HAL
open science

Quantifying the sub-optimality of uniprocessor fixed-priority scheduling

Sanjoy Baruah, Alan Burns

To cite this version:

Sanjoy Baruah, Alan Burns. Quantifying the sub-optimality of uniprocessor fixed-priority scheduling. 16th International Conference on Real-Time and Network Systems (RTNS 2008), Isabelle Puaut, Oct 2008, Rennes, France. inria-00336484

HAL Id: inria-00336484

https://inria.hal.science/inria-00336484v1

Submitted on 4 Nov 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Quantifying the sub-optimality of uniprocessor fixed-priority scheduling*

Sanjoy Baruah
The University of North Carolina
baruah@cs.unc.edu.edu

Alan Burns
The University of York
burns@cs.york.ac.uk.

Abstract

There are sporadic task systems schedulable by EDF, which no fixed-priority (FP) algorithm can schedule successfully. For instance, it is known that while all implicit-deadline sporadic task systems with utilization at most one are EDF-schedulable on a unit-capacity uniprocessor, there are implicit-deadline sporadic task systems that are not schedulable by any FP algorithm, and have utilization exceeding $\ln 2$ *(approx. 0.69) by an arbitrarily small amount. A quantitative metric of the relative effectiveness of fixed-priority scheduling of constrained-deadline sporadic task systems is derived here, which generalizes this well-known 0.69 utilization bound for implicit-deadline systems.*

Keywords. *Sporadic task systems; Preemptive uniprocessors; Fixed-priority scheduling; Earliest deadline first; Resource augmentation analysis.*

1. Introduction

The *sporadic task model* is widely used for representing real-time systems comprised of collections of independent recurrent real-time tasks. In this model, a task τ_i is characterized by three parameters – a *worst-case execution requirement* C_i, a *relative deadline* D_i, and a *minimum inter-arrival separation* or *period* T_i. Such a task $\tau_i = (C_i, D_i, T_i)$ generates an infinite sequence of jobs, with successive jobs arriving at least T_i time units apart and each job having an execution requirement no more than C_i and a deadline D_i time-units after its arrival time. A task system τ is comprised of a collection of such tasks which execute upon a shared computing platform. The utilization $\mathcal{U}(\tau)$ of sporadic task system τ is defined to be the sum of the ratios of the tasks' WCET's to their periods: $\mathcal{U}(\tau) \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau} C_i/T_i$.

A number of different scheduling algorithms have been proposed for scheduling sporadic task systems upon preemptive uniprocessors. Some of the more widely studied ones are Earliest Deadline First (EDF) [10, 4, 2], and the Fixed Priority (FP) [10, 9, 7, 8, 13] algorithms Rate Monotonic (RM) [10] and Deadline Monotonic (DM) [9]. EDF is known to be *optimal* in scheduling such systems, in the sense that if a system can be scheduled to meet all deadlines by any algorithm then it is also be scheduled to meet all deadlines by EDF. All FP algorithms are known to not be optimal in this sense — there are task systems that can be scheduled by, e.g., EDF, but upon which all FP algorithms would miss deadlines.

Implicit-deadline — also known as "Liu and Layland" (LL) — sporadic task systems are sporadic task systems in which each task is constrained to have its relative-deadline parameter equal to its inter-arrival separation parameter ($D_i = T_i$ for all i). It has been shown [10] that RM can successfully schedule on a preemptive uniprocessor any LL task system with utilization at most $\ln 2$ (which is ≈ 0.69) times the computing capacity of the processor. In conjunction with the obvious observation that a necessary condition for a LL task system to be feasible – schedulable by an optimal clairvoyant algorithm – is that the utilization not exceed the processor's computing capacity, this immediately yields the conclusion that the "penalty" for using the rate-monotonic algorithm instead of some other algorithm (for instance, the optimal EDF) is no greater than $(1 - (\ln 2))$, or approx. 0.31, the computing capacity of the processor.

Constrained-deadline sporadic task systems are sporadic task systems in which each task is constrained to have its relative-deadline parameter be no larger than its inter-arrival separation parameter ($D_i \leq T_i$ for all i). The deadline monotonic scheduling algorithm (DM) [9] is widely used

*Supported in part by NSF Grant Nos. CNS-0408996, CCF-0541056, and CCR-0615197, ARO Grant No. W911NF-06-1-0425, and funding from IBM and the Intel Corporation.

for scheduling constrained-deadline sporadic task systems upon a preemptive uniprocessor. While the penalty bound ($\approx 31\%$ of the computing capacity of the processor) for using RM in scheduling LL task systems is known, no similar results are, to our knowledge, known with respect to DM scheduling (or any other FP algorithm) of constrained-deadline sporadic task systems. The objective of the research described in this paper is to obtain bounds for DM that are analogous to RM's utilization bound, thereby enabling us to provide a quantitative bound on the penalty for using DM scheduling. It is known [9] that any constrained-deadline system that is FP schedulable for some priority assignment is also DM-schedulable; this immediately implies that any FP scheme for scheduling constrained-deadline sporadic task systems also suffers a penalty at least this large.

