Algorithms for Determining the Load of a Sporadic Task System
TR-051201

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

Nathan Fisher Sanjoy Baruah
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

Abstract

In this report, we discuss a metric that characterizes the load of a sporadic task system. We give an exact, exponential-time algorithm that determines a task system's load by essentially simulating the execution of task system. In addition, we also give an algorithm that can determine the load of a task system within an arbitrarily small threshold $\epsilon > 0$. While the worst-case time complexity of the approximation is still possibly exponential, we have empirically observed that this algorithm generally provides a very significant reduction in the time that we must simulate the task system to obtain the load. Additionally, we provide proofs of correctness for our algorithms.

1 Introduction

In the *sporadic task model* [4], 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. 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, $u_{sum}(\tau) \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau} u_i$. That $u_{sum}(\tau) \leq m$ is a necessary and sufficient condition for τ to be feasible on a preemptive m processor system. That this is a sufficient condition follows from the fact that if τ satisfies the utilization condition then we can "reserve" a fraction u_i of each time unit to execute τ_i on the processing platform.

For general deadlines (i.e. $d_i \neq p_i$) $u_{sum}(\tau)$ 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. 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.

$$\tau = \{\tau_1 = (1, 1, 2), \tau_2 = (1, 1, 2), \tau_3 = (1, 1, 2)\}$$

Observe that $u_{sum}(\tau) = 1.5 \leq 2$; 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. ■

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 $\lambda_i = \frac{e_i}{\min(d_i, p_i)}$. This is sometimes called the *density* of τ_i. It was shown in [3] that $\lambda_{sum}(\tau) \leq 1$ is a sufficient condition for feasibility of sporadic task systems with preemptive EDF scheduling. By the usual processor sharing argument, it follows that $\sum(\tau) \leq m$ is a sufficient condition for feasibility under ideal processor sharing with general deadlines. However, this condition is not necessary for feasibility, as shown by the following example.

Example 2 Consider the following sporadic task system consisting of three tasks to be scheduled on a multiprocessor system comprised of two unit-capacity processors.

$$\tau = \{\tau_1 = (1, 1, 1), \tau_2 = (1, 1, 2), \tau_3 = (1, 2, 3)\}$$

Observe that $\lambda_{sum}(\tau) = 2.5 > 2$, but the task system is clearly feasible, by scheduling task τ_1 on one processor and placing the other two tasks on the other processor. ■

In summary:

1. $u_{sum}(\tau)$ is a lower bound on demand, giving a necessary condition for feasibility
2. $\lambda_{sum}(\tau)$ is an upper bound on demand, giving 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 leave a gap, where there is uncertainty whether a task set is feasible.

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 that it can be computed effectively with enough accuracy to be useful in practice.

The conceptual relationship of δ_{sum} to u_{sum} and λ_{sum} 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 δ_{sum} is that it reduces the region of uncertainty significantly, as compared to u_{sum}.

To the best of our knowledge, there is no prior published algorithm for computing $\delta_{sum}(\tau)$ for an arbitrary sporadic task system. We give an exact algorithm $\delta_{sum}(\tau)$ that involves simulating τ to its hyperperiod (the least-common-multiple of the task systems periods, $LCM_{i=1}^{n} p_i$). This exact algorithm is too computationally expensive to be useful. Therefore, we give an algorithm which approximates $\delta_{sum}(\tau)$ within an arbitrary threshold $\epsilon > 0$ of its exact value. We have observed that our approximation algorithm

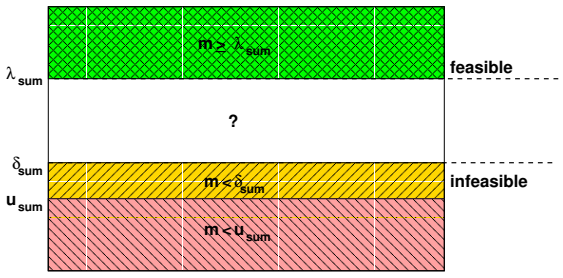

Figure 1. Relationship of bounds on computational load

achieves a significant reduction in computation time even for $\epsilon = 0.001$.

The remainder of the report is organized as follows. We formally define the load-bound function $\delta_{sum}(\tau)$ in Section 2. We then derive a simple method for exactly determining $\delta_{sum}(\tau)$ in Section 3. We describe the more practical approximation of load in Section 4. Finally, we empirically evaluate our algorithms on of a collection of pseudo-randomly generated task systems.

2 Load-Bound Function

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

$$f(\tau, t) \stackrel{\text{def}}{=} \frac{\sum_{i=1}^{n} \text{DBF}(\tau_i, t)}{t}$$

and

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

Figure 2 illustrates an example of $\text{DBF}(\tau_i, t)$ (see [2] for a formal discussion of $\text{DBF}(\tau_i, t)$). Since the values of t are real and unbounded, the notation $\max_{t>0}$ here denotes the least upper bound of $f(\tau, t)$.

