Algorithms for Determining the Demand-Based Load of a Sporadic Task System*

Nathan Fisher † Theodore P. Baker ‡ Sanjoy Baruah †

Abstract

The load *parameter of a sporadic task system is defined to be the largest possible cumulative execution requirement that can be generated by jobs of the task system over any time interval, normalized by the length of the interval. This parameter is known to play a very important role in the uniprocessor feasibility analysis of sporadic task systems. In this paper, it is shown that the load of a sporadic task system may be used as an accurate indicator of its feasibility upon preemptive multiprocessors as well. Exact algorithms, and approximate ones that can be guaranteed to be accurate to within an arbitrary additive error > 0, for computing a task system's load are presented and proven correct. The performance of these algorithms is evaluated by simulation over randomly generated task systems.*

1 Introduction

In the *sporadic task model* [7], 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 separation* p_i. (It is assumed that $e_i \leq d_i$ and $e_i \leq p_i$.) The *utilization* of task τ_i is denoted by $u_i \stackrel{\text{def}}{=} e_i/p_i$. A sporadic task system τ is collection of sporadic tasks $\{\tau_1, \tau_2, \ldots, \tau_n\}$.

An important scheduling theory concept is determining the *feasibility* of a task system on a specified platform. A sporadic task system is said to be *feasible* upon a specified platform if it is possible to schedule the system on the platform such that all jobs of all tasks will meet all deadlines, under all permissible (also called *legal*) combinations of job-arrival sequences by the different tasks comprising the system. Conceptually, one would like to be able to define a measure of "computational demand" for task systems, and a measure of "computational capacity" for computational platforms, such that a task system is feasible on given platform if and only if the computational demand of the task system does not exceed the computational capacity of the platform. In particular, suppose a multiprocessor system comprised of m unit-capacity processors is defined to have a computational capacity equal to m. A necessary and sufficient condition for τ to be feasible on an m processor system would be that the computational demand of τ does not exceed m.

This model of schedulability analysis is known to work for *implicit-deadline* sporadic task systems, where each task has its relative deadline parameter equal to its minimum inter-arrival separation parameter (i.e. $d_i = p_i$). The concept of computational demand there can be captured by the total system utilization, $u_{sum}(\tau) \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau} u_i$. The condition $u_{sum}(\tau) \leq m$ is necessary and sufficient for τ to be feasible on a preemptive m processor system.

For general deadlines (where it is possible that d_i is *not* equal to p_i), $u_{sum}(\tau)$ is still a lower bound on computational demand, but not an upper bound. That is, while satisfaction of the utilization bound is a necessary condition for feasibility, it is not sufficient. In fact, it can be shown that there exist infeasible task systems with arbitrarily small utilization. This is illustrated in the following example:

Example 1 Consider the following sporadic task system consisting of three tasks to be scheduled on a multiprocessor system comprised of two unit-capacity processors.

$$\tau = \{\tau_1 = (1, 1, r), \tau_2 = (1, 1, r), \tau_3 = (1, 1, r)\}$$

where $r \geq 2$. Observe that $u_{sum}(\tau) = 3/r \leq 3/2 \leq 2$, and $\lim_{r \to \infty} u_{sum}(\tau) = 0$; however, if each task of τ releases a job at time-instant zero, each job must complete one unit of execution by time-instant one. There is no possible way to schedule τ over the interval $[0, 1)$; therefore, τ is infeasible on two processors. ■

*This research has been partially supported in part by the National Science Foundation (Grant Nos. ITR-0082866, CCR-0204312, CCR-0309825, and CHR-EHS 0509131).

†The University of North Carolina at Chapel Hill, Department of Computer Science, CB-3175, Chapel Hill, NC 27599-3175 USA, {fishern,baruah}@cs.unc.edu

‡Florida State University, Department of Computer Science, Tallahassee, FL 32306-4530, USA, baker@cs.fsu.edu

An upper bound on computational demand for sporadic task systems with arbitrary relative deadlines is $\lambda_{sum}(\tau) \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau} \lambda_i$, where $\lambda_i = \frac{e_i}{\min(d_i, p_i)}$. (The quantity λ_i is referred to as the *density* of τ_i.) It was shown in [6] that $\lambda_{sum}(\tau) \leq 1$ is a sufficient condition for the preemptive uniprocessor feasibility of sporadic task systems — this result is easily extended to show that $\lambda_{sum}(\tau) \leq m$ is a sufficient condition for feasibility upon a multiprocessor platform comprised of m unit-capacity processors. However, this condition is not necessary for feasibility: consider the following example.

Example 2 Given a sporadic task system τ consisting of n tasks to be scheduled on a single preemptive processor. The i'th task has execution-requirement 1, relative deadline i, and inter-arrival separation n. It may be verified (see, e.g., [4]) that this system is feasible. Its density $\lambda_{sum}(\tau) = \sum_{i=1}^{n}(1/i)$, which grows without bound with increasing n. This example illustrates that there are sporadic task systems τ of arbitrarily high density $\lambda_{sum}(\tau)$, which are feasible. ■

In summary, with respect to the preemptive scheduling of a sporadic task system τ on a multiprocessor platform comprised of m unit-capacity processors:

1. $u_{sum}(\tau)$ is a lower bound on demand: $u_{sum}(\tau) \leq m$ is a necessary condition for feasibility;
2. $\lambda_{sum}(\tau)$ is an upper bound on demand, $\lambda_{sum}(\tau) \leq m$ is a sufficient condition for feasibility;
3. when $d_i = p_i$, the two coincide, giving us a necessary and sufficient condition;
4. when $d_i \neq p_i$ the two bounds may leave a gap, where there is uncertainty whether a task set is feasible; Examples 1 and 2 show the gap may be large.

In this report, we consider an improved lower bound on computational demand for general sporadic task systems, which we call the *load-bound function* and denote by $\delta_{sum}(\tau)$. We will show that this $\delta_{sum}(\tau)$ is an improvement over $u_{sum}(\tau)$ and $\lambda_{sum}(\tau)$, and that it can be computed effectively with enough accuracy to be useful in practice.

The conceptual relationship of $\delta_{sum}(\tau)$ to $u_{sum}(\tau)$ and $\lambda_{sum}(\tau)$ is illustrated in Figure 1. It can be seen that there is a region of uncertainty between the lower and upper bounds, and that precision in determining which task sets are feasible and which are not is improved by finding upper bounds that are smaller and/or lower bounds that are larger. The importance of $\delta_{sum}(\tau)$ is that it reduces the region of uncertainty significantly, as compared to $u_{sum}(\tau)$.

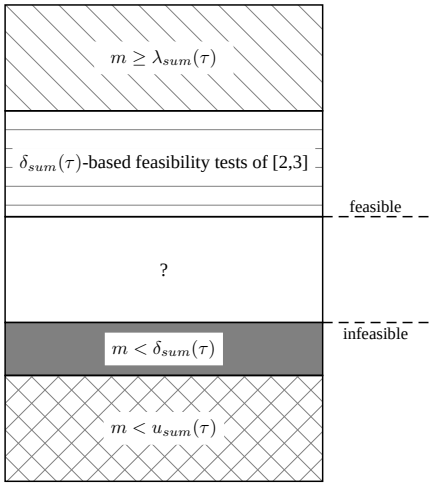

Figure 1. Relationship of bounds on computational load

In addition, recent work [2, 3] has derived sufficient conditions for feasibility based on $\delta_{sum}(\tau)$. The fact that $\delta_{sum}(\tau)$ can be used (independently) as a necessary or sufficient condition for feasibility makes $\delta_{sum}(\tau)$ a useful and reasonable metric for task system "computational demand".