The remainder of this paper is organized as follows. In Section 2 we describe the task and machine model used in this research, and provide additional useful definitions. In Section 3 we define, and explain, the *processor speedup factor* of a scheduling algorithm, a metric for quantifying the penalty that one pays for choosing the scheduling algorithm. We restate some currently-known results, viz., the optimality of EDF and the RM utilization bound, in terms of processor speedup factor. In Sections 4 and 5, we obtain a lower bound and an upper bound respectively, on DM's processor speedup factor; this enables us to provide quantitative bounds on DM's behavior vis-a-vis the behavior of an optimal algorithm such as EDF. We conclude in Section 6 with a brief summary of the results here, and a discussion on open questions.

2. Model

In this paper, we restrict our attention to the scheduling of sporadic task systems upon preemptive uniprocessors.

A processor is characterized by a speed parameter s, indicating that a job executing upon the the processor for t units of time completes $s \times t$ units of execution. We assume that the processor is fully preemptive: an executing job may be preempted at any instant, and such preemptions incur negligible overhead.

As stated in Section 1, each sporadic task $\tau_i = (C_i, D_i, T_i)$ is characterized by the three parameters WCET C_i, relative deadline D_i, and period T_i. Any collection of such sporadic tasks may legally generate infinitely many different collections of jobs. Of these, one particular collection is of particular importance:

Definition 1 (Synchronous arrival sequence (SAS)) *The synchronous arrival sequence (SAS) of jobs for any sporadic task system τ is the collection of jobs generated by the system when each task generates a job at time-instant zero, and each task τ_i generates a job at each integer multiple of T_i.*

The *utilization* of a sporadic task τ_i is defined to be the ratio C_i/T_i of its WCET to its period; the utilization of a sporadic task system is the sum of the utilizations of all the tasks that comprise it.

The *demand bound function* $\text{DBF}(\tau_i, t)$ of a sporadic task τ_i for an interval-length t is defined to be the maximum cumulative execution of jobs of τ that both arrive in, and have deadlines within, any time-interval of length t. Formulas are known [2, 12] for computing $\text{DBF}(\tau_i, t)$ for any given τ_i and t.

The *load* $\text{LOAD}(\tau)$ of a sporadic task system τ is defined as follows:

$$\text{LOAD}(\tau) \stackrel{\text{def}}{=} \max_{t \geq 0} \frac{\sum_{\tau_i \in \tau} \text{DBF}(\tau_i, t)}{t} . \qquad (1)$$

Efficient algorithms have been designed [1, 6, 5] for computing LOAD *exactly*, or *approximately* to any arbitrary degree of accuracy, in pseudo-polynomial[1] or polynomial time respectively.

2.1. Feasibility and schedulability

A sporadic task system is said to be *feasible* upon a specified platform if there exists a schedule for every collection of jobs that the task system may generate. Exact (necessary and sufficient) conditions for testing sporadic task systems for feasibility on preemptive uniprocessors are known:

- It has been shown [2] that any sporadic task system τ is feasible on a preemptive speed-s processor if and only if $\text{LOAD}(\tau) \leq s$.
- For the special case of implicit-deadline (LL) systems, a simpler feasibility test is known [10]: a LL sporadic task system is feasible on a preemptive speed-s processor if and only if its utilization is $\leq s$.

With respect to any scheduling algorithm A, a task system τ is said to be *A-schedulable* if and only if it is the case that

[1] Actually the computation is guaranteed to be pseudo-polynomial only for *bounded-utilization* systems; for general systems, it may take exponential time. However, bounded-utilization systems represent the common case by far.

algorithm A guarantees to meet all deadlines when scheduling τ. (Thus while feasibility is a property of a task system, A-schedulability depends upon both the algorithm A and the task system.)

