Rate-monotonic scheduling on uniform multiprocessors*

Sanjoy K. Baruah
The University of North Carolina at Chapel Hill
Email: baruah@cs.unc.edu

Joël Goossens
Université Libre de Bruxelles
Email: joel.goossens@ulb.ac.be

Abstract

Each processor in a **uniform** *multiprocessor machine is characterized by a* speed *or computing capacity, with the interpretation that a job executing on a processor with speed s for t time units completes $(s \times t)$ units of execution. The scheduling of systems of periodic tasks on uniform multiprocessor platforms using the* **rate-monotonic scheduling algorithm** *is considered here. A simple, sufficient test is presented for determining whether a given periodic task system will be successfully scheduled by algorithm upon a particular uniform multiprocessor platform — this test generalizes earlier results concerning rate-monotonic scheduling upon identical multiprocessor platforms.*

Keywords. *Uniform multiprocessors; periodic tasks; global scheduling; static priorities; rate-monotonic algorithm.*

1. Introduction and Motivation

Safety-critical embedded systems are often comprised of very simple processes that are restricted to execute in strictly-controlled environments. Such systems are typically modelled as finite collections of simple, highly repetitive tasks, each of which generates jobs in a very predictable manner. These jobs have upper bounds upon their worst-case execution requirements, and associated deadlines. In the **periodic** model of hard real-time tasks, a **task** is characterized by two parameters – an execution requirement and a period – with the interpretation that the task generates a *job* at each integer multiple of its period, and each such job must execute for an amount equal to the execution requirement of the job by a deadline equal to the next integer multiple of the period. A periodic task system consists of several independent such periodic tasks that are to execute on a specified preemptive processor architecture.

Dynamic and static priorities *Run-time scheduling* is the process of determining, during the execution of a real-time application system, which job[s] should be executed at each instant in time. Run-time scheduling algorithms are typically implemented as follows: at each time instant, assign a **priority** to each active (informally, a job becomes *active* at its ready time, and remains so until it has executed for an amount of time equal to its execution requirement, or until its deadline has elapsed) job, and allocate the available processors to the highest-priority jobs.

With respect to certain run-time scheduling algorithms, it is possible that some tasks τ_i and τ_j both have active jobs at times t_1 and t_2 such that at time t_1, τ_i's job has higher priority than τ_j's while at time t_2, τ_j's job has higher priority than τ_i's. Run-time scheduling algorithms that permit such "switching" of the order of priorities between tasks are known as **dynamic** priority algorithms. The earliest deadline first scheduling algorithm (Algorithm EDF) [10, 6] is an example of a dynamic priority algorithm. By contrast, **static** priority algorithms must satisfy the constraint that for every pair of tasks τ_i and τ_j, whenever τ_i and

*Supported in part by the National Science Foundation (Grant Nos. CCR-9988327, ITR-0082866, and CCR-0204312).

τ_j both have active jobs, it is always the case that the same task's jobs have higher priority.

For systems comprised of periodic tasks that are to execute upon a *single shared processor*, a very popular static-priority run-time scheduling algorithm is the *rate-monotonic scheduling algorithm* (Algorithm RM) [10]. Algorithm RM assigns each task a priority inversely proportional to its period – the smaller the period, the higher the priority, with ties broken arbitrarily but in a consistent manner: if periodic tasks τ_i and τ_j have equal periods and τ_i's job is given priority over τ_j's job once, then all of τ_i's jobs are given priority over all of τ_j's jobs.

Multiprocessor Machines. In multiprocessor computing platforms there are several processors available upon which jobs may execute. In this paper, we study the scheduling of hard-real-time systems on multiprocessor platforms, under the assumptions that while job *preemption* and *interprocessor migration* is permitted (i.e., a job executing on a processor can be interrupted at any time, and its execution resumed later on the same or a different processor, with no cost or penalty), *intra-job parallelism* is forbidden (i.e., at any instant in time each job may be executing on at most one processor).

In much previous work concerning hard-real-time scheduling on multiprocessors, it has been assumed that all processors are identical. However, scheduling theorists distinguish between at least three different kinds of multiprocessor machines:

Identical parallel machines: These are multiprocessors in which all the processors are identical, in the sense that they have the same computing power.