To the best of our knowledge, there is no prior published algorithm for computing $\delta_{sum}(\tau)$ for an arbitrary sporadic task system. We present an exact algorithm for computing $\delta_{sum}(\tau)$ that involves simulating the scheduling of τ to its hyperperiod (the least-common-multiple of the task systems periods, $LCM_{i=1}^{n} p_i$). We also present two other algorithms which approximate $\delta_{sum}(\tau)$ within an arbitrary threshold $\epsilon > 0$ of its exact value. The first approximate algorithm runs in time pseudo-polynomial in the representation of the task system; the second algorithm, in time polynomial in the representation of the task system. We have observed that both approximation algorithms achieve a significant reduction in computation time even for $\epsilon = 0.001$.

The remainder of this paper is organized as follows. We formally define the load-bound function $\delta_{sum}(\tau)$ and prove useful properties of $\delta_{sum}(\tau)$ in Section 2. Previously derived multiprocessor feasibility and schedulability conditions are discussed in Section 3. We then derive a simple method for exactly determining $\delta_{sum}(\tau)$ in Section 4. We describe the more practical approximations of load in Section 5. Finally, we empirically evaluate our algorithms on a collection of pseudo-randomly generated task systems in Section 6.

2 Load-Bound Function

For any sporadic task τ_i and any real number $t \geq 0$, the *demand bound function* $\mathrm{DBF}(\tau_i, t)$ is defined to be the largest cumulative execution requirement of all jobs that can be generated by τ_i to have both their arrival times and their deadlines within a contiguous interval of length t. It has been shown [4] that the cumulative execution requirement of jobs of τ_i over an interval $[t_o, t_o + t)$ is maximized if one job arrives at the start of the interval – i.e., at time-instant t_o – and subsequent jobs arrive as rapidly as permitted — i.e., at instants $t_o + p_i$, $t_o + 2p_i$, $t_o + 3p_i, \ldots$:

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

Let $f(\tau_i, t)$ be defined to be $\mathrm{DBF}(\tau_i, t)$ normalized by the interval length: $f(\tau_i, t) \stackrel{\text{def}}{=} \dfrac{\mathrm{DBF}(\tau_i, t)}{t}$.

Given a sporadic task system $\tau = \{\tau_1, \ldots, \tau_n\}$, the load-bound function is defined by $\delta_{sum}(\tau) \stackrel{\text{def}}{=} \max_{t>0} f(\tau, t)$ where $f(\tau, t) \stackrel{\text{def}}{=} \sum_{i=1}^{n} f(\tau_i, t)$.

Figure 2 illustrates an example of $f(\tau_i, t)$ for two different tasks. The load-bound function, $\delta_{sum}(\tau)$ describes the maximum value of $f(\tau, t)$ over all positive values of t. Since the values of t are real and unbounded, the notation $\max_{t>0}$ here denotes the least upper bound of $f(\tau, t)$.

In [4], it was shown that $\delta_{sum}(\tau) \leq 1$ is a necessary and sufficient condition for sporadic task system τ to be feasible upon a preemptive unit-capacity processor. Upon *multi*processors, it has been shown previously that $\delta_{sum}(\tau) \leq m$ is a necessary condition for the feasibility of task τ on a platform with m unit-capacity processors [2]. We will next show that $\delta_{sum}(\tau)$ is potentially superior to $u_{sum}(\tau)$ as a lower bound on computational load, as it falls between $u_{sum}(\tau)$ and $\lambda_{sum}(\tau)$.

The next lemma shows that $u_{sum}(\tau)$ is a lower bound on the load-bound function:

Lemma 1 $\delta_{sum}(\tau) \geq u_{sum}(\tau)$

Proof: Observe that for each task $\tau_i \in \tau$, $\lim_{t\to\infty} f(\tau_i, t) = u_i$.

More precisely, for a given i and t, let $0 \leq r < p_i$ be the value such that $\lfloor \frac{t-d_i}{p_i} \rfloor = \frac{t-d_i-r}{p_i}$. It follows that

$$\begin{aligned}\frac{(\lfloor \frac{t-d_i}{p_i} \rfloor + 1)e_i}{t} &= \frac{(\frac{t-d_i-r}{p_i} + 1)e_i}{t} \\ &= \frac{(t + p_i - d_i - r)e_i}{tp_i} \\ &= u_i + u_i \frac{p_i - d_i - r}{t}\end{aligned}$$

Since $-d_i < p_i - d_i - r < p_i - d_i$, the absolute value of the fraction on the right above is decreasing with respect to t, and so the limit of the entire expression is u_i. It follows that $\lim_{t\to\infty} f(\tau, t) = u_{sum}(\tau)$. The lemma immediately follows from this limit. ■

The following lemma shows that $\lambda_{sum}(\tau)$ is an upper bound on the load-bound function:

Lemma 2 $\delta_{sum}(\tau) \leq \lambda_{sum}(\tau)$

Proof:

$$f(\tau, t) = \sum_{i:d_i<t} \frac{\lfloor \frac{t+p_i-d_i}{p_i} \rfloor e_i}{t} \leq \sum_{i:d_i<t} u_i(1 + \frac{p_i - d_i}{t})$$

If $p_i \geq d_i$ the term $\frac{p_i-d_i}{t}$ is non-increasing with respect to t, and since $d_i < t$, $\frac{p_i-d_i}{t} \leq \frac{p_i-d_i}{d_i} = \frac{p_i}{d_i} - 1$.

Otherwise, the term $\frac{p_i-d_i}{t}$ is increasing with respect to t, and in the limit $\frac{p_i-d_i}{t} = 0$.

Therefore,

$$\begin{aligned}f(\tau, t) \leq \sum_{i:d_i<t} u_i(1 + \max(0, \frac{p_i}{d_i} - 1)) &= \sum_{i:d_i<t} \lambda_i \\ &\leq \sum_{i=1}^{n} \lambda_i = \lambda_{sum}(\tau)\end{aligned}$$

■

3 Previously Known Load-based Schedulability Conditions

In this section, we review some prior work on multiprocessor feasibility and schedulability conditions based on $\delta_{sum}(\tau)$ and $\delta_{max}(\tau) \stackrel{\text{def}}{=} \max_{\tau_i \in \tau} e_i/d_i$. Recent work on the multiprocessor schedulability of general sporadic task systems has produced several conditions which rely upon $\delta_{sum}(\tau)$'s value [2, 3, 5]. In the area of multiprocessor partitioned scheduling, we will state one sufficient condition for schedulability:

Theorem 1 (from [2]) *For a sporadic task system τ where for all $\tau_i \in \tau$, $d_i \leq p_i$, and an m-processor (identical) platform, τ is partitionable upon m processors using the first-fit scheduling heuristic (tasks ordered by non-decreasing d_i) if*

$$\delta_{sum}(\tau) \leq \frac{m(1 - \delta_{max}(\tau)) + \delta_{max}(\tau)}{2}$$

■

A parameter analogous to load can also be defined for collections of independent aperiodic jobs. Baruah and Fisher [3] define demand-based load for aperiodic