2.2. Fixed priority (FP) scheduling

In *fixed priority (FP)* preemptive uniprocessor scheduling of sporadic task systems, there is a unique priority assigned to each task in the system, and at each instant the processor is assigned to the job of the greatest-priority task that is awaiting execution at that instant. The *rate-monotonic (RM)* scheduling algorithm is an example, in which tasks are assigned priorities in inverse order of their period parameters: tasks with smaller period are assigned greater priority with ties broken arbitrarily. It has been shown [10] that RM is an optimal FP scheduling algorithm for LL sporadic task systems in the sense that if such a task system is A-schedulable for any FP scheduling algorithm A, then it is RM-schedulable. The *deadline-monotonic (DM)* algorithm is another FP scheduling algorithm. In DM, tasks are assigned priorities according to their relative deadline parameters – the smaller the relative deadline the greater the priority (with ties broken arbitrarily). It has been shown [9] that DM priority assignment is an optimal priority assignment for constrained-deadline sporadic task systems in the same sens as above: if a constrained-deadline sporadic task system is A-schedulable for any FP algorithm A, then it is also DM-schedulable. (Observe that for LL task systems, both DM and RM assign priorities to the tasks in an identical manner.)

2.3. Earliest Deadline First (EDF) scheduling

In EDF scheduling, the processor is assigned at each time instant to the job with the earliest deadline that has not yet completed execution. As stated above, it is known that EDF is optimal in the scheduling of sporadic task systems: if a sporadic task system is A-schedulable for any scheduling algorithm, then it is EDF-schedulable as well. This fact, in conjunction with the known feasibility tests described above, immediately yields exact EDF-schedulability tests for sporadic task systems upon preemptive uniprocessoers: system τ is EDF-schedulable on a speed-s processor if and only if $\text{LOAD}(\tau) \leq s$. For LL system τ, $\mathcal{U}(\tau) \leq s$ is a simpler exact EDF-schedulability test.

3. Processor speedup factors: definition and significance

There are several approaches to quantifying the "goodness" or the effectiveness of different scheduling algorithms. One relatively recent novel approach [11] asks the following question about an algorithm:

> *Suppose that a task system is known to be feasible — schedulable by an all-powerful clairvoyant scheduler. How much additional resources (in terms of a greater number of processors, or faster processors, or a combination of both) over and above those needed by the clairvoyant scheduler would the algorithm need, in order to guarantee to meet all deadlines of this task system?*

This approach has come to be known as *resource augmentation* analysis. Since we are interested here in uniprocessor scheduling, we will restrict ourselves to only considering augmentations to the processor speed; this restricted form of resource-augmentation in the context of uniprocessor sporadic task systems is formalized in the following definition:

Definition 2 (Processor speedup factor) *Scheduling algorithm A for scheduling sporadic task systems has a* processor speedup factor *f if it is guaranteed that any sporadic task system that is feasible upon a specified platform is A-schedulable upon a platform in which each processor is at least f times as fast.*

Intuitively, the processor speedup factor of a scheduling algorithm measures how far it is from being optimal (in the sense of successfully scheduling all feasible task systems). A processor speedup factor equal to one implies that the scheduling algorithm is able to schedule all feasible task systems, while a processor speedup factor of infinity would imply that there are feasible task systems the scheduling algorithm cannot schedule, even if it is provided with processors that are arbitrarily faster. The smaller the processor speedup factor, the better the algorithm.

A recent series of papers (see, e.g., [3]) has compared FP and Earliest Deadline First (EDF) scheduling upon preemptive uniprocessors. One of the aspects highlighted in this comparison is that EDF is optimal (in the sense of scheduling any feasible system) on preemptive uniprocessors [10, 4]. On the other hand, there are LL task systems with utilization just a bit greater than $\ln 2$ (i.e., ≈ 0.69) that are not FP-schedulable.

This comparison can be re-phrased in terms of the processor speedup factor metric. Since EDF can schedule all feasible task systems, its processor speedup factor is one:

Theorem 1 *The processor speedup factor of preemptive uniprocessor EDF is one, when scheduling arbitrary sporadic task systems (and hence, constrained-deadline and LL systems as well, since these are special cases).* ■

Suppose, however, that we are given that a LL task system τ is known to be feasible upon a processor of speed s. It is possible that RM will need a processor of speed $(1/\ln 2) \times s$ (i.e., $\approx 1.44s$) in order to meet all deadlines for τ. This observation is formalized in the following theorem

Theorem 2 *The processor speedup factor of preemptive uniprocessor RM is* $(1/\ln 2)$ *(i.e.,* ≈ 1.44*), when scheduling implicit-deadline sporadic task systems.* ■

Theorem 2 above thus quantifies the "goodness" of RM when scheduling LL task systems. To our knowledge, no (upper or lower) bounds like these were previously known on the processor speedup factor of DM when scheduling constrained-deadline sporadic task systems. Such bounds are determined in the remaining sections of this paper.