Uniform parallel machines: By contrast, each processor in a *uniform parallel* machine is characterized by its own computing capacity, with the interpretation that a job that executes on a processor of computing capacity s for t time units completes $s \times t$ units of execution. (Observe that identical parallel machines are a special case of uniform parallel machines, in which the computing capacities of all processors are equal.)

Unrelated parallel machines: In *unrelated parallel* machines, there is an execution rate $r_{i,j}$ associated with each job-processor ordered pair (J_i, P_j), with the interpretation that job J_i completes $(r_{i,j} \times t)$ units of execution by executing on processor P_j for t time units.

Real-time scheduling theorists have extensively studied *uniprocessor* hard-real-time scheduling; recently, steps have been taken towards obtaining a better understanding of hard-real-time scheduling on *identical multiprocessors* (see, e.g., [4, 12, 11, 3, 1]). However, not much is known about hard-real-time scheduling on uniform or unrelated multiprocessors. While it may be argued that the unrelated parallel machines model is a theoretical abstraction of little significance to the designers of real-time systems, we believe that the uniform parallel machines model is a very relevant one for modelling many actual application systems. There are several reasons for this:

- The existence of this model gives application system designers the freedom to use processors of different speeds, rather than constraining them to always use identical processors. In fact, uniform multiprocessor platforms are already commercially available – for instance the Compaq AlphaServer GS series (specifically the series GS 160 & GS 320 – see, e.g., [8]) supports mixed processor speeds with up to 32 mixed processors.

- Even when all the processors available are identical, they may not all be exclusively available for the execution of the real-time periodic tasks, but may be required to devote a certain fraction of their computing capacity to some other (non real-time) tasks. Each such processor can be modelled by another of lower computing capacity, with this computing capacity indicative of the fraction of its actual computing capacity that can be devoted to periodic tasks.

- As new and faster processors become available, one may choose to improve the performance of a system by upgrading some of its processors. If the only model we have available is the identical multiprocessors model, we must necessarily replace all the processors simultaneously. With the uniform parallel machines model, we can however choose to replace just a few of the processors, or indeed simply add some faster processors while retaining all the previous processors.

Partitioned and global scheduling. In designing scheduling algorithms for multiprocessor environments, one can distinguish between at least two distinct approaches. In **partitioned** scheduling, all jobs generated by a task are required to execute on the *same* processor. **Global scheduling**, by contrast, permits *task migration* (i.e., different jobs of an individual task may execute upon different processors) as well as job-migration: an individual job that is preempted may resume execution upon a processor different from the one upon which it had been executing prior to preemption.

It has been proven by Leung and Whitehead [9] that the partitioned and global approaches to static-priority scheduling on identical multiprocessors are *incomparable*, in the sense that (i) there are task systems that are feasible on m identical processors under the partitioned approach but for which no priority assignment exists which would cause all jobs of all tasks to meet their deadlines under global scheduling on the same m processors; and (ii) there are task systems that are feasible on m identical processors under the global approach, but which cannot be partitioned into m distinct subsets such that each individual partition is uniprocessor static-priority feasible. This result of Leung and Whitehead [9] provides a very strong motivation to study both the partitioned and the non-partitioned approaches to static-priority multiprocessor (identical as well as uniform) scheduling, since it is provably true that neither approach is strictly better than the other.

In [2], the rate-monotonic scheduling of periodic real-time task systems upon identical multiprocessor platforms was studied. A *utilization bound* was derived such that any periodic task system with cumulative utilization no larger than this bound is guaranteed to be successfully scheduled by Algorithm RM upon an identical multiprocessor platform (this result is described in Section 2). In a sense, the results of [2] are a logical extension to the results concerning uniprocessor rate-monotonic scheduling. In this paper, we carry this logical progression one step further, by further generalizing the multiprocessor machine model. That is, we study the rate-monotonic scheduling of periodic task systems upon uniform multiprocessor platforms: multiprocessors which may be comprised of processors of different computing capacities. As in [2], we derive sufficient conditions for determining whether any periodic task system is successfully scheduled by Algorithm RM upon a given uniform multiprocessor platform.

The remainder of this paper is organized as follows. In Section 2, we provide the background necessary for understanding the results that follow. In Section 3, we derive our sufficient RM-feasibility test for periodic task systems upon uniform multiprocessors. We conclude in Section 4 with a summary of the results presented here.

