Real-Time Systems, 25, 187–205, 2003
© 2003 Kluwer Academic Publishers. Manufactured in The Netherlands.

Priority-Driven Scheduling of Periodic Task Systems on Multiprocessors

JOËL GOOSSENS
Department of Computer Science, Université Libre de Bruxelles, Belgium

SHELBY FUNK
Department of Computer Science, The University of North Carolina at Chapel Hill, USA

SANJOY BARUAH
Department of Computer Science, University of North Carolina, Chapel Hill, USA

Abstract. The scheduling of systems of periodic tasks upon multiprocessor platforms is considered. Utilization-based conditions are derived for determining whether a periodic task system meets all deadlines when scheduled using the earliest deadline first scheduling algorithm (EDF) upon a given multiprocessor platform. A new priority-driven algorithm is proposed for scheduling periodic task systems upon multiprocessor platforms: this algorithm is shown to successfully schedule some task systems for which EDF may fail to meet all deadlines.

Keywords: multiprocessor scheduling, periodic tasks, earliest deadline first

1. Introduction

Over the years, the preemptive periodic task model (Liu, 1969; Liu and Layland, 1973) has proven remarkably useful for the modelling of recurring processes that occur in hard-real-time computer application systems. In this model, 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 execution by a deadline equal to the next integer multiple of T_i. A periodic task system consists of several such periodic tasks that are to execute on a specified processor architecture. The jobs are assumed to be independent in the sense that each job does not interact in any manner (accessing shared data, exchanging messages, etc.) with other jobs of the same or another task. It is also assumed that the model allows for job preemption; i.e., a job executing on a processor may be preempted prior to completing execution, and its execution may be resumed later, at no cost or penalty.

The real-time scheduling of periodic task systems has been much studied. In the uniprocessor context—when there is exactly one shared processor available upon which to execute all the jobs generated by all the tasks in the system—it is known that the earliest deadline first scheduling algorithm (Algorithm EDF) (Liu and Layland, 1973; Dertouzos, 1974), which executes at each instant in time the currently active[1] job whose deadline parameter is the smallest, is an optimal scheduling algorithm in the sense that if a system can be scheduled such that all deadlines can be met, then Algorithm EDF will schedule this system to meet all deadlines. The problem of scheduling such periodic task systems on identical multiprocessor platforms—when there are several identical processors available upon which the jobs generated by the periodic tasks are to execute (with the constraint that an individual job may execute on either zero or one processor at any instant in time)—was first posed by Liu in a seminal paper (Liu, 1969) in 1969. In this paper, Liu identified a set of properties for periodic task systems which are sufficient (albeit not necessary) to guarantee feasibility upon an m-processor identical multiprocessor platform; i.e., any periodic task system satisfying these properties can always be scheduled upon an m-processor identical multiprocessor platform to meet all deadlines. Several other multiprocessor algorithms for scheduling periodic task systems have been proposed (see, for example, Leung, 1989; Burchard et al., 1995); however, none of these algorithms were optimal in the sense of successfully scheduling all feasible periodic task systems. Then in Baruah et al. (1996) presented necessary and sufficient feasibility conditions, and an optimal scheduling algorithm, based upon the notion of pfair scheduling.

1.1. Pfair Scheduling

Pfair scheduling was proposed as a way of optimally and efficiently scheduling periodic tasks on a multiprocessor system. Pfair scheduling differs from more conventional real-time scheduling disciplines in that tasks are explicitly required to make progress at steady rates. In the periodic task model, each task $\tau_i = (C_i, P_i)$ executes at an implicit rate given by C_i/P_i. However, this notion of a rate is a bit inexact: a job of τ_i may be allocated C_i time units at the beginning of its period, or at the end of its period, or its computation may be spread out more evenly. Under pfair scheduling, this implicit notion of a rate is strengthened to require each task to be executed at a rate that is uniform across each job. Pfair scheduling algorithms ensure uniform execution rates by breaking jobs into smaller subjobs. Each subjob must execute within a window of time, the end of which acts as its pseudodeadline. These windows divide each period of a task into subintervals of approximately equal length. By breaking tasks into smaller executable units, pfair scheduling algorithms circumvent many of the bin-packing-like problems that lie at the heart of intractability results involving multiple-resource real-time scheduling problems. Intuitively, it is easier to evenly distribute small, uniform items among the available bins than larger, non-uniform items.

