + p_i \geq b + e_i > a$. Moreover, due to the sporadic constraint, there is no release in $[b - p_i, b)$, so no job released before b can be impacted by the shift. As a consequence, only the job released at b changes (due to the shift) its contribution to the minimum demand, which means that the minimum demand near a is increased by exactly the shift amount, $a - b$.

Consider now what happens near the end of the interval, where the overall minimum demand tends to be decreased. The minimum demand of jobs near $a + t$ is the amount of execution that cannot be postponed until after $a + t$. The latest that each job can be postponed is the interval of length e_i immediatly preceding its deadline. These intervals are non overlapping, since the job deadlines are all separated by p_i, and $e_i < p_i$. Due to this fact, the shift cannot decrease the minimum demand near the end of the interval more than the shift amount, $a - b$.

So, any decrease in minimum demand near $a + t$ is offset by the increase near a. As a further note, consider that in the particular case that the same job is influenced by both the increase near a and the decrease near $a + t$, the overall reasoning remains valid, and so even in this case shifting all the release times up by $a - b$ does not decrease the minimum demand of τ_i in $[a, a + t)$.

The last step is to observe that the minimum demand in $[a, a + t)$ by periodic releases starting at a is the same as the minimum demand in $[0, t)$ with periodic releases starting at zero.

□

A more detailed version of the proof above may be found in the technical report [6].

The following necessary condition for feasibility of a sporadic task set follows very directly from the above definitions and theorem.

Theorem 2 (Infeasibility test) *If a set τ of sporadic tasks is feasible on m processors for every release time assignment then $m\ell(\tau) \leq m$.*

proof: Suppose $m\ell(\tau) > m$. By the definition of $m\ell$, there is some time duration t for which $\sum_{i=1}^{n} md(\tau_i, t)/t > m$. By the definition of $m\ell(\tau)$ and the theorem above, given the canonical release sequence of each task, the total execution time in $[0, t)$ must exceed mt, which is a contradiction. □

By Theorem 2, the condition $m\ell(\tau) \leq m$ is necessary for m-processor feasibility of a sporadic task set τ. While this condition is not sufficient for feasibility in general it is sufficient as well as necessary for the case $m = 1$.

Corollary 1 (Feasibility test for one processor) *If $m\ell(\tau) \leq 1$ then τ is schedulable on a single processor.*

proof: The proof follows directly from the fact that $m\ell(\tau) \geq \delta_{sum}(\tau)$ (shown in Lemma 2 below) and the

well-known fact that τ is schedulable by EDF on one processor if $\delta_{sum}(\tau) \leq 1$ [8]. □

Example 3 The task set described by the table below illustrates why the above result does not generalize to multiple processors. For this task set, $m\ell(\tau) = 2$. The task set is clearly not feasible on $m = 2$ processors. The problem is that τ_3 needs to execute for two time units in the interval $[0, 3)$ and there are two time units of execution time available, but the only two units of execution time available run in parallel on two different processors.

i	p_i	e_i	d_i
1	2	1	1
2	2	1	1
3	3	2	3

□

4 How to compute maxmin-demand

In order to be able to use Theorem 2 as a test of infeasibility, one needs to compute the function $m\ell(\tau)$. The first step is to compute $md(\tau_i, t)$.

The value of $md(\tau_i, t)$ can be computed for any t as follows:

Theorem 3 *For any sporadic task τ_i and time duration t,*

$$md(\tau_i, t) = j_t e_i + \max(0, t - (j_t p_i + d_i - e_i)) \quad (1)$$

where

$$j_t \stackrel{def}{=} \max(0, \lfloor \frac{t - d_i}{p_i} \rfloor + 1)$$

proof: By Theorem 1, computing $md(\tau_i, t)$ is the same as computing $cd(\tau_i, t)$. Let j_t be the number of jobs of τ_i that must execute to completion entirely within $[0, t)$. It follows that $j_t = 0$ if-and-only-if $t < d_i$. For $j_t \geq 1$ the deadline of the j_tth job falls on or before t and the deadline of the next job falls after t, *i.e.*,

$$(j_t - 1)p_i + d_i \leq t < j_t p_i + d_i \quad (2)$$

$$\frac{t - d_i}{p_i} < j_t \leq \frac{t - d_i}{p_i} + 1 \quad (3)$$

Since j_t is an integer,

$$j_t = \lfloor \frac{t - d_i}{p_i} \rfloor + 1$$

Whether some portion of the execution of the j_{t+1}th job must complete in $[0, t)$ depends on whether $t - (j_t p_i + d_i - e_i) > 0$ (i.e., whether the j_{t+1}th job has "throwforward" on t[14]).