2. Model and background

In this section, we (i) introduce some definitions and formal notation concerning scheduling upon uniform multiprocessor platforms, and (ii) present without proof previously-published results that will be used in this paper.

Periodic task systems. A *periodic task* $\tau_i = (C_i, T_i)$ is characterized by two parameters – an execution requirement C_i and a period T_i – with the interpretation that the task generates a *job* at each integer multiple of T_i, and each such job has an execution requirement of C_i execution units, and must complete by a deadline equal to the next integer multiple of T_i. We assume that preemption is permitted – an executing job may be interrupted, and its execution resumed later, with no loss or penalty. A periodic task system consists of several independent such periodic tasks that are to execute on a specified preemptive processor architecture. Let $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ denote a periodic task system. For each task τ_i, define its *utilization* U_i to be the ratio of τ_i's execution requirement to its period: $U_i \stackrel{\text{def}}{=} C_i/T_i$. We define the *cumulative utilization* $U(\tau)$ of periodic task system τ (often, the word "cumulative" is dropped and this is called simply the *utilization* of τ) to be the sum of the utilizations of all tasks in τ: $U(\tau) \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau} U_i$. Furthermore, we define the *maximum utilization* $U_{\max}(\tau)$ of periodic task system τ to be the largest utilization of any task in τ: $U_{\max}(\tau) \stackrel{\text{def}}{=} \max_{\tau_i \in \tau} U_i$.

Without loss of generality, we assume that $T_i \leq T_{i+1}$ for all i, $1 \leq i < n$; i.e., the tasks are indexed according to period.

In this paper, we assume that our model allows for job *preemptions* and *interprocessor migrations* for free — i.e., a job executing on a processor may be interrupted at any instant and its execution resumed later on the same or a different processor with no cost or penalty. In actual systems (particularly in distributed

systems), these assumptions are often not valid: in particular, interprocessor migration incurs a cost since the run-time state of the executing task must also be migrated to the destination processor. Nevertheless, we can easily *bound* from above the maximum number of such interprocessor migrations that any individual job will have to undergo; the total cost of all such migrations can be amortized among the individual jobs by "charging" each job for a certain number of such migrations (i.e., by inflating each job's execution requirement by an appropriate amount). In this manner, we abstract away the effects of interprocessor migrations from our formal model.

Our model forbids *job-level parallelism* – at any instant in time, each job may be executing upon at most one processor.

Uniform multiprocessors. In contrast to identical multiprocessors (in which all processors are assumed to be equally powerful), each processor in a uniform multiprocessor machine is characterized by a *speed* or computing capacity, with the interpretation that a job executing on a processor with speed s for t time units completes $(s \times t)$ units of execution. It is convenient to define some notation with respect to uniform multiprocessors.

Definition 1 Let π denote a **uniform multiprocessor platform**.

- The number of processors comprising π is denoted by $m(\pi)$.
- For all i, $1 \leq i \leq m(\pi)$, the speed (the computing capacity) of the i'th-fastest processor in π is denoted by $s_i(\pi)$, i.e., the speeds are indexed in a non-increasing manner.
- The total computing capacity of all the processors in π is denoted by $S(\pi)$: $S(\pi) \stackrel{\text{def}}{=} \sum_{i=1}^{m(\pi)} s_i(\pi)$.

■

In scheduling theory, a scheduling algorithm is said to be *work conserving* (equivalently, to *not use inserted idle time*), if it is the case that the algorithm never idles a processor while there is some active job awaiting execution which may legally execute upon this processor. With respect to uniform multiprocessor platforms, we need a somewhat stronger notion of work-conservation: not only should no processor be idled while there are jobs awaiting execution, but if there are fewer active jobs than processors available at any instant in time, then it is the slowest processors that is idled. This notion is formalized in the following definition:

Definition 2 (Greedy scheduling algorithm.) A uniform multiprocessor scheduling algorithm is said to be **greedy** if it satisfies all three of the following conditions.

1. It never idles a processor when there are jobs awaiting execution.
2. If it must idle some processor (because fewer active jobs are available than processors), then it idles the slowest processors. That is, if it is the case that at some instant t the j'th-slowest processor is idled, then the k'th-slowest processor is also idled at that instant t, for all $k > j$.
3. It always executes higher-priority jobs upon faster processors. That is, if the j'th-slowest processor is executing job J at time t, it must be the case that the priority of J is no smaller than the priorities of the jobs (if any) executing on the k'th slowest processor, for all $k > j$.