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 [1]. 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 $\lambda_{sum}(\tau)$.

Example 3 To see that δ_{sum} more effective in detecting infeasible task sets, consider the following sporadic task system consisting of three tasks to be scheduled on a multiprocessor system comprised of two unit-capacity processors.

$$\tau = \{\tau_1 = (1, 1, 1), \tau_2 = (1, 1, 2), \tau_3 = (1, 1, 3)\}$$

Observe that $u_{sum}(\tau) = 1.83333 < 2 < \delta_{sum}(\tau) = 3$. The task system is not feasible, since if all tasks are released together at time zero three units of computation must

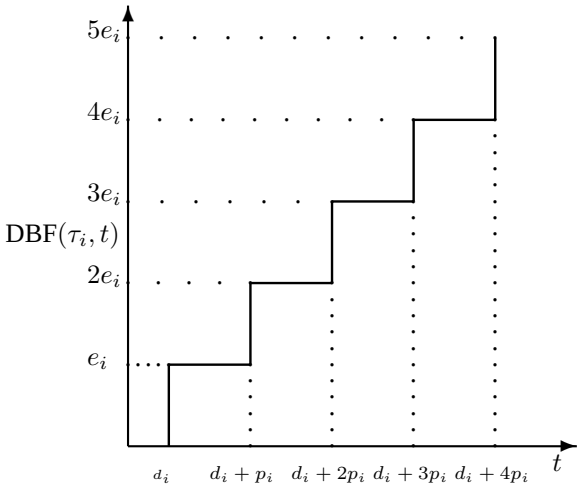

Figure 2. Plot of DBF(τ_i, t) as a function of t.

be completed by time 1. That this task system is not feasible cannot be detected using u_{sum}, since $u_{sum}(\tau) = 1.83333 < 2$. However, it can be detected by the fact that $\delta_{sum}(\tau) > 2$. ■

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

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

Proof: Observe that each term of $f(\tau, t)$ approaches u_i in the limit, for increasing values of t.

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 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:

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

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

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,

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

□

To see that δ_{sum} is non-trivially tighter than u_{sum} as a lower bound on load, yet still below λ_{sum}, reconsider the task set in Example 2 (which is feasible). Observe that $u_{sum}(\tau) = 1.8333333 < \delta_{sum}(\tau) = 2 < \lambda_{sum}(\tau) = 2.5$.

3 An Exact Algorithm

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 LCM provides an upper bound on the maximum possible t that we need to evaluate $f(\tau, t)$ at.

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

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

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 the correctness of the algorithm represented in Figure 3.

4 An Approximation Algorithm

We can further limit the number of values t that need to be considered if we introduce a bounded level of inaccuracy in our calculation. That is, for arbitrary $\epsilon > 0$, we can reduce the number of values of t we must consider if we allow our calculated value of $\delta_{sum}(\tau)$ to within ϵ of the actual value. The next lemma formalizes this concept:

EXACT–$\delta_{sum}(\tau)$

```
1  fmax ← u_sum(τ);
2  for each t = jp_i + d_i, in increasing order, loop
   exit when t ≥ LCM_{i=1}^{n} p_i;
3          if f(τ,t) > fmax then
4                  fmax ← f(τ,t);
5          end if;
6  end loop;
7  return fmax;
```

Figure 3. Pseudo-code for determining load-bound function exactly.

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

Proof:

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

□

Let ϵ be a tolerance within which δ_{sum} is to be approximated. Lemma 5 above allows us iteratively reduce the number of t values to check as the value of $f(\tau, t)$ increases greater than $u_{sum}(\tau)$. We may use this iterative reduction in steps in an approximation for $\delta_{sum}(\tau)$ shown in Figure 4.

5 Performance Tests

To give some idea of the advantage of the load-bound function δ_{sum} over u_{sum} as a measure of computational demand, the real-time cost of computing it, and the advantages of the three heuristics used to limit the range of time

APPROX–$\delta_{sum}(\tau, \epsilon)$

```
1  limit ← min(LCM_{i=1}^{n} p_i, u_sum(τ) max_{τ_i∈τ}(p_i − d_i) / ε)
2  fmax ← u_sum(τ);
3  for each t = jp_i + d_i, in increasing order, loop
   exit when t ≥ limit;
4        if f(τ, t) > fmax
           then
5              fmax ← f(τ, t);
6              limit ← min(limit, u_sum(τ) max_{τ_i∈τ}(p_i − d_i) / (fmax − u_sum(τ)))
7              exit when fmax> λ_sum(τ) − ε;
8        end if;
9  end loop;
10 return fmax;
```

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

values considered in the computation, the values of u_{sum}, δ_{sum}, and λ_{sum} were computed for several collections of pseudo-randomly chosen sporadic task systems. The computation of δ_{sum} was done approximately, using the tolerance $\epsilon = 0.001 \cdot m$.