Case 1: If $t - (j_t p_i + d_i - e_i) \leq 0$ then the j_{t+1}th job can complete by the deadline without executing at all in the interval $[0, t)$.

Case 2: If $t - (j_t p_i + d_i - e_i) > 0$ then the j_{t+1}th job cannot complete by the deadline $j_t p_i + d_i$ unless it has already completed at least $t - (j_t p_i + d_i - e_i)$ execution by time t.

Taking the maximum of these two cases, and adding $j_t e_i$ for the execution times of the first j_t jobs, one obtains (1).

□

Note that the function $md(\tau_i, t)$ defined here is similar to DBF(τ_i, t). It differs only in being larger by the throwforward term, $\max(0, t - (jp_i + d_i - e_i))$. Therefore, prior techniques for computing rapid approximations to DBF(τ_i, t) [1, 12] can be modified to fit $md(\tau_i, t)$. In particular, the function $md(\tau_i, t)$ can be approximated within a tolerance of $u_i(p_i - e_i)$ for sufficiently large t, as follows.

Lemma 1 *For any sporadic task τ_i, if $t \geq d_i$ then*

$$u_i(t - d_i + e_i) \leq md(\tau_i, t) < u_i(t - d_i + p_i) \quad (4)$$

proof: Since $t \geq d_i$, by Theorem 3, $j_t = \lfloor \frac{t - d_i}{p_i} \rfloor + 1 \geq 1$ and $md(\tau_i, t)$ is the maximum of two functions, given by $j_t e_i$ and $j_t e_i + t - j_t p_i - d_i + e_i$, which coincide at all the points t such that $t = j_t p_i + d_i - e_i$ for some integer j_t.

The value of $md(\tau_i, t)$ is constant with respect to t and has the value $j_t e_i$ when

$$(j_t - 1)p_i + d_i \leq t < j_t p_i + d_i - e_i$$

It increases linearly with t and has the value $t - d_i - j_t(p_i - e_i) + e_i$ when

$$j_t p_i + d_i - e_i \leq t < j_t p_i + d_i$$

Therefore, $md(\tau_i, t)$ is bounded above by the linear interpolation of the points where $md(\tau_i, t)$ changes from increasing to constant and bounded below by the linear interpolation of the points where $md(\tau_i, t)$ changes from constant to increasing. The upper bound can be obtained by interpolating the values of $md(\tau_i, t)$ at the points $t = (j_t - 1)p_i + d_i$.

$$md(\tau_i, t) = j_t e_i = \frac{t - d_i + p_i}{p_i} e_i = u_i(t - d_i + p_i)$$

and the lower bound can be obtained by interpolating the values of $md(\tau_i, t)$ at the points $t = j_t p_i + d_i - e_i$.

$$md(\tau_i, t) = j_t e_i = \frac{t - d_i + e_i}{p_i} e_i = u_i(t - d_i + e_i)$$

□

Note that the upper bound for $\mathrm{DBF}(\tau_i, t)$ (see [12]) is the same as for $md(\tau_i, t)$, but the lower bound for $\mathrm{DBF}(\tau_i, t)$ is smaller by $u_i e_i$.

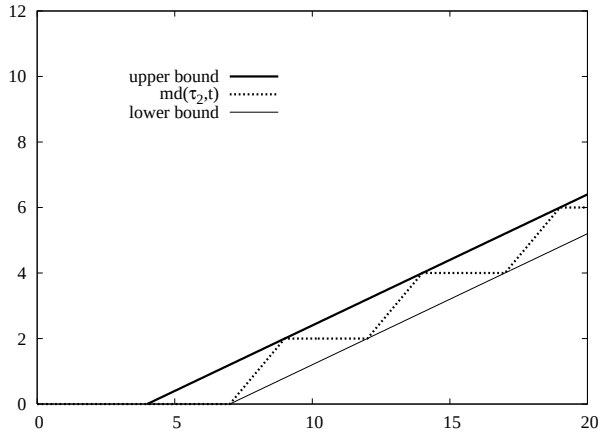

Figure 2. $md(\tau_2, t)$ with lower and upper bounds

Example 4 Consider the task system below.

i	p_i	e_i	d_i
1	7	2	6
2	5	2	9

The zig-zag dotted line in Figure 2 shows the function $md(\tau_i, t)$ for τ_2. The solid lines are the upper and lower bounds of (4). □

5 How to approximate maxmin load

Calculating $m\ell(\tau)$ requires finding the maximum of the function $\sum_{i=1}^{n} md(\tau_i, t)/t$ over an unbounded range of real numbers t. This can be done efficiently because one can show that there is a finite range of values of t at which the maximum of $\sum_{i=1}^{n} md(\tau_i, t)/t$ can occur.