4. A lower bound on processor speedup factor for FP scheduling

In this section, we determine a lower bound on the processor speedup factor of DM scheduling. Specifically, we show that DM has a processor speedup factor no smaller than 1.5. Since DM is an optimal FP scheduling algorithm for scheduling constrained-deadline sporadic task systems (as explained in Section 2 above), this immediately implies that no FP algorithm for scheduling constrained-deadline sporadic task systems can have a processor speedup factor smaller than 1.5. In conjunction with the result in Theorem 1 above that EDF has a processor speedup factor of unity, this indicates that one pays a significant penalty, in terms of processor speedup factor, in choosing FP scheduling over EDF.

Our approach in proving a lower bound on DM's processor speedup factor is as follows. We first show (Lemma 1 below) that there exist task systems with load no larger that $2/3$, upon which DM would miss deadlines. We know, based on previously obtained results (stated in Section 2 above) that such a system is feasible upon a processor of speed $2/3$. Together, these facts allow us to conclude that the processor speedup factor of DM is at least $1/(2/3)$, or $3/2$.

Lemma 1 *There are constrained-deadline task systems* τ *with* $\text{LOAD}(\tau) = 2/3 + \epsilon$ *that are not DM-schedulable upon a unit-capacity processor, for arbitrarily small positive* ϵ.

Proof: We present such a task system τ. This task system τ is comprised of the following three tasks:

$$\begin{array}{llll} \tau_1: & C_1 = 2; & D_1 = 6; & T_1 = 6 \\ \tau_2: & C_2 = 1; & D_2 = 8; & T_2 = 8 \\ \tau_3: & C_3 = 3; & D_3 = 9; & T_3 = 24 \end{array}$$

It is known [10, 9, 8] that a constrained-deadline sporadic task system is DM-schedulable if and only if its synchronous arrival sequence (SAS – see Definition 1) is scheduled by DM such that each task's first job meets its deadline. According to DM, τ_1 has the highest priority and τ_3 the lowest. Consequently, the synchronous arrival sequence (SAS) of τ would be scheduled by DM as shown in Figure 1.

As is evident from this diagram, τ_1's, τ_2's, and τ_3's first jobs all meet their first deadlines (at time-instants 6, 8, and 9 respectively) in the SAS; hence, task system τ is DM-schedulable. However, it is evident that if any of the three tasks's execution requirements (i.e., C_1, C_2, or C_3) is increased by an arbitrarily small amount, then τ_3's first job would miss its deadline at time-instant 9. This follows from the observations that

- Suppose that C_1 and/ or C_2 were to increase.
  - Jobs of the higher-priority tasks τ_1 and τ_2 already completely consume the processor over the interval $[0, 3)$. Hence, increasing either C_i or C_2 (or both) would postpone the start of the execution of τ_3's first job.
  - Jobs of τ_1 and τ_2 already completely consume the processor after time-instant 6, all the way to τ_3's job's deadline at 9. Hence, increasing either C_i or C_2 (or both) would not allow τ_3's job any additional execution over $[6, 9)$.

  That is, the start of τ_3's execution is postponed, but it does not get any additional execution to compensate. It consequently executes for < 3 time units, and misses its deadline.

- If C_3 were to increase while C_1 and C_2 were unchanged, then, too, τ_3's job would continue to execute for 3 time units (regardless of the fact that its requirement is now > 3), and would hence miss its deadline.

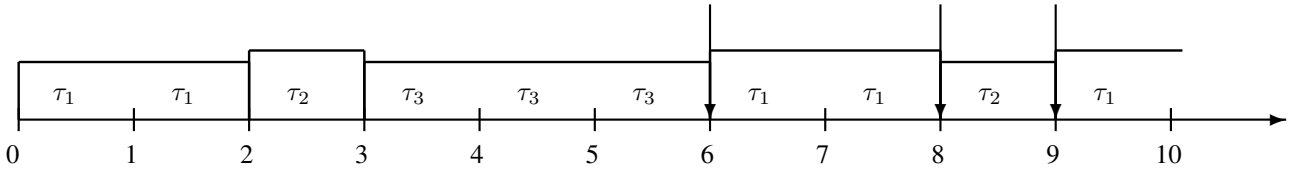

Figure 1. DM schedule on a preemptive unit-capacity processor for the synchronous arrival sequence (SAS) of the task system τ of in Section 4.