The results of a 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 $u_{sum} \leq m$ (not demonstrably infeasible by the utilization bound test) and $\lambda_{sum}(\tau) > m$ (not verifiably feasible by the λ_{sum} test). For all the experiments the task periods were chosen uniformly from 1 to 1000.

For the experiment shown in Figures 5, 6, and 7, the number of processors was $m = 2$, 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 shows how often δ_{sum} is more accurate than u_{sum} for proving infeasibility. The horizontal axis corresponds to ranges of the value δ_{sum}. The upper line indicates the count of task systems that might be feasible according to the u_{sum} and λ_{sum} tests. The lower line, whose data points are plotted with X's, indicates the count of task systems that might be feasible according to the δ_{sum} test (i.e., $M \geq \delta_{sum}$).

The area between those lines (which is large) indicates the number of cases in which there was a gain in accuracy of δ_{sum} over u_{sum} as a test for infeasibility.

Figure 6 shows how much the time to compute δ_{sum} 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 δ_{sum}. The plot labeled "LCM" corresponds to the hyperperiod (least common multiple of the task periods) bound of Lemma 4. The plot labeled "BHR" represents the initial bound provided by Lemma 5 using u_{sum} as approximation for δ_{sum}. The plot labeled "ITR" represents the final bound on t obtained by iterative application of Lemma 5 during the computation. The plot labeled "ACT" represents the actual largest value of t considered, which can be smaller than the bound if it turns out that a load greater than $\lambda_{sum}(\tau) - \epsilon$ is found early. In this test, the computation was also halted early if a load value greater than m (indicating the task system is not feasible) was found.

The following patterns can be observed:

1. The LCM (X's in the graph) can be very large. Significant numbers of systems had LCMS that overflowed the 64-bit precision used for the computation. Those cases were mapped to zero, which caused a spike at data point zero in the histograms for some of the experiments.

2. The initial BHR heuristic based on u_{sum} (boxes in the graph) reduced the search considerably, as shown by the large spike between 2^{12} and 2^{20}.

3. The iterative application of BHR (circles in the graph) cut down the search by about 10 more binary orders of magnitude, as shown by the hump around 2^{10}.

4. The actual largest time value that needed to be considered (triangles in the graph) was often still smaller, due to early termination when when a deadline was missed. The one peak corresponds to the ITR bound, and the smaller earlier peak corresponds to the early terminations.

For most cases, the iterative application of the BHR heuristic, starting from u_{sum} would have been enough alone, without the LCM bound.

Figure 7 shows how much real time was required for the computation of δ_{sum}. The horizontal axis is logarithmic, and represents the amount of real time required to compute δ_{sum}, in nanoseconds. The computation was performed on a Pentium 4 Xeon processor at 2GHz. The longest execution time for any of these 1,000,000 task sets was 200 milliseconds, and the mean was 188 microseconds.

6 Conclusion

The utilization of a task system does 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.

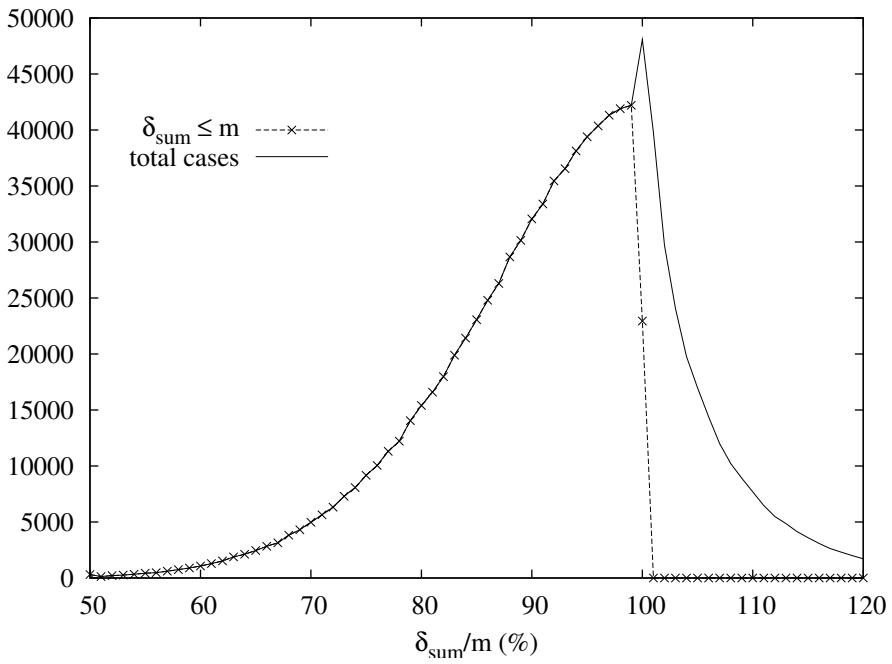

Figure 5. Advantage of δ_{sum} over u_{sum}: two processors, uniform utilizations, uniform constrained deadlines

However, for the lower bound, the load of a task system is a more accurate metric. We describe in this report how to compute the load of a sporadic task system by providing an exact and approximate algorithm. Both of these algorithms are proven correct. Experiments verify that the approximate algorithm performs well, and significantly narrows the range of uncertainty about whether a task system is feasible.