Observe that $md(\tau_i, t)$ alternates between constant intervals and intervals of linear increase. The changes from increasing to constant occur at the points $jp_i + d_i$, for $j = 1, 2, \ldots$.

Example 5 Consider the task set given in Example 4. Figure 3 shows $md(\tau_1, t)$ (thin solid line), $md(\tau_2, t)$ (dotted line), and $md(\tau_1, t) + md(\tau_2, t)$ (heavier solid line). □

The function $\sum_{i=1}^{n} md(\tau_i, t)$ is made up of constant segments and linearly increasing segments. Each constant segment begins at one of the points $jp_i + d_i$, for $j = 1, 2, \ldots$ and $i = 1, 2, \ldots, n$. After the sum is divided by t, there are peaks at those same points, as shown in Figure 4.

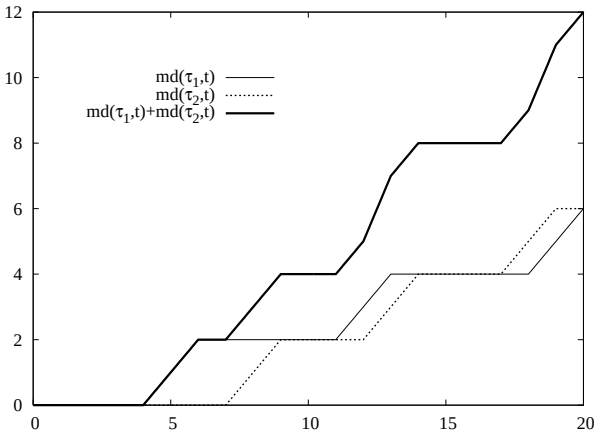

Figure 3. $md(\tau_1, t)$, $md(\tau_2, t)$, and their sum

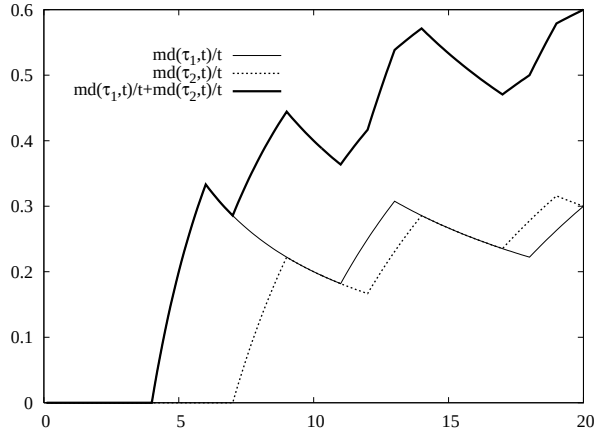

Figure 4. $md(\tau_1, t)/t$, $md(\tau_2, t)/t$, and their sum

Lemma 1 provides the following bounds for $md(\tau_i, t)/t$, for $t \geq d_i$, within a tolerance of $u_i(p_i - e_i)/t$:

$$u_i(1 + \frac{e_i - d_i}{t}) \leq \frac{md(\tau_i, t)}{t} \leq u_i(1 + \frac{p_i - d_i}{t}) \quad (5)$$

Since the value of the upper bound expression (on the right above) tends to u_i for sufficiently large t, the global maximum is at one of the early peaks. The question is how far to look.

By computing $md(\tau_i, t)$ exactly for small values of t and using the above linear approximations for large values of t, the search space for the maximum can be limited to a size that is polynomial in the length of the task set. The technique is analogous to that used to compute δ_{sum} in [12], which is based on earlier work by Albers and Slomka [1] for uniprocessor feasibility analysis.