■

In the remainder of this paper, we will assume that Algorithm RM is implemented in a greedy manner upon uniform multiprocessors.

We now define two important additional parameters of uniform multiprocessor platforms; as we will see later in this paper, these parameters succinctly capture the characteristics of uniform multiprocessors that are particularly relevant in determining whether a task system is successfully scheduled upon the platform by Algorithm RM.

Definition 3 (λ and μ)

For any uniform multiprocessor platform π, we define a parameter $\underline{\lambda(\pi)}$ as follows:

$$\lambda(\pi) \stackrel{\text{def}}{=} \max_{i=1}^{m(\pi)} \left\{ \frac{\sum_{j=i+1}^{m(\pi)} s_j(\pi)}{s_i(\pi)} \right\} \qquad (1)$$

For any uniform multiprocessor platform π, we define a parameter $\underline{\mu(\pi)}$ as follows:

$$\mu(\pi) \stackrel{\text{def}}{=} \max_{i=1}^{m(\pi)} \left\{ \frac{\sum_{j=i}^{m(\pi)} s_j(\pi)}{s_i(\pi)} \right\} \qquad (2)$$

■

Parameters $\lambda(\pi)$and $\mu(\pi)$ of a uniform multiprocessor system π intuitively measure the "degree" by which π differs from an identical multiprocessor platform. More specifically, $\lambda(\pi) = (m-1)$ and $\mu(\pi) = m$ if π is comprised of m identical processors, and both become progressively smaller as the speeds of the processors differ from each other by greater amounts; in the extreme, if $s_i(\pi) >> s_{i+1}(\pi)$ for all i, then $\lambda(\pi)$ approaches zero and $\mu(\pi)$ approaches one.

Real-time job instances. At times, we find it convenient to represent a real-time system using a model that is somewhat more general than the periodic task model. We will then assume that a hard-real-time **instance** is modelled as a collection of independent jobs. Each **job** $J_j = (r_j, c_j, d_j)$ is characterized by an arrival time r_j, an execution requirement c_j, and a deadline d_j, with the interpretation that this job needs to execute for c_j units over the interval $[r_j, d_j)$. (Thus, the periodic task $\tau_i = (C_i, T_i)$ generates an infinite sequence of jobs with parameters $(k \cdot T_i, C_i, (k+1) \cdot T_i)$, $k = 0, 1, 2, \ldots$; in the remainder of this paper, we will sometimes use the symbol τ itself to denote the infinite set of jobs generated by the tasks in periodic task system τ.)

Definition 4 ($\mathbf{W(A, \pi, I, t)}$.) *Let I denote any collection of jobs, and π any uniform multiprocessor platform. For any algorithm A and time instant $t \geq 0$, let $W(A, \pi, I, t)$ denote the amount of work done by algorithm A on jobs of I over the interval $[0, t)$, while executing on π.* ■

The following theorem was proved in [7]; we will be using it later in this paper.

Theorem 1 (From [7]) Let π_o and π denote uniform multiprocessor platforms. Let A_o denote any uniform multiprocessor scheduling algorithm, and A any *greedy* uniform multiprocessor scheduling algorithm. If the following condition is satisfied by platforms π_o and π:

$$S(\pi) \geq S(\pi_o) + \lambda(\pi) \cdot s_1(\pi_o) \tag{3}$$

then for any collection of jobs I and any time-instant $t \geq 0$,

$$W(A, \pi, I, t) \geq W(A_o, \pi_o, I, t)\ . \tag{4}$$

3. RM-feasibility upon uniform processors

For any scheduling algorithm A and any processor platform π, we say that τ is **A-feasible** upon π if τ meets all deadlines when scheduled upon π by algorithm A. We say τ is **feasible** upon π if it can be scheduled to meet all deadlines upon π by an optimal algorithm. In this section, we obtain sufficient conditions for determining whether any periodic task system is RM-feasible upon any given uniform multiprocessor platform.

Consider a periodic task system τ and a uniform multiprocessor platform π, and suppose that τ and π satisfy the following relationship:

$$S(\pi) \geq 2 \cdot U(\tau) + \mu(\pi) \cdot U_{\max}(\tau) \tag{5}$$

Let us consider the RM-scheduling of τ upon π.