Hence, this task system is, in a sense, *maximally* DM-schedulable — increasing any of the execution requirements by an arbitrarily small amount would render it non DM-schedulable.

We use the algorithm in [5] to compute $\text{LOAD}(\tau)$. That is, we compute $(\text{DBF}(\tau_1, t) + \text{DBF}(\tau_3, t) + \text{DBF}(\tau_3, t))/t$ at all values of $t \in [0, 24)$ at which any of the DBF's changes, and determine the maximum of all these values[2]: the values are listed in Table 1. As can be seen, $(\sum_{i=1}^{3} \text{DBF}(\tau_i, t))/t$ is maximized for $t = 9$ and $t = 12$; at both these values, it equals $2/3$. We have thus determined that $\text{LOAD}(\tau) = 2/3$.

We have seen that task system τ

1. is DM-schedulable;
2. has $\text{LOAD}(\tau) = 2/3$; and
3. increasing any of C_1, C_2, or C_3 by an arbitrarily small amount would render it non- DM-schedulable.

Thus, task systems that are obtained from τ by increasing any of C_1, C_2, or C_3 by an arbitrarily small amount bear witness to the correctness of this lemma. ■

Using Lemma 1, a lower bound on the processor speedup factor of DM is easily obtained:

Theorem 3 *The processor speedup factor of preemptive uniprocessor DM is at least* $3/2$.

Proof: Consider some constrained-deadline sporadic task system τ with $\text{LOAD}(\tau) = 2/3+\epsilon$ for ϵ arbitrarily small and positive, that is not DM-schedulable upon a unit-capacity processor — the existence of such task systems is guaranteed by Lemma 1. As stated in Section 2, τ is feasible upon a speed-$(2/3 + \epsilon)$ processor. It immediately follows that the processor speedup bound of DM is $> 1/(2/3 + \epsilon)$; as $\epsilon \to 0$, we get the desired DM processor speedup bound of $3/2$. ■

[2]We can stop at $t = 24$ because $24 = \text{lcm}\{T_1, T_2, T_3\}$

5. An upper bound on processor speedup factor for FP scheduling

In Section 4 above, we showed that the processor speedup bound of DM is at least 1.5. We now derive an upper bound – we prove that DM has a processor speedup bound that is no greater than 2.0. In other words, we will show that DM is able to schedule any feasible constrained-deadline sporadic task system upon a processor that is at most twice as fast.

Lemma 2 *All constrained-deadline sporadic task systems τ with $\text{LOAD}(\tau) \leq \ell$ are DM-schedulable upon a speed-2ℓ processor.*

Proof: We prove the contrapositive of the statement of the lemma — we assume that τ is not DM-schedulable upon a speed-2ℓ processor, and show that $\text{LOAD}(\tau) > \ell$.

Let $\tau_1, \tau_2, \ldots$ denote the tasks in τ, indexed according to DM-priority:

$$\forall\, i :\ i \geq\ 1 : D_i \leq D_{i+1} \tag{2}$$

Since τ is not DM-schedulable upon a speed-2ℓ processor, the first job of some task in τ misses its deadline when the

$t =$	6	8	9	12	16	18	24
$\text{DBF}(\tau_1, t) + \text{DBF}(\tau_2, t) + \text{DBF}(\tau_3, t) =$	2	3	6	8	9	10	14
$(\text{DBF}(\tau_1, t) + \text{DBF}(\tau_2, t) + \text{DBF}(\tau_3, t)/t =$	1/3	3/8	**2/3**	**2/3**	9/16	5/9	7/12

Table 1. Computing DBF and LOAD for the task system τ of Section 4

synchronous arrival sequence (SAS) of τ is scheduled using DM upon such a processor. Let τ_k denote the smallest-indexed (i.e., highest-priority) task whose first job misses its deadline — which is at time-instant D_k — in this SAS schedule.

Consider now the DM schedule for the SAS of the task system $\tau' \stackrel{\text{def}}{=} \{\tau_1, \tau_2, \ldots, \tau_k\}$ — i.e., the task system obtained by discarding from τ all the tasks $\tau_{k+1}, \tau_{k+2}, \ldots$ that are of lower priority than τ_k. It follows from the definition of FP scheduling that the first job of τ_k will miss its deadline in this schedule as well.