For an approximation to $md(\tau_i, t)/t$ from below, the

function $g_\epsilon(\tau_i, t)$ be defined as follows

$$g_\epsilon(\tau_i, t) \stackrel{\text{def}}{=} \begin{cases} md(\tau_i, t)/t, & \text{if } t < \max(d_i, n\frac{u_i(p_i-e_i)}{\epsilon}) \\ u_i(1 + \frac{e_i-d_i}{t}) & \text{otherwise} \end{cases}$$

Note that the approximation can be done from either above or below, depending on whether one is interested in proving infeasibility (for m processors) or feasibility (for a single processor). However, the latter is not very interesting, since the δ_{sum} test is already sufficient for the single-processor case. Therefore, and because of space limitations, we only describe in full the approximation from below. For the approximation to $md(\tau_i, t)$ from above one just needs to replace $u_i(1 + \frac{e_i-d_i}{t})$ by $u_i(1 + \frac{p_i-d_i}{t})$.

If $t < \max(d_i, n\frac{u_i(p_i-e_i)}{\epsilon})$, $md(\tau_i, t)/t - g_\epsilon(\tau_i, t) = 0 \leq \frac{\epsilon}{n}$. Otherwise, from (5) it follows that, for every $t \geq 0$,

$$\begin{aligned} 0 &\leq md(\tau_i, t)/t - g_\epsilon(\tau_i, t) \\ &\leq \frac{u_i(p_i - e_i)}{t} \leq \frac{\epsilon}{n} \end{aligned}$$

Summing, we obtain

$$0 \leq \sum_{i=1}^{n} md(\tau_i, t)/t - \sum_{i=1}^{n} g_\epsilon(\tau_i, t) \leq \epsilon \qquad (6)$$

Let

$$\hat{g}_\epsilon(\tau) \stackrel{\text{def}}{=} \operatorname*{lub}_{t \geq 0} \sum_{i=1}^{n} g_\epsilon(\tau_i, t)$$

It follows from (6) that

$$\hat{g}_\epsilon(\tau) \leq m\ell(\tau) \leq \hat{g}_\epsilon(\tau) + \epsilon$$

Observe that $g_\epsilon(\tau_i, t)$ is monotonic with respect to t except at the points where $\lfloor \frac{t-d_i}{p_i} \rfloor$ makes a jump, *i.e.*, only at values $t = kp_i + d_i$ such that $t < n\frac{u_i(p_i-e_i)}{\epsilon}$, for $k = 0, 1, 2, \ldots$ and $i = 1, \ldots, n$. Therefore, local maxima of $\sum_{i=1}^{n} g_\epsilon(\tau_i, t)$ can occur only at such points. The set $\mathcal{S}(\tau, \epsilon)$ of such points can be described as follows:

$$\{kp_i + d_i \mid 0 < k < \frac{nu_i(p_i - e_i)}{\epsilon p_i} - \frac{d_i}{p_i}, 1 \leq i \leq n\} \quad (7)$$

Therefore,

$$\hat{g}_\epsilon(\tau) = \max_{t \in \mathcal{S}(\tau,\epsilon)} \sum_{i=1}^{n} g_\epsilon(\tau_i, t)$$

and since $g_\epsilon(\tau_i, t) \leq md(\tau_i, t)/t$,

$$\hat{g}_\epsilon(\tau) \leq \max_{t \in \mathcal{S}(\tau,\epsilon)} \sum_{i=1}^{n} md(\tau_i, t)/t \leq m\ell(\tau)$$

The value $\hat{g}_\epsilon(\tau)$ can be computed by evaluating $\sum_{i=1}^{n} g_\epsilon(\tau_i, t)$ at each of the points in $\mathcal{S}(\tau, \epsilon)$. Given any fixed tolerance $\epsilon > 0$, and assuming that each $u_i \leq 1$, (7) provides the following bound on the number of points $t \in \mathcal{S}(\tau, \epsilon)$.

$$\begin{aligned} \sum_{i=1}^{n} \frac{nu_i(p_i - e_i)}{\epsilon p_i} - \frac{d_i}{p_i} &= \sum_{i=1}^{n} \frac{nu_i}{\epsilon} - \sum_{i=1}^{n} (\frac{nu_i^2}{\epsilon} + \frac{d_i}{p_i}) \\ &\leq \frac{n}{\epsilon} u_{sum} \end{aligned}$$

A straightforward implementation of $\sum_{i=1}^{n} g_\epsilon(\tau_i, t)$ has $\mathcal{O}(n)$ complexity, so the total complexity is $\mathcal{O}(n^2 u_{sum}/\epsilon)$. Note that since $u_{sum} \leq n$, $\mathcal{O}(n^2 u_{sum}/\epsilon) \in \mathcal{O}(n^3/\epsilon)$. That is, the runtime of the algorithm APPROXIMATE-$m\ell$ is polynomial in the number n of tasks in τ and $1/\epsilon$, and is independent of the task parameters.