Without loss of generality, assume that ties are broken by Algorithm RM such that τ_i has greater priority than τ_{i+1} for all i, $1 \leq i < n$. Notice that whether jobs of τ_k meet their deadlines under Algorithm RM depends only upon the jobs generated by the tasks $\{\tau_1, \tau_2, \ldots, \tau_k\}$, and are completely unaffected by the presence of the tasks $\tau_{k+1}, \ldots, \tau_n$. For $k = 1, 2, \ldots, n$, let us define the task-set $\tau^{(k)}$ as follows:

$$\tau^{(k)} \stackrel{\text{def}}{=} \{\tau_1, \tau_2, \ldots, \tau_k\}.$$

We now present three lemmas that characterize the scheduling of periodic task system τ on uniform multiprocessors, and which lead up to our major result that any periodic task system τ satisfying Condition 5 is RM-feasible upon π. For each k, Lemma 1 below identifies a uniform multiprocessor platform π_o upon which $\tau^{(k)}$ is feasible — while it is not relevant to our purpose here, this π_o is the "minimal" (i.e., least powerful) platform upon which $\tau^{(k)}$ is feasible. Using this lemma and Theorem 1, we are able to conclude (Lemma 2) that for any π satisfying Condition 5, the total work done on jobs of $\tau^{(k)}$ by any time instant t by Algorithm RM executing upon π is at least t times the cumulative utilization of $\tau^{(k)}$ (intuitively, this means that RM while executing $\tau^{(k)}$ on platform π never "falls behind" with respect to the total amount of work done). These two lemmas together allow us to derive Lemma 3, which concludes that all jobs of $\tau^{(k)}$ will meet their deadlines in the RM-generated schedule upon π. Proofs of Lemmas 1 and 2 are provided here; however the proof of

Lemma 3 is considerably longer and is hence omitted due to space considerations. A proof of Lemma 3 may be found in the journal version of this paper [5].

Lemma 1 Task system $\tau^{(k)}$ is feasible on a uniform multiprocessor platform π_o satisfying the conditions

1. $S(\pi_o) = U(\tau^{(k)})$, and
2. $s_1(\pi_o) = U_{\max}(\tau^{(k)})$.

Proof: Set π_o equal to a uniform multiprocessor consisting of k processors, with computing capacities equal to C_1/T_1, C_2/T_2, ..., C_k/T_k respectively. It is easy to see that $\tau^{(k)}$ is feasible upon this π_o: an optimal scheduling algorithm OPT would simply schedule each task exclusively upon the processor that has computing capacity equal to that task's utilization. ■

Lemma 2 For all $k \geq 1$ and all for all $t \geq 0$,

$$W(\mathsf{RM}, \pi, \tau^{(k)}, t) \geq t \cdot \left(\sum_{j=1}^{k} U_j \right) \tag{6}$$

Proof Sketch: Recall that we are assuming that π and τ satisfy Condition 5. Since

$$\left(2 \cdot U(\tau) \geq 2 \cdot U(\tau^{(k)}) \geq U(\tau^{(k)})\right) \text{ and } (\mu(\pi) \geq \lambda(\pi)),$$

we may conclude that

$$S(\pi) \geq 2 \cdot U(\tau) + \mu(\pi) \cdot U_{\max}(\tau) \quad \text{(By Cond 5)}$$

$$\textbf{i.e.} \quad S(\pi) \geq U(\tau^{(k)}) + \lambda(\pi) \cdot U_{\max}(\tau^{(k)}) \tag{7}$$

From Inequality 7, Condition 3 and Lemma 1 above, and the fact that Algorithm RM is greedy, it follows that

$$W(\mathsf{RM}, \pi, \tau^{(k)}, t) \geq W(\text{OPT}, \pi_o, \tau^{(k)}, t)$$

where π_o and OPT are as described in the proof of Lemma 1. However,

$$W(\text{OPT}, \pi_o, \tau^{(k)}, t) = \left(\sum_{j=1}^{k} U_j \right);$$

and Inequality 6 thus follows directly from Theorem 1. ■

The proof of the following lemma may be found in [5].

Lemma 3 All jobs of τ_k meet their deadlines when $\tau^{(k)}$ is scheduled upon π using Algorithm RM.