Now, all jobs that are scheduled over the interval $[0, D_k)$ in the DM schedule of the SAS of τ' must have *arrived* prior to time-instant D_k; consequently, they all have both their arrival instants and their deadlines within the interval $[0, D_k + \max_{i=1}^{k-1}\{D_i\})$. By Condition 2 above, it follows that $\max_{i=1}^{k-1}\{D_i\}$ is no larger than D_k; i.e., $(D_k + \max_{i=1}^{k-1}\{D_i\}) \leq 2D_k$. Consequently, it is the case that all the jobs executing over $[0, D_k)$ in the DM schedule of the SAS of τ' arrive in, and have their deadlines within, the interval $[0, 2D_k)$ of size $2D_k$. Since these jobs completely consume the processor over this interval, and in fact do not allow τ_k's job to complete its execution, it must be the case that

$$\begin{aligned} & \sum_{\tau_i \in \tau'} \text{DBF}(\tau_i, 2D_k) \;>\; 2\ell D_k \\ \Rightarrow \quad & \text{LOAD}(\tau') \;>\; \frac{2\ell D_k}{2D_k} \\ \equiv \quad & \text{LOAD}(\tau') > \ell \\ \Rightarrow \quad & \text{LOAD}(\tau) > \ell \end{aligned}$$

and (the contra-positive of) the lemma is proved. ■

Theorem 4 *The processor speedup factor of preemptive uniprocessor DM is $\leq$ 2, when scheduling constrained-deadline sporadic task systems.*

Proof: Consider any constrained-deadline sporadic task system τ that is feasible upon a speed-s preemptive processor. By the results stated in Section 2, it must be the case that $\text{LOAD}(\tau) \leq s$. By Lemma 2, it follows that τ is DM-schedulable upon a speed-$2s$ processor. That is, any system that is feasible upon a speed-s processor is DM-schedulable upon a speed-$2s$ processor; the theorem follows. ■

6. Conclusions

Many considerations go into choosing a scheduling algorithm for use in a real-time system. One of these considerations should be *how far removed the algorithm is from optimal behavior*. The processor speedup factor of a scheduling algorithm is a quantitative metric of this: the larger the processor speedup factor, the further removed the algorithm is from exhibiting optimal behavior.

In this paper, we have determined upper and lower bounds on the processor speedup factor of uniprocessor FP algorithms in scheduling sporadic task systems. Using the well-known RM utilization bound, we have shown that FP scheduling has a processor speedup factor of ≈ 1.44 in scheduling implicit-deadline task systems. For constrained-deadline systems, we have shown that FP scheduling has a processor speedup factor of at least 1.5, and no more than 2. As is evident, there is a large gap between these lower and upper bounds — tightening this gap is the major open issue left unresolved by our work[3].

References

[1] T. P. Baker, N. Fisher, and S. Baruah. Algorithms for determining the load of a sporadic task system. Technical Report TR-051201, Department of Computer Science, Florida State University, 2005.

[2] S. Baruah, A. Mok, and L. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Pro-*

[3] In conjunction with Thomas Rothvoß and Robert Davis, we believe that we have recently obtained a proof that $1/\Omega$ is a tight bound on the processor speedup factor of uniprocessor FP algorithms for scheduling constarained-deadline sporadic task systems, where Ω is the well-known "omega constant" – approx. 0.567143. We are currently working on validating and writing up this result; if it holds, this will resolve the issue left open in this paper.

ceedings of the 11th Real-Time Systems Symposium, pages 182–190, Orlando, Florida, 1990. IEEE Computer Society Press.

[3] G. Buttazzo. Rate-monotonic vs. EDF: Judgement day. *Real-Time Systems: The International Journal of Time-Critical Computing*, 29(1):5–26, 2005.

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

[5] N. Fisher, T. Baker, and S. Baruah. Algorithms for determining the demand-based load of a sporadic task system. In *Proceedings of the International Conference on Real-time Computing Systems and Applications*, Sydney, Australia, August 2006. IEEE Computer Society Press.

[6] 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. IEEE Computer Society Press.

[7] M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, Oct. 1986.

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

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

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

[11] C. A. Phillips, C. Stein, E. Torng, and J. Wein. Optimal time-critical scheduling via resource augmentation. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 140–149, El Paso, Texas, 4–6 May 1997.

[12] I. Ripoll, A. Crespo, and A. K. Mok. Improvement in feasibility testing for real-time tasks. *Real-Time Systems: The International Journal of Time-Critical Computing*, 11:19–39, 1996.

[13] A. Wellings, M. Richardson, A. Burns, N. Audsley, and K. Tindell. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8:284–292, 1993.