Note further that in an application the number of processors m is known and one can assume that $u_{sum} \leq m$, so $\mathcal{O}(n^2 u_{sum}/\epsilon) \in \mathcal{O}(n^2 m/\epsilon)$.

Additional heuristics, based on the following two lemmas, can often reduce the actual running time below the $\mathcal{O}(n^2 m/\epsilon)$ worst-case bound.

Lemma 2 *For any sporadic task set τ,*

$$u_{sum}(\tau) \leq \delta_{sum}(\tau) \leq m\ell(\tau) \leq \lambda_{sum}(\tau)$$

proof: It was shown in [12] that $u_{sum}(\tau) \leq \delta_{sum}(\tau) \leq \lambda_{sum}(\tau)$. The function $m\ell(\tau)$ defined here is similar to $\delta_{sum}(\tau)$, but differs by using $md(\tau_i, t) = \text{DBF}(\tau_i, t) + \max(0, t - (jp_i + d_i - e_i))$ instead of $\text{DBF}(\tau_i, t)$. Therefore, it is clear that $\delta_{sum}(\tau) \leq m\ell(\tau)$, and so only the upper bound needs to be proved.

If $t < d_i$ then, from the definition of $md(\tau_i, t)$,

$$md(\tau_i, t) = \max(0, t - d_i + e_i)$$

This is non-decreasing with respect to t, and so the maximum value of $md(\tau_i, t)/t$ for $t \leq d_i$ is $e_i/d_i \leq \lambda_i$.

If $t \geq d_i$, then, from (5),

$$\frac{md(\tau_i, t)}{t} \leq u_i(1 + \frac{p_i - d_i}{t})$$

If $p_i \geq d_i$ the term $\frac{p_i-d_i}{t}$ is non-increasing with respect to t, and so for $t \geq d_i$,

$$\frac{md(\tau_i, t)}{t} \leq u_i(1 + \frac{p_i - d_i}{d_i}) = \frac{e_i}{d_i} = \lambda_i$$

If $p_i < d_i$ the term $\frac{p_i-d_i}{t}$ is increasing with respect to t, so the least upper bound is the limit for large t.

$$\frac{md(\tau_i, t)}{t} \leq \lim_{t \to \infty} \frac{md(\tau_i, t)}{t} = u_i = \lambda_i$$

It follows that the least upper bound of $\sum_{i=1}^{n} md(\tau_i, t)/t$ is bounded by λ_{sum}. □

Another condition for early termination of the search for $m\ell(\tau)$ is based on the fact that $md(\tau_i, t)/t$ tends to u_i for sufficiently large t (cfr. (5)), and so the difference between $u_{sum}(\tau)$ and $\sum_{i=1}^{n} md(\tau_i, t)/t$ gives an upper bound on t. This is expressed in Lemma 3 below.

Lemma 3 *If*

$$u_{sum}(\tau) + \gamma \leq \sum_{i=1}^{n} \frac{md(\tau_i, t)}{t} \tag{8}$$

for some $\gamma > 0$, then

$$t \leq \sum_{i=1}^{n} \frac{u_i \max(0, p_i - d_i)}{\gamma} \tag{9}$$

proof: It follows from (5) above that if $t \geq d_i$ then

$$\frac{md(\tau_i, t)}{t} \leq u_i(1 + \frac{p_i - d_i}{t}) \leq u_i(1 + \frac{\max(0, p_i - d_i)}{t})$$

There are two other cases:

1. If $t < d_i - e_i$ then

$$\frac{md(\tau_i, t)}{t} = 0 \leq u_i(1 + \frac{\max(0, p_i - d_i)}{t})$$

2. If $d_i - e_i \leq t < d_i$ then, since $u_i \leq 1$, $u_i(t - d_i) \geq t_i - d_i$, and so

$$\begin{aligned} \frac{md(\tau_i, t)}{t} &= \frac{t - d_i + e_i}{t} \\ &\leq \frac{u_i(t - d_i) + e_i}{t} = \frac{u_i(t - d_i) + u_i p_i}{t} \\ &= u_i(1 + \frac{p_i - d_i}{t}) \\ &\leq u_i(1 + \frac{\max(0, p_i - d_i)}{t}) \end{aligned}$$