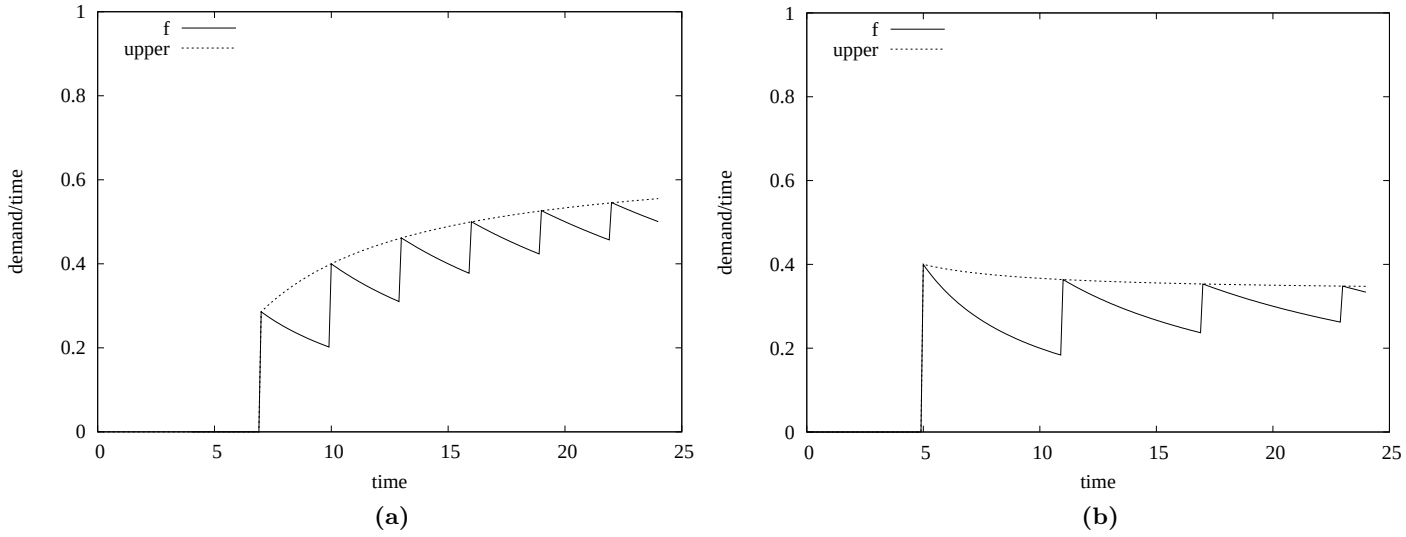

Figure 2. The zigzag solid lines in both graphs represent values $f(\tau_i, t)$ with respect to t where in (a) $\tau_i \stackrel{\text{def}}{=} (e_i, d_i, p_i) = (2, 7, 3)$, and in (b) $\tau_i \stackrel{\text{def}}{=} (2, 5, 6)$. The dotted lines represent approximations to $f(\tau_i, t)$ with $k_i = 0$. (Approximations are described in 5.2.)

jobs and derive non-trivial feasibility conditions for the collection when scheduled upon a multiprocessor platform. It is shown for any collection of independent jobs satisfying the load-based feasibility condition that there exists a feasible schedule upon an identical m-processor platform.

Prior work, however, left open the question of how to efficiently compute the $\delta_{sum}(\tau)$ parameter; thereby making it difficult and time-consuming to empirically evaluate the partitioning algorithm with respect to $\delta_{sum}(\tau)$. Deriving feasibility conditions based on $\delta_{sum}(\tau)$ is the subject of active research; for such conditions to be practical they must be efficiently computable. The remainder of the paper focuses on answering the question of how to efficiently compute $\delta_{sum}(\tau)$ for sporadic task systems.

4 An Exact Algorithm for Calculating Load

To calculate $\delta_{sum}(\tau)$, we must limit the number of values of t for which we evaluate $f(\tau, t)$ to a finite number. It may seem that $f(\tau, t)$ needs to be checked at an infinite number of t values. However, the following two observations are useful in showing that only a finite number of values need to be checked:

1. **The maximum value of $f(\tau, t)$ only occurs at "step" points** (Lemma 3). Therefore, the set of potential test points is countable.
2. **$f(\tau, t)$ is maximized prior to τ's hyperperiod** (Lemma 4). Therefore, the maximum test point has a bounded value.

The following lemma formally restates and proves the first observation:

Lemma 3

$$\max_{t>0} f(\tau, t) = \max\{f(\tau, jp_i + d_i) \mid i = 1, \ldots, n;\ j = 0, \ldots\}$$

Proof: Since $f(\tau, t)$ is generally locally decreasing with respect to t, attention can be limited to the values of t for which the derivative is discontinuous, i.e., $t = jp_i + d_i$ for positive integer values j. ■

We may now show that the hyperperiod provides an upper bound on the maximum possible t that we need to evaluate $f(\tau, t)$.

Lemma 4 *Let $L = LCM_{i=1}^{n} p_i$. If $\delta_{sum}(\tau) > u_{sum}(\tau)$ then $\delta_{sum}(\tau) = f(\tau, t)$ for some $t \leq L$.*

Proof: The proof is by contradiction. Let $L + x$ be the least value for which $f(\tau, L + x) \geq u_{sum}(\tau)$ and $f(\tau, L + x) > f(\tau, t)$ for every $t < L + x$. If the lemma is false, there must be such a value.

Let $a_i = u_i L$ and $b_i = (\lfloor \frac{x - d_i}{p_i} \rfloor + 1) e_i$.

$$\begin{aligned} f(\tau, L + x) &= \sum_{i:d_i < L+x}^{n} \frac{(\lfloor \frac{L+x-d_i}{p_i} \rfloor + 1) e_i}{L + x} \\ &= \sum_{i:d_i < L+x}^{n} \frac{\frac{L}{p_i} e_i + (\lfloor \frac{x-d_i}{p_i} \rfloor + 1) e_i}{L + x} \\ &= \sum_{i:d_i < L+x}^{n} \frac{a_i + b_i}{L + x} \geq u_{sum}(\tau) \geq \sum_{i=1}^{n} \frac{a_i}{L} \end{aligned}$$

By algebra it can be shown that

$$\sum_{i:d_i<L+x}^{n} \frac{a_i+b_i}{L+x} \geq \sum_{i=1}^{n} \frac{a_i}{L} \Rightarrow \sum_{i:d_i<L+x}^{n} \frac{a_i+b_i}{L+x} \leq \sum_{i=1}^{n} \frac{b_i}{x}$$

Therefore,

$$f(\tau, L+x) \leq f(\tau, x)$$

which is a contradiction. ■

The following corollary to Lemma 4 and Lemma 1 shows that if $f(\tau,t)$ does not exceed $u_{sum}(\tau)$ prior to the hyperperiod of τ, we may infer that $\delta_{sum}(\tau) = u_{sum}(\tau)$.