Theorem 2 Given a periodic task system τ and a uniform multiprocessor platform π,

$$S(\pi) \geq 2 \cdot U(\tau) + \mu(\pi) \cdot U_{\max}(\tau)$$

is a sufficient condition for ensuring that τ is RM-feasible upon π.

Proof: When τ and π satisfy the condition of the theorem (which is exactly Condition 5), it follows from Lemma 3 that Algorithm RM schedules $\tau^{(k)}$ in such a manner that all jobs of the lowest-priority task τ_k complete by their deadlines. The correctness of Theorem 2 immediately follows, by induction on k. ■

We can apply Theorem 2 to identical multiprocessors to obtain a result similar to a result in [2]:

Corollary 1 Any periodic task system in which each task's utilization is no more than one-third, and the sum of the utilizations of all the tasks is no more than $m/3$, is successfully scheduled by Algorithm RM upon m unit-capacity processors.

Proof: Let τ denote the periodic task system: by the antecedents of the corollary, $U(\tau) \leq m/3$ and $U_{\max}(\tau) \leq 1/3$. Let π denote an m-processor identical multiprocessor platform comprised of unit-capacity processors; by definition,

$$\mu(\pi) \stackrel{\text{def}}{=} \max_{i=1}^{m(\pi)} \left\{ \frac{\sum_{j=i}^{m(\pi)} s_j(\pi)}{s_i(\pi)} \right\} \quad = \quad \left\{ \frac{m}{1} \right\} \quad = \quad m \,.$$

By Theorem 2, τ is RM-feasible upon π if

$$\begin{aligned} & S(\pi) \geq 2 \cdot U(\tau) + \mu(\pi) \cdot U_{\max}(\tau) \\ \equiv \quad & \left(m \geq 2 \cdot \frac{m}{3} + m \cdot \frac{1}{3} \right) \\ \equiv \quad & (m \geq m) \ , \end{aligned}$$

which is true. ■

4. Conclusions

Our contribution in this paper can be summarized as follows: *we have obtained here the first non-trivial feasibility test for a static-priority scheduling algorithm that adopts a global approach to task allocation upon uniform multiprocessors.* That is, we have studied the

behaviour of Algorithm RM — one such previously-defined [10, 2] static-priority global scheduling algorithm — upon uniform multiprocessor platforms. We have obtained simple sufficient conditions for determining whether any given periodic task system will be succesfully scheduled by Algorithm RM upon a given uniform multiprocessor platform.

References

[1] Anderson, J., and Srinivasan, A. Early release fair scheduling. In *Proceedings of the EuroMicro Conference on Real-Time Systems* (Stockholm, Sweden, June 2000), IEEE Computer Society Press, pp. 35–43.

[2] Andersson, B., Baruah, S., and Jansson, J. Static-priority scheduling on multiprocessors. In *Proceedings of the IEEE Real-Time Systems Symposium* (December 2001), IEEE Computer Society Press, pp. 193–202.

[3] Aydin, H., Mejia-Alvarez, P., Melhem, R., and Mosse, D. Optimal reward-based scheduling of periodic real-time tasks. In *Proceedings of the Real-Time Systems Symposium* (Phoenix, AZ, December 1999), IEEE Computer Society Press.

[4] Baruah, S., Cohen, N., Plaxton, G., and Varvel, D. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica 15*, 6 (June 1996), 600–625.

[5] Baruah, S., and Goossens, J. Rate-monotonic scheduling on uniform multiprocessors. *IEEE Transactions on Computers*. Accepted for publication.

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

[7] Funk, S., Goossens, J., and Baruah, S. Online scheduling on uniform multiprocessors. In *Proceedings of the IEEE Real-Time Systems Symposium* (December 2001), IEEE Computer Society Press, pp. 183–192.

[8] Gharachorloo, K., Sharma, M., Steely, S., and Doren, S. V. Architecture and design of AlphaServer GS320. *ACM SIGPLAN Notices 35*, 11 (Nov. 2000), 13–24.

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

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

[11] Moir, M., and Ramamurthy, S. Pfair scheduling of fixed and migrating tasks on multiple resources. In *Proceedings of the Real-Time Systems Symposium* (Phoenix, AZ, December 1999), IEEE Computer Society Press.

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