Therefore,

$$\sum_{i=1}^{n} \frac{md(\tau_i, t)}{t} \leq u_{sum}(\tau) + \frac{\sum_{i=1}^{n} u_i \max(0, p_i - d_i)}{t} \tag{10}$$

Composing (8) and (10) yields

$$u_{sum}(\tau) + \gamma \leq u_{sum}(\tau) + \frac{\sum_{i=1}^{n} u_i \max(0, p_i - d_i)}{t}$$

from which (9) follows. □

Pseudo-code for the algorithm APPROXIMATE-$m\ell$ is given in Figure 5. The initial value given to $m\ell$ in line 1 is based on the relation $m\ell(\tau) \geq u_{sum}(\tau)$ from Lemma 2. The algorithm incorporates two heuristics that permit the computation of $m\ell(\tau)$ to terminate without looking at all the elements of $\mathcal{S}(\tau, \epsilon)$. One of these is the relationship $m\ell(\tau) \leq \lambda_{sum}(\tau)$, from Lemma 2, which is applied in line 8 to return immediately if the maximum of $md(\tau_i, t)$ over the range of values examined exceeds $\lambda_{sum}(\tau) - \epsilon$. The other heuristic is applied in line 8 to return immediately if the range of values examined so far exceeds the value of *limit* computed at line 7, based on Lemma 3.

```
APPROXIMATE-mℓ(τ, ε)
 1  mℓ ← u_sum(τ)
 2  d_max ← max_{i=1}^{n} d_i
 3  limit ← max(S(τ, ε))
 4  for each t ∈ S(τ, ε), in increasing order do
 5     mℓ ← max(mℓ, Σ_{i=1}^{n} md(τ_i, t)/t)
 6     if t > d_max then
 7        limit ← min(limit,
              Σ_{i=1}^{n} u_i max(0, p_i − d_i)/(mℓ − u_sum))
 8     if t ≥ limit or mℓ(τ) > λ_sum(τ) − ε then
 9        return mℓ
10     return mℓ;
```

Figure 5. Pseudo-code for approximate computation of $m\ell(\tau)$ from below.

Note again that the above computation approximates $m\ell(\tau)$ from below, for use in proving a task set is not feasible on m processors. Similar reasoning can be used to approximate $m\ell(\tau)$ from above, with similar runtime complexity. The technique of approximation from above is explained in more detail, in the context of computing $\delta_{sum}(\tau)$, by [12].

6 Empirical performance

It is clear that the new load-bound function $m\ell(\tau)$ is an improvement over $\delta_{sum}(\tau)$ for screening out infeasible task sets, since we have shown in Lemma 2 that $\delta_{sum}(\tau) \leq m\ell(\tau)$ and we have shown by Example 1 that in some cases $\delta_{sum}(\tau) \leq m$ while $m\ell(\tau) > m$.

To get a sense of how often the difference between these two functions is enough to matter, we ran experiments on a large number of pseudo-randomly chosen sets of tasks. The experiments compared the number of task sets eliminated by the new load-bound function against the number eliminated by the old load-bound function δ_{sum}. Because of the page limit, the results of only a few such experiments are reported here.

Figures 6 and 7 show the result of experiments on

1,000,000 pseudo-randomly generated task sets with uniformly distributed utilizations and uniformly distributed constrained deadlines, with total utilizations limited to $u_{sum} \leq m$ for $m = 2$ and 8. Each graph is a histogram in which the X axis corresponds to values of u_{sum} and the Y axis corresponds to the number of task sets with u_{sum} in the range $[X, X + 0.01)$ that satisfy a given criterion.

- For the top line, which is unadorned, there is no additional criterion. That is, the Y value is simply the number of task sets with $X \leq u_{sum} < X+0.01$. The experiments did not include any task sets with $u_{sum} > m$, since they are clearly not feasible.
- For the second line, decorated with large boxes, the criterion is $\delta_{sum} \leq m$. The Y value is the number of task sets that *might* be feasible according to this criterion. The space between this line and the first indicates how many task sets are detected as infeasible.
- The third line, decorated with small solid squares, corresponds to the criterion $m\ell(\tau) \leq m$. The Y value is the number of task sets that *might* be feasible according to this criterion. The region between this line and the one above it indicates the improvement in recognition of infeasible task sets due to using $m\ell$, as compared to δ_{sum}. It can be seen that the condition $m\ell(\tau) \leq m$ identifies significantly more infeasible task sets than the condition $\delta_{sum}(\tau) \leq m$, especially for systems with a large number of tasks and processors.
- The bottom line, decorated with small circles, corresponds to the criterion $\lambda_{sum} \leq m$. This indicates how many task sets at each utilization level are *definitely* feasible according to this criterion. The region between this line and the one above it indicates the number of task sets whose feasibility remains indeterminate using the simple tests described in this paper.