Corollary 1 *If $f(\tau,t) \leq u_{sum}(\tau)$ for all $t \leq L$, then $\delta_{sum}(\tau) = u_{sum}(\tau)$.*

Lemmas 3 and 4, and Corollary 1 imply that we need to only check $f(\tau,t)$ up to the task system's hyperperiod. We can further limit the number of values t that need to be considered in the exact computation of δ_{sum} if we iteratively make use of the information we have gained by looking at the value of $f(\tau,t)$ up to any given point in the computation. The next lemma formalizes this concept:

Lemma 5 *If $f(\tau,t) \geq u_{sum}(\tau) + \gamma$ for some $\gamma > 0$ then $t \leq u_{sum}(\tau)\frac{\max_{\tau_i \in \tau}(p_i - d_i)}{\gamma}$.*

Proof: Observe that $\max_{\tau_i \in \tau}(p_i - d_i) > 0$; otherwise, for all $\tau_i \in \tau$,

$$\begin{aligned}
\left(\left\lfloor \frac{t-d_i}{p_i} \right\rfloor + 1\right) e_i \quad &\leq \left(\tfrac{t-d_i}{p_i} + 1\right) e_i \\
&\leq u_i t - u_i d_i + u_i p_i \quad \leq u_i t \\
\Rightarrow \quad &\mathrm{DBF}(\tau_i, t) \quad \leq u_i t \\
\Rightarrow \quad &f(\tau, t) \quad \leq u_{sum}(\tau)
\end{aligned}$$

The last statement contradicts the supposition of the lemma. Thus, there must exist a $\tau_i \in \tau$ such that $p_i - d_i > 0$. Therefore,

$$\begin{aligned}
u_{sum}(\tau) + \gamma \quad &\leq \quad f(\tau,t) = \frac{\sum_{i=1}^{n} \max(0, (\lfloor \frac{t-d_i}{p_i} \rfloor + 1) e_i)}{t} \\
&\leq \quad \sum_{i:d_i<t} u_i + \sum_{i:d_i<t} u_i \frac{\max_{\tau_j \in \tau}(p_j - d_j)}{t} \\
&\leq \quad u_{sum}(\tau)\left(1 + \frac{\max_{\tau_i \in \tau}(p_i - d_i)}{t}\right) \\
\Rightarrow \gamma \quad &\leq \quad u_{sum}(\tau)\frac{\max_{\tau_i \in \tau}(p_i - d_i)}{t} \\
\Rightarrow t \quad &\leq \quad u_{sum}(\tau)\frac{\max_{\tau_i \in \tau}(p_i - d_i)}{\gamma}
\end{aligned}$$

■

Our algorithm for calculating $\delta_{sum}(\tau)$ is represented in Figure 3 by CALCULATE–δ_{sum}. The subroutine exactly calculates $\delta_{sum}(\tau)$ when passed an ϵ parameter equal to zero; note that when $\epsilon = 0$, Line 1 always sets `limit` to the hyperperiod, and Line 6 never updates `limit` (non-zero ϵ values will be discussed in Section 5.1). Lemmas 3, 4, and 5, and Corollary 1 show that CALCULATE–δ_{sum} is correct when $\epsilon = 0$.

Run-time Complexity The "worst-case scenario" with respect to CALCULATE–$\delta_{sum}(\tau, 0)$'s execution time occurs when $\delta_{sum}(\tau) = u_{sum}(\tau)$; in this case, $f(\tau,t) \leq u_{sum}(\tau)$ for all $t > 0$. Therefore, `fmax` $\leq u_{sum}(\tau)$ for all iterations of CALCULATE–$\delta_{sum}(\tau, 0)$. Observe that Line 6 of CALCULATE–δ_{sum} updates `limit` only if `fmax` $- u_{sum}(\tau) > 0$ (assuming $\epsilon = 0$). Consequentially, `limit` is never updated after Line 1 sets it to τ's hyperperiod, and the algorithm calculates $f(\tau,t)$ for all integer values in the task system's hyperperiod. If $p_1, p_2, \ldots, p_n$ are relatively prime and p_i are integers greater than 1, then $LCM_{i=1}^{n} p_i = \prod_{i=1}^{n} p_i \geq 2^n$. Therefore, the number of time $f(\tau,t)$ is evaluated in CALCULATE–$\delta_{sum}(\tau, 0)$ is potentially exponential in the number of tasks in the task system.

5 Approximation Algorithms

We can accelerate the convergence of our load-bound function calculation if we permit a bounded level of inaccuracy in our calculation. That is, we can significantly reduce the number of values of t we must consider if we allow our calculated value of $\delta_{sum}(\tau)$ to lie within a specified range (or "tolerance") of the actual value.

Let ϵ denote a tolerance within which $\delta_{sum}(\tau)$ is to be approximated, for arbitrary $\epsilon > 0$. In this section, we propose two algorithms that calculate $\delta_{sum}(\tau)$ to within an additive error ϵ of its actual value. The first algorithm, discussed in Section 5.1, is a pseudo-polynomial-time algorithm based on the idea of iterative convergence. The second algorithm, presented in Section 5.2, is a polynomial-time approximation scheme for determining load.

5.1 Pseudo-polynomial-time Approximation Scheme

The effect of introducing a bounded amount of inaccuracy allows us to limit the number of values of t at which we evaluate $f(\tau,t)$. A useful observation is that after a sufficiently large value of t, $f(\tau,t)$ does not exceed $u_{sum}(\tau)$ by more than ϵ. The next lemma quantifies the value of t for which $f(\tau,t)$ is within ϵ of u_{sum}:

Lemma 6 *If $t \geq \frac{\sum_{i=1}^{n} e_i}{\epsilon}$, then $f(\tau,t) \leq u_{sum}(\tau) + \epsilon$.*

Proof: For any $\tau_i \in \tau$ with $t \geq d_i$, $\mathrm{DBF}(\tau_i, t) = (\lfloor \frac{t-d_i}{p_i} \rfloor + 1)e_i$; otherwise, if $t < d_i$, $\mathrm{DBF}(\tau_i, t) = 0$.

CALCULATE–$\delta_{sum}(\tau, \epsilon)$

▷ Interpret a divide by zero, as infinity; ϵ can be zero for exact-case.

1 limit $\leftarrow \min\left\{(LCM_{i=1}^{n} p_i), \left(\frac{\sum_{i=1}^{n} e_i}{\epsilon}\right), \left(u_{sum}(\tau)\frac{\max_{\tau_i \in \tau}(p_i - d_i)}{\epsilon}\right)\right\}$

2 fmax $\leftarrow u_{sum}(\tau)$;

3 **for** each $t = jp_i + d_i$, in increasing order, **loop**

exit when $t \geq$ limit;

4 **if** $f(\tau, t) >$ fmax

then

5 fmax $\leftarrow f(\tau, t)$;

6 limit $\leftarrow \min(\text{limit}, u_{sum}(\tau)\frac{\max_{\tau_i \in \tau}(p_i - d_i)}{\text{fmax} - u_{sum}(\tau) + \epsilon})$ ▷ Interpret a divide by zero, as infinity, again.

7 **if** fmax$> \lambda_{sum}(\tau) - \epsilon$ **then return** $\lambda_{sum}(\tau)$;

8 **end if;**

9 **end loop;**

10 **return** fmax;

Figure 3. Pseudo-code for determining load-bound function within a value of ϵ. When ϵ equals zero, the algorithm calculates the exact value of the load-bound function; otherwise, it is an approximation.