References

[1] 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.

[2] S. Baruah, R. Howell, and L. Rosier. Feasibility problems for recurring tasks on one processor. *Theoretical Computer Science*, 118(1):3–20, 1993.

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

[4] 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.

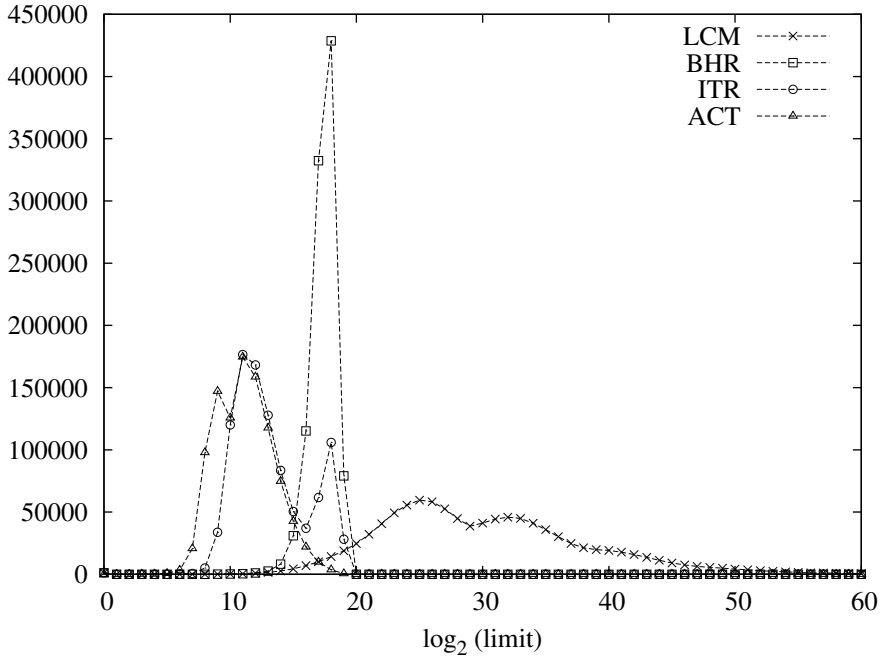

Figure 6. Comparison of search-limiting heuristics

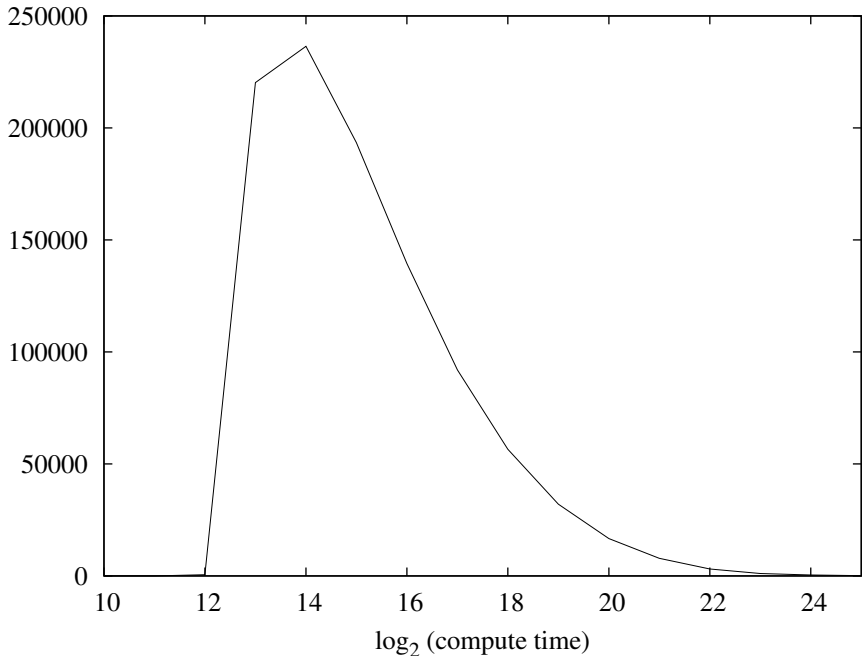

Figure 7. Time to compute load bound (nanoseconds)