Figure 8 shows the number of iterations taken by algorithm Approximate-$m\ell$ to compute $m\ell(\tau)$ and the number of iterations taken to compute δ_{sum}. For fairness in comparison, δ_{sum} was approximated from below, using an algorithm similar to Approximate-$m\ell$. Note that this is distinct from the algorithms for approximating δ_{sum} reported in [12] because the primary subject of interest was feasibility, and we are approximating from below because the current subject of interest is infeasibility.

Observe that the computation of $m\ell(\tau)$ converges faster than the computation of $\delta_{sum}(\tau)$. Since the algorithms are virtually the same, the reason for the faster convergence is that the linear function used to estimate the error for $md(\tau_i, t)$ is larger than that for $\mathrm{DBF}(\tau_i, t)$.

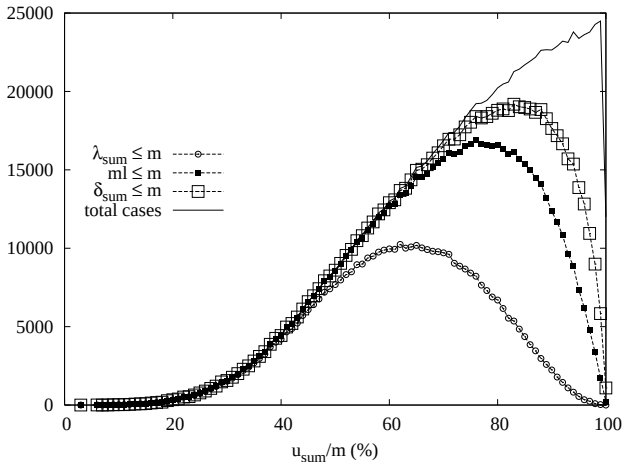

Figure 6. Histograms for $u_{sum} \leq m$, $m = 2$

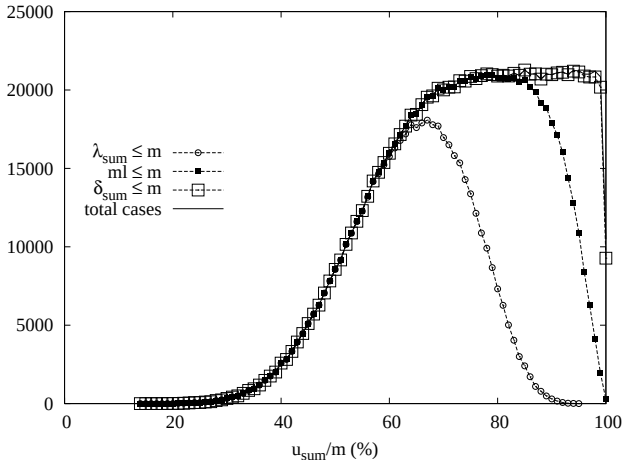

Figure 7. Histograms for $u_{sum} \leq m$, $m = 8$

7 Conclusion

The new load bound function $m\ell(\tau)$ provides a more accurate and more rapidly computable method than has been known before for detecting that a sporadic task set is infeasible. The value of this result is in narrowing the area of uncertainty, between the task sets that can be verified as feasible (*e.g.*, by showing they are schedulable according to some specific algorithms such as EDF, Pfair, *etc.*) and those that are not schedulable.

Although the method for computing $m\ell(\tau)$ is only approximate, an approximation is adequate for all practical purposes. In any real application of a schedu-