Since $\lfloor x \rfloor \leq x$, the function $f(\tau, t)$ can be bounded above as follows:

$$f(\tau,t) \leq \sum_{i:d_i<t} \frac{\frac{t-d_i}{p_i}e_i + e_i}{t} \leq \sum_{i:d_i<t} u_i(1 - \frac{d_i}{t}) + \frac{\sum_{i=1}^{n} e_i}{t}$$

$$\Rightarrow f(\tau,t) \leq u_{sum}(\tau) - \frac{\sum_{i:d_i<t} u_i d_i}{t} + \frac{\sum_{i=1}^{n} e_i}{t}$$

It follows that for $t \geq \frac{\sum_{i=1}^{n} e_i}{\epsilon}$,

$$f(\tau,t) \leq u_{sum}(\tau) - \frac{\sum_{i:d_i<t} u_i d_i}{t} + \epsilon$$

Since $-\frac{\sum_{i:d_i<t} u_i d_i}{t} < 0$, for all $t > \frac{\sum_{i=1}^{n} e_i}{\epsilon}$ the following condition is true: $f(\tau, t) \leq u_{sum}(\tau) + \epsilon$. ■

Since, `fmax` never decreases and is initially $u_{sum}(\tau)$ in CALCULATE–$\delta_{sum}(\tau, \epsilon)$, Lemma 6 implies that we need only evaluate values of $t \leq \frac{\sum_{i=1}^{n} e_i}{\epsilon}$. This optimization is reflected in Line 1 of CALCULATE–$\delta_{sum}(\tau, \epsilon)$.

In addition, we can use Lemma 5 to further reduce the number of steps taken by CALCULATE–$\delta_{sum}(\tau, \epsilon)$. Suppose at step i in an iterative approximate computation of $\delta_{sum}(\tau)$ the maximum value of $f(\tau, t)$ has been computed over all values $t \leq t_i$, and that value is `fmax`. The computation can terminate unless there is a value $t > t_0$ such that $f(\tau, t) > f(\tau, t_0) + \epsilon$. Letting $\gamma =$ `fmax` $- u_{sum}(\tau) + \epsilon$ in Lemma 5 above we have

$$t \leq u_{sum}(\tau)\frac{\max_{\tau_i \in \tau}(p_i - d_i)}{\text{fmax} - u_{sum}(\tau) + \epsilon}$$

The above observations are applied in Lines 1 and 6 of the algorithm CALCULATE–$\delta_{sum}(\tau, \epsilon)$ for approximate computation of $\delta_{sum}(\tau)$ (shown in Figure 3). Line 7 allows the algorithm to terminate early if `fmax` is within ϵ of our upper bound of $\lambda_{sum}(\tau)$.

Run-time Complexity The values of t for which we must evaluate $f(\tau, t)$ in CALCULATE–$\delta_{sum}(\tau, \epsilon)$ is at most $\min\left\{\left(\frac{\sum_{i=1}^{n} e_i}{\epsilon}\right), \left(u_{sum}(\tau)\frac{\max_{\tau_i \in \tau}(p_i - d_i)}{\epsilon}\right)\right\}$. Both of these terms are polynomial in the task system's parameters and $1/\epsilon$, and each evaluation of $f(\tau, t)$ requires $\mathcal{O}(n)$ time. Therefore, CALCULATE–$\delta_{sum}(\tau, \epsilon)$ represents a *pseudo-polynomial approximation scheme* (PPTAS).

5.2 Polynomial-time Approximation Scheme

Further theoretical reduction in the number of potential values for which to evaluate $f(\tau, t)$ can be achieved by an approximation to DBF(τ_i, t). Our approximation will allow us to "skip" intermediate test points in the calculation of $\delta_{sum}(\tau)$. We may approximate DBF(τ_i, t) for each task τ_i by "tracking" it exactly for $k_i + 1$ steps (how to pick k_i will be discussed shortly), and then using the tightest linear upper bound of DBF(τ_i, t) with slope u_i after $k_i + 1$ steps. A similar approximation was defined by Albers and Slomka [1] for uniprocessor feasibility analysis. For-

mally, the approximation can be expressed by:

$$\mathrm{DBF}^*(\tau_i,t) \stackrel{\text{def}}{=} \begin{cases} \mathrm{DBF}(\tau_i,t), & \text{if } t < k_i p_i + d_i, \\ e_i + (t-d_i)u_i, & \text{otherwise} \end{cases}$$

We can now describe approximations to $\delta_{sum}(\tau)$ using $\mathrm{DBF}^*(\tau_i,t)$:

$$\begin{array}{ll} f^*(\tau_i,t) & \stackrel{\text{def}}{=} \frac{\mathrm{DBF}^*(\tau_i,t)}{t}, \\ f^*(\tau,t) & \stackrel{\text{def}}{=} \sum_{i=1}^{n} f^*(\tau_i,t), \end{array}$$

and

$$\delta^*_{sum}(\tau) \stackrel{\text{def}}{=} \max_{t>0} f^*(\tau,t).$$

Visual examples of $f^*(\tau_i,t)$ are shown in Figure 2 with $k_i = 0$.

It can be shown that if we pick $k_i \stackrel{\text{def}}{=} \max\left(\lceil \frac{nu_i}{\epsilon} - \frac{d_i}{p_i} \rceil, 0\right)$, then $\delta^*_{sum}(\tau)$ is within ϵ of $\delta_{sum}(\tau)$. The following lemma proves this assertion:

Lemma 7 *For sporadic task system τ, if for all $\tau_i \in \tau$, $k_i = \max\left(\lceil \frac{nu_i}{\epsilon} - \frac{d_i}{p_i} \rceil, 0\right)$, then $\delta_{sum}(\tau) \le \delta^*_{sum}(\tau) \le \delta_{sum}(\tau) + \epsilon$.*

Proof: To prove the lemma it suffices to show for all $t > 0$ that $f(\tau,t) \le f^*(\tau,t) \le f(\tau,t) + \epsilon$. Obviously, $f(\tau,t) \le f^*(\tau,t)$; so, we will focus on showing that $f^*(\tau,t) \le f(\tau,t) + \epsilon$ for the k_i specified in the lemma.

Consider the following partition of $\tau = \tau_{\mathrm{dbf-exact}}(t) \cup \tau_{\mathrm{dbf-approx}}(t)$ where $\tau_{\mathrm{dbf-approx}}(t) \stackrel{\text{def}}{=} \{\tau_i | k_i p_i + d_i \le t\}$ and $\tau_{\mathrm{dbf-exact}}(t) \stackrel{\text{def}}{=} \tau - \tau_{\mathrm{dbf-approx}}(t)$. Informally, $\tau_{\mathrm{dbf-exact}}(t)$ is the set of tasks that have not taken $k_i + 1$ exact steps of DBF^* at time t; and, $\tau_{\mathrm{dbf-approx}}(t)$ is the set of tasks where $\mathrm{DBF}^*(\tau_i,t)$ uses the linear approximation to $\mathrm{DBF}(\tau_i,t)$ at time t.