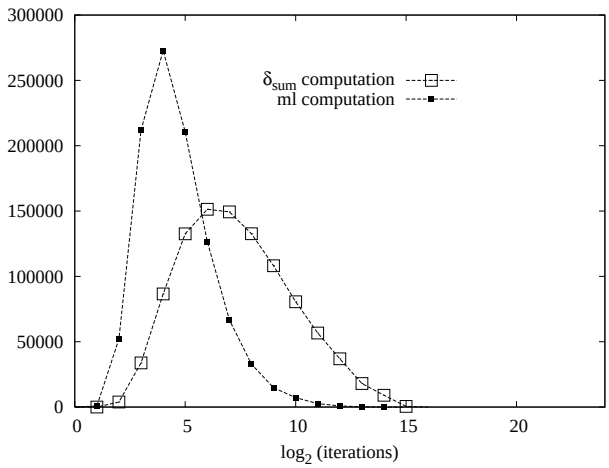

Figure 8. Iterations to compute load-bound for $u_{sum} \leq 4$

lability test one needs to allow a margin of safety, to account for such factors as errors in predicting the worst-case task execution times, inaccuracy of the timer used to schedule the tasks, and preemption delays due to critical sections.

Incidental to the main objective of this research, it turned out that $m\ell(\tau) \leq 1$ is also a necessary and sufficient test for feasibility and EDF schedulability of sporadic task sets on multiprocessor systems. This confirms that $m\ell(\tau)$ is a consistent and natural refinement of δ_{sum}. Moreover, since the approximate computation of $m\ell(\tau)$ (from below) converges faster than δ_{sum}, it would seem to have a practical advantage.

The problem of finding an efficient feasibility test that is both necessary and sufficient for multiprocessor systems, or to show that this problem is NP-hard, appears to remain open.

Acknowledgments

The authors are thankful to Sanjoy Baruah and Nathan Fisher for their collaboration with one author on the computation of δ_{sum}, reported in [12], which laid the essential groundwork for the developments reported in this paper and stimulated further thought about this interesting problem. We are also thankful to the anonymous RTSS reviewers for their constructive suggestions, which helped us to improve the paper.

References

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

[2] B. Andersson, S. Baruah, and J. Jonsson. Static-priority scheduling on multiprocessors. In *Proc. 22nd IEEE Real-Time Systems Symposium*, pages 193–202, London, UK, Dec. 2001.

[3] T. P. Baker. An analysis of EDF scheduling on a multiprocessor. *IEEE Trans. on Parallel and Distributed Systems*, 15(8):760–768, Aug. 2005.

[4] T. P. Baker. An analysis of fixed-priority scheduling on a multiprocessor. *Real Time Systems*, 2005.

[5] T. P. Baker. Brute-force determination of mulitprocessor schedulability for sets of sporadic hard-deadline tasks. Technical Report TR-061001, Florida State University Department of Computer Science, Tallahassee, FL, Nov. 2006.

[6] T. P. Baker and M. Cirinei. A necessary and sometimes sufficient condition for the feasibility of sets of sporadic hard-deadline tasks. Technical Report TR-060501, Florida State University Department of Computer Science, Tallahassee, FL, May 2006.

[7] S. Baruah and N. Fisher. Partitioned multiprocessor scheduling of sporadic task systems. In *Proc. of the 26th IEEE Real-Time Systems Symposium*, Miami, Florida, Dec. 2005. IEEE Computer Society Press.

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

[9] S. K. Baruah, A. K. Mok, and L. E. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. *Proc. 11th IEE Real-Time Systems Symposium*, pages 182–190, 1990.

[10] M. Bertogna, M. Cirinei, and G. Lipari. Improved schedulability analysis of EDF on multiprocessor platforms. In *Proc. 17th Euromicro Conference on Real-Time Systems*, pages 209–218, Palma de Mallorca, Spain, July 2005.

[11] M. Bertogna, M. Cirinei, and G. Lipari. New schedulability tests for real-time task sets scheduled by deadline monotonic on multiprocessors. In *Proc. of the 9th International Conf. on Principles of Distributed Systems*, Pisa, Italy, Dec. 2005.

[12] N. Fisher, T. P. Baker, and S. K. Baruah. Algorithms for determining the demand-based load of a sporadic task system. In *12th IEEE International Conf. on Embedded and Real-Time Computing Systems and Applications*, Sydney, Australia, Dec. 2006. (to appear).

[13] J. Goossens, S. Funk, and S. Baruah. Priority-driven scheduling of periodic task systems on multiprocessors. *Real Time Systems*, 25(2–3):187–205, Sept. 2003.

[14] H. H. Johnson and M. S. Maddison. Deadline scheduling for a real-time multiprocessor. In *Proc. Eurocomp Conference*, pages 139–153, 1974.

[15] A. Srinivasan and S. Baruah. Deadline-based scheduling of periodic task systems on multiprocessors. *Information Processing Letters*, 84:93–98, 2002.