Observe that $\mathrm{DBF}^*(\tau_i,t) \le \mathrm{DBF}(\tau_i,t) + e_i$ for all $t > 0$ and $\tau_i \in \tau$. Therefore,

$$\begin{array}{ll} f^*(\tau,t) & = \sum_{\tau_i \in \tau_{\mathrm{dbf-exact}}} \frac{\mathrm{DBF}(\tau_i,t)}{t} \\ & \quad + \sum_{\tau_i \in \tau_{\mathrm{dbf-approx}}} \frac{\mathrm{DBF}^*(\tau_i,t)}{t} \\ & \le \sum_{i=1}^{n} \frac{\mathrm{DBF}(\tau_i,t)}{t} + \sum_{\tau_i \in \tau_{\mathrm{dbf-approx}}} \frac{e_i}{t} \\ & = f(\tau,t) + \sum_{\tau_i \in \tau_{\mathrm{dbf-approx}}} \frac{e_i}{t} \end{array}$$

Note that for all $\tau_i \in \tau_{\mathrm{dbf-approx}}$, the value of t is lower bounded by $k_i p_i + d_i$. This implies $t \ge \left(\frac{nu_i}{\epsilon} - \frac{d_i}{p_i}\right) p_i + d_i = \frac{ne_i}{\epsilon}$. Thus,

$$\begin{array}{ll} f^*(\tau,t) & \le f(\tau,t) + \sum_{\tau_i \in \tau_{\mathrm{dbf-approx}}} \frac{\epsilon}{n} \\ & \le f(\tau,t) + \epsilon \end{array}$$

■

The immediate implication of the previous lemma is that we may approximate $\delta_{sum}(\tau)$ by effectively computing $\delta^*_{sum}(\tau)$.

Informally speaking, to determine $\delta^*_{sum}(\tau)$ we need to only evaluate $f^*(\tau,t)$ at times t where the derivative of f^* is discontinuous. The points at which such discontinuities occur for a given sporadic task system τ are:

$$\mathcal{S}(\tau,\epsilon) \stackrel{\text{def}}{=} \bigcup_{i=1}^{n} \{jp_i + d_i | i = 1,\ldots,n; j = 0,\ldots,k_i\}$$

The next lemma formalizes the assertion that to correctly calculate $\delta^*_{sum}(\tau)$ it is sufficient to only evaluate $f^*(\tau,t)$ for values of $t \in \mathcal{S}(\tau,\epsilon)$. Let $t_1, t_2 \in \mathcal{S}(\tau,\epsilon)$ $(t_1 < t_2)$ be *adjacent* if there does not exist a $t' \in \mathcal{S}(\tau,\epsilon)$ such that $t_1 < t' < t_2$.

Lemma 8 *Consider any two adjacent elements of $t_1, t_2 \in \mathcal{S}(\tau,\epsilon) \cup \{0\}$ where $t_1 < t_2$; for all t such that $t_1 < t < t_2$, the following condition holds,*

$$f^*(\tau,t) \le \max(f^*(\tau,t_1), f^*(\tau,t_2), u_{sum}(\tau)).$$

Proof: Let $\tau_{\mathrm{dbf-approx}}(t)$ and $\tau_{\mathrm{dbf-exact}}(t)$ be the partition of τ defined in Lemma 7. Consider any t in the interval (t_1,t_2). Clearly, $\tau_{\mathrm{dbf-exact}}(t_1) = \tau_{\mathrm{dbf-exact}}(t)$. Moreover, there does not exist t' in (t_1,t_2), $\tau_i \in \tau_{\mathrm{dbf-exact}}(t_1)$, and $\ell \in \mathbb{N}^+$ such that $t' = \ell p_i + d_i$. This implies for all $\tau_i \in \tau_{\mathrm{dbf-exact}}(t)$,

$$\begin{array}{ll} f^*(\tau_i,t) & = \frac{\mathrm{DBF}(\tau,t_1)}{t} = \frac{\mathrm{DBF}(\tau,t_1)/t_1}{t/t_1} \\ & = \frac{t_1 f^*(\tau_i,t_1)}{t} \end{array}$$

Also, $\tau_{\mathrm{dbf-approx}}(t_1) = \tau_{\mathrm{dbf-approx}}(t)$ which implies for all $\tau_i \in \tau_{\mathrm{dbf-approx}}(t)$ that $f^*(\tau_i,t) = \frac{t_1 f^*(\tau_i,t_1)}{t} + \frac{u_i(t-t_1)}{t}$. So, for t in the interval (t_1,t_2), we may express $f^*(\tau,t)$ in terms of this partition:

$$\begin{array}{ll} f^*(\tau,t) & = \sum_{\tau_i \in \tau_{\mathrm{dbf-exact}}} f^*(\tau_i,t) \\ & \quad + \sum_{\tau_i \in \tau_{\mathrm{dbf-approx}}} f^*(\tau_i,t) \\ & = \sum_{\tau_i \in \tau_{\mathrm{dbf-exact}}} \frac{t_1 f^*(\tau_i,t_1)}{t} \\ & \quad + \sum_{\tau_i \in \tau_{\mathrm{dbf-approx}}} \left[\frac{t_1 f^*(\tau_i,t_1)}{t} + \frac{u_i(t-t_1)}{t}\right] \\ & = \frac{t_1 f^*(\tau,t_1) + (t-t_1)\sum_{\tau_i \in \tau_{\mathrm{dbf-approx}}} u_i}{t} \end{array}$$

Let us now look at the partial derivative of $f^*(\tau,t)$ with respect to t:

$$\begin{array}{ll} \frac{\partial f^*(\tau,t)}{\partial t} & = \frac{t\sum_{\tau_i \in \tau_{\mathrm{dbf-approx}}} u_i}{t^2} \\ & \quad - \frac{\left(t_1 f^*(t_1) + (t-t_1)\sum_{\tau_i \in \tau_{\mathrm{dbf-approx}}} u_i\right)}{t^2} \\ & = \frac{t_1\left(\sum_{\tau_i \in \tau_{\mathrm{dbf-approx}}} u_i - f^*(t_1)\right)}{t^2} \end{array}$$

PTAS–$\delta_{sum}(\tau, \epsilon)$

```
1  fmax ← u_sum(τ) + ε;
   ▷ k_i ≝ max(⌈nu_i/ε − d_i/p_i⌉, 0) for all τ_i ∈ τ.
2  for each t ∈ S(τ, ε), in increasing order, loop
3        if f*(τ, t) > fmax
           then
4             fmax ← f*(τ, t);
5             if fmax≥ λ_sum(τ) then return λ_sum(τ);
6        end if;
7  end loop;
8  return fmax;
```

Figure 4. Pseudo-code for determining load-bound function within a value of ϵ in polynomial-time.

Therefore, if $f^*(\tau, t_1) \geq \sum_{\tau_i \in \tau_{\text{dbf}-\text{approx}}} u_i$, then $f^*(\tau, t)$ is *non-increasing* and $f^*(\tau, t) \leq f^*(\tau, t_1)$. Otherwise, $f^*(\tau, t)$ is bounded from above by u_{sum}. To complete the lemma, consider t in interval $(0, \min\{\mathcal{S}\})$ (i.e. 0 and $\min\{\mathcal{S}\}$ are adjacent), in this case, $f^*(\tau, t) = 0 \leq f^*(\tau, \min\{\mathcal{S}\})$. ■

Let $t_{max} \stackrel{\text{def}}{=} \max\{t \in \mathcal{S}(\tau, \epsilon)\}$. By the previous lemma, values of t in the interval $(0, t_{max})$ that are not in $\mathcal{S}(\tau, \epsilon)$ do not contribute to the calculation of $\delta^*_{sum}(\tau)$ (i.e. $f^*(\tau, t) \neq \delta^*_{sum}(\tau)$). The next lemma shows that values of t in the interval (t_{max}, ∞) also do not contribute to $\delta^*_{sum}(\tau)$:

Lemma 9 *For all $t > t_{max}$, the following inequality holds* $\max\left(f^*(\tau, t_{max}), u_{sum}(\tau)\right) \geq f^*(\tau, t)$.

Proof: Define $\tau_{\text{dbf}-\text{approx}}(t_{max})$ as in Lemma 7. For all $t > t_{max}$ and $\tau_i \in \tau$, $t > k_i p_i + d_i$. This implies that $\tau_{\text{dbf}-\text{approx}}(t_{max})$ equals τ. Therefore $f^*(\tau, t) = \frac{t_{max} f^*(\tau, t_{max}) + (t - t_{max}) u_{sum}(\tau)}{t}$. If $f^*(\tau, t_{max}) > u_{sum}(\tau)$, then $f^*(\tau, t)$ is decreasing in the interval of (t_{max}, ∞) implying $f^*(\tau, t_{max}) \geq f^*(\tau, t)$; otherwise, $f^*(\tau, t) \leq u_{sum}(\tau)$. ■

The algorithm PTAS–$\delta_{sum}(\tau, \epsilon)$ is presented in Figure 4. Lemma 7 showed that to approximate $\delta_{sum}(\tau)$ we could calculate $\delta^*_{sum}(\tau)$ instead (via evaluating $f^*(\tau, t)$). Lemmas 8 and 9 showed that we only need to consider the values of t in the set $\mathcal{S}(\tau, \epsilon)$. By these lemmas, PTAS–$\delta_{sum}(\tau, \epsilon)$ correctly approximates $\delta_{sum}(\tau)$ to within an additive value of ϵ. It should be noted that the heuristics of Lemmas 4, 5, and 6 can also be applied to PTAS–$\delta_{sum}(\tau, \epsilon)$ to further speed-up computation.

The number of iterations of PTAS–$\delta_{sum}(\tau, \epsilon)$ is entirely based on the size of the set $\mathcal{S}(\tau, \epsilon)$. The size of the set is:

$$\sum_{i=1}^{n} (k_i + 1) \in \mathcal{O}\left(\frac{n^2}{\epsilon}\right).$$

because $k_i \leq n/\epsilon$. A straightforward implementation of $f^*(\tau, t)$ has $\mathcal{O}(n)$ complexity. Therefore, the total time complexity of PTAS–$\delta_{sum}(\tau, \epsilon)$ is $\mathcal{O}(n^3/\epsilon)$. The runtime of the algorithm is polynomial in the number of tasks and ϵ, and independent of the task parameters. Therefore, PTAS–$\delta_{sum}(\tau, \epsilon)$ represents a polynomial-time approximation scheme.

6 Performance Tests

To give some idea of the advantage of the load-bound function $\delta_{sum}(\tau)$ over $u_{sum}(\tau)$ as a measure of computational demand, the real-time cost of computing it, and the advantages of the various heuristics used to limit the range of time values considered in the computation, the values of $u_{sum}(\tau)$, $\delta_{sum}(\tau)$, and $\lambda_{sum}(\tau)$ were computed for several collections of pseudo-randomly chosen sporadic task systems. The computation of $\delta_{sum}(\tau)$ was done approximately, using the tolerance $\epsilon = 0.001$.

The results of one representative experiment are summarized in figures below. Each experiment involved 1,000,000 systems, with up to 63 tasks in each system, all of which had values of $u_{sum}(\tau) \leq m$ (not demonstrably infeasible by the utilization bound test). For all the experiments the task periods were chosen uniformly from 1 to 1000.

For the experiment whose results are shown in the figures here, the maximum total utilization per task set was 2, the utilizations were chosen uniformly from $1/p_i$ to 1, and the deadlines were chosen according uniformly from e_i to p_i.

Figure 5(a) shows how often $\delta_{sum}(\tau)$ is more accurate than $u_{sum}(\tau)$ for proving infeasibility. It is a histogram, with the horizontal axis corresponding to one-percent ranges of the value $u_{sum}(\tau)/2$. The upper line indicates the count of task systems generated with $u_{sum}(\tau) \leq 2$. The lower line, plotted with asterisks, indicates the count of task systems generated that are feasible on two processors according to the test $\lambda_{sum}(\tau) \leq 2$. The middle line, plotted with boxes, indicates the count of task systems that might be feasible according to the $\delta_{sum}(\tau) \leq 2$ test. The area between the middle and upper lines (which is large) indicates the number of cases in which there was a gain in accuracy of $\delta_{sum}(\tau)$ over $u_{sum}(\tau)$ as a test for infeasibility.

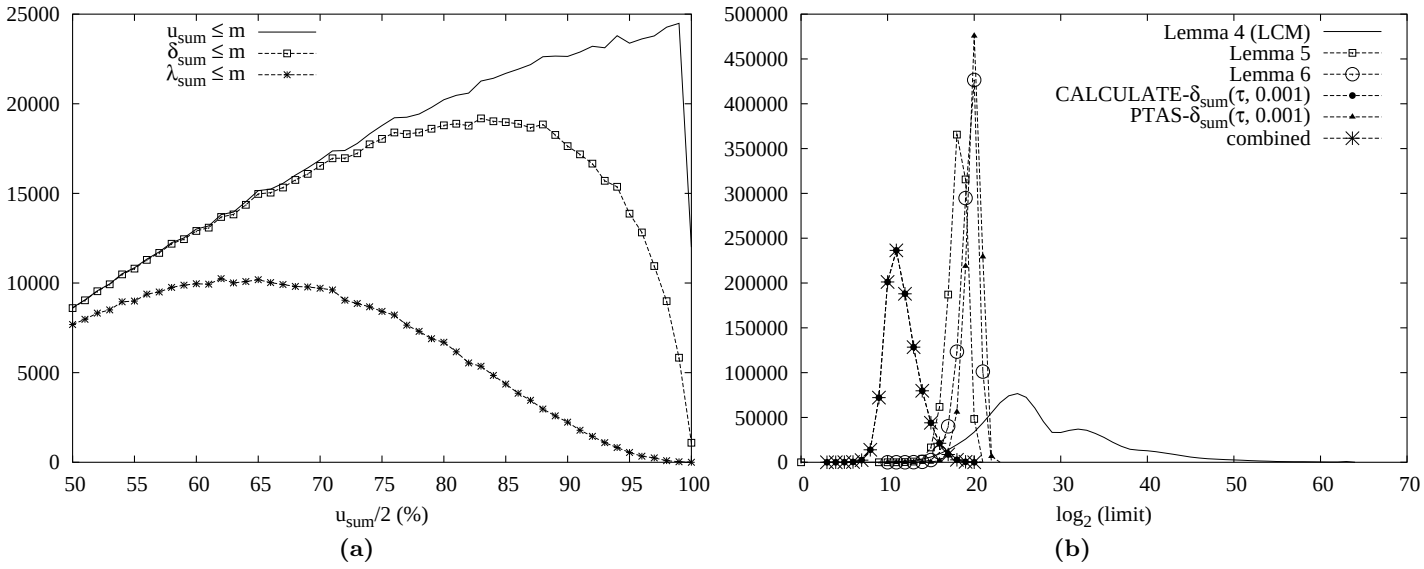

Figure 5. Comparison of heuristics (a) and number of iterations (b).

6.1 Relative Computational Efficiency of Load Algorithms

In this section, we compare the computational efficiency of each of the algorithms discussed in this paper. Our results show that the pseudo-polynomial heuristics empirically provide the most efficient method for computing $\delta_{sum}(\tau)$. Figure 5(b) shows how much the time to compute $\delta_{sum}(\tau)$ is reduced by each of the heuristics used. The horizontal axis is logarithmic, and represents the highest value of t that needed to be considered in computing $\delta_{sum}(\tau)$. The plot labeled "Lemma 4 (LCM)" corresponds to the hyperperiod (least common multiple of the task periods) bound of Lemma 4. The plot labeled "Lemma 5" represents the initial bound provided by Lemma 5 using $u_{sum}(\tau)$ as approximation for $\delta_{sum}(\tau)$. The plot labeled "Lemma 6" represents the bound on t obtained by Lemma 6. The plot labeled "CALCULATE–δ_{sum}" represents the actual largest value of t considered by the pseudo-polynomial time algorithm, the above heuristics, as well as the application of Lemma 5 iteratively to reduce the bound, and can be smaller than that bound if it turns out that a load greater than $\lambda_{sum}(\tau) - \epsilon$ is found early. The plot labeled "PTAS–δ_{sum}" indicates the actual largest value of t considered by that algorithm, with no other heuristics. The plot labeled "combined" indicates the largest t considered by using PTAS–δ_{sum} with the heuristics described in Lemmas 4, 5, and 6.

The following patterns can be observed:

1. The LCM (X's in the graph) can be very large. For other experiments, with higher values of m, significant numbers of systems had LCMs that overflowed the 64-bit precision used for the computation.

2. The Lemma 5 heuristic based on u_{sum} (boxes in the graph) reduced the search considerably, as shown by the large spike around 2^{19}.

3. The Lemma 6 heuristic (circles in the graph) cut down the search by about one more binary order of magnitude.

4. The heuristics of the pseudo-polynomial-time scheme CALCULATE–δ_{sum} (shown by the solid dots) further reduces the range of values considered, by about eight binary orders of magnitude, as shown by the spike around 2^{11}.

5. The polynomial-time approximation scheme PTAS–δ_{sum} (shown by the triangles) is tested here without any other heuristics; thus, the maximum value of t considered by the algorithm is still quite large. The strength of the scheme lies in not having to examine all the points within the search range.

6. The heuristics of Lemmas 4, 5, and 6 applied to PTAS–δ_{sum} (shown by asterisks) give the same performance as CALCULATE–δ_{sum}, as shown by the spike around 2^9.

Figure 6(a) is a histogram comparing the distributions of compute times for CALCULATE–δ_{sum} (shown by the X's) against the compute times for PTAS–δ_{sum}, for the collection of 1,000,000 task sets described above, with utilization in the range 1 to 2. Figure 6(b) is for

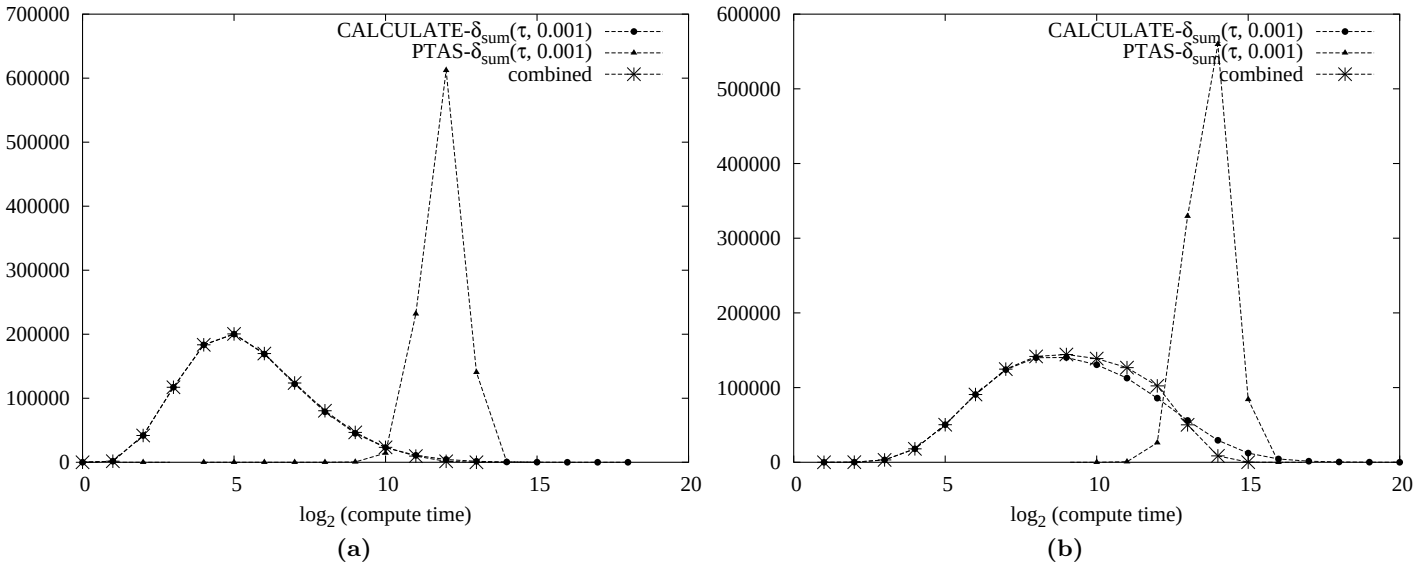

Figure 6. Comparison of actual number of steps required.

a similar collection of task sets with utilizations in the range 2 to 4 (and so, on the average, two times as many tasks per set). The horizontal scale is logarithmic, and represents the number of iterations (*i.e.*, evaluations of $f(\tau, t)$), using the two algorithms presented here. The line labeled "combined" in both Figure 6(a) and (b) is the heuristics of Lemma 4, 5, and 6 applied to PTAS–δ_{sum}. The combined (polynomial-time) method slightly more efficiently computes $\delta_{sum}(\tau)$ for a larger number of tasks sets than CALCULATE–δ_{sum}. Both the combined method and CALCULATE–δ_{sum} perform better than PTAS–δ_{sum} (without heuristics) in the experiments.

7 Conclusion

The utilization and density parameters of a task system do not effectively characterize the computational demand of a sporadic task system when relative deadlines can differ from periods. For these task systems, the computational demand can be bounded below by the utilization and above by the density. Previous and ongoing research has developed feasibility and schedulability conditions based on the *demand-based load* $\delta_{sum}(\tau)$ of a sporadic task system τ. For these conditions to be practical, efficient methods of computing $\delta_{sum}(\tau)$ must be devised. We describe in this report how to compute $\delta_{sum}(\tau)$ of a sporadic task system by providing an exact and approximate algorithms. We demonstrate that it is possible to approximate $\delta_{sum}(\tau)$ to within an additive constant $\epsilon > 0$ in polynomial-time. Experiments verify that the approximate algorithm also performs exceptionally well in practice.

References

[1] 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.

[2] S. Baruah and N. Fisher. The partitioned multiprocessor scheduling of sporadic task systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, Miami, Florida, December 2005. IEEE Computer Society Press.

[3] S. Baruah and N. Fisher. Multiprocessor feasibility analysis. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, Dresden, Germany, July 2006.

[4] S. Baruah, A. Mok, and L. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the 11th Real-Time Systems Symposium*, pages 182–190, Orlando, Florida, 1990. IEEE Computer Society Press.

[5] N. Fisher, S. Baruah, and T. Baker. The partitioned scheduling of sporadic tasks according to static-priorities. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, Dresden, Germany, July 2006.

[6] T. M. Ghazalie and T. P. Baker. Aperiodic servers in a deadline scheduling environment. *Real-Time Systems*, 9, 1995.

[7] 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.