However, this very feature of pfair scheduling algorithms can also prove a disadvantage in certain implementations—one consequence of ''breaking'' each job of each task into subjobs, and making individual scheduling decisions for each subjob, is that jobs tend to get preempted after each of their constituent sub-jobs completes execution. As a result, pfair schedules are likely to contain a large number of job preemptions and context-switches. For some applications, this is not an issue; for others, however, the overhead resulting from too many preemptions may prove unacceptable. Pfair scheduling is not the appropriate scheduling approach for such application systems.

1.2. Priority-Driven Scheduling

Run-time scheduling is essentially 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 job, and allocate the available processors to the highest-priority jobs.

Different scheduling algorithms differ from one another in the manner in which priorities get assigned to individual jobs by the algorithms. Some scheduling algorithms are observed to have certain desirable features in terms of ease (and efficiency) of implementation, particularly upon multiprocessor platforms. Some of the important characteristics of such algorithms were studied by Ha and Liu (1993, 1994) and Ha (1995), who proposed the following definition:

DEFINITION 1 (*Priority-driven algorithms, Ha and Liu 1994). A scheduling algorithm is said to be a priority driven scheduling algorithm if and only if it satisfies the condition that for every pair of jobs J_i and J_j, if J_i has higher priority than J_j at some instant in time, then J_i always has higher priority than J_j.*

By this definition, Algorithm EDF is a priority-driven algorithm while Algorithm PF is not.

From an implementation perspective, there are significant advantages to using priority-driven algorithms in real-time systems; while it is beyond the scope of this document to describe in detail all these advantages, some important ones are listed below.

- Very efficient implementations of priority-driven scheduling algorithms have been designed (see, for example, Mok, 1988).
- It can be shown that when a set of jobs is scheduled using a priority-driven algorithm then the total number of preemptions is bounded from above by the number of jobs in the set (and consequently, the total number of context switches is bounded at twice the number of jobs).
- It can similarly be shown that the total number of interprocessor migrations of individual jobs is bounded from above by the number of jobs.

1.3. This Research

The earliest deadline first scheduling algorithm (Algorithm EDF) is one of the most popular scheduling algorithms used in research about real-time systems. In this paper, we first study the EDF scheduling of periodic task systems upon identical multiprocessors—we provide tight utilization-based conditions (Theorems 5 and 6) for determining whether a particular periodic task system can be successfully scheduled by EDF upon a given multiprocessor platform. More generally, however, we believe that most features of EDF-scheduling that make it such a popular scheduling algorithm—efficient implementations, bounded preemptions and inter-processor migrations, etc.—are not unique to EDF, but instead hold for all priority-driven scheduling algorithms. Therefore, we propose a variant of the EDF scheduling algorithm that falls within the framework of priority-driven algorithms, and which is provably superior to EDF in the sense that it schedules all periodic task systems that EDF can schedule, and in addition schedules some periodic task systems for which EDF may miss some deadlines.

1.4. Organization

The remainder of this paper is organized as follows. In Section 2, we review some prior results from real-time scheduling theory that we will be using in later sections. In Section 3, we apply resource-augmentation techniques (Kalyanasundaram and Pruhs, 1995; Phillips et al., 1997) to develop theoretical results that allow us to relate the feasibility (the existence of schedules) and EDF-schedulability (the ability of EDF to successfully meet all deadlines) of periodic task systems—while feasibility is equivalent to EDF-schedulability in uniprocessor systems, this is not the case upon multiprocessors. In Section 4, we apply the theory developed in Section 3 to obtain utilization-based EDF-schedulability bounds for periodic task systems upon multiprocessors; furthermore, we prove that these bounds are tight. In Section 5, we propose and analyze a new priority-driven scheduling algorithm for scheduling periodic task systems upon multiprocessor platforms which is provably superior to EDF. In Section 6, we summarize the results presented in this paper.

2. Background

We briefly describe below some results in multiprocessor real-time scheduling theory that will be used in the remainder of this paper.

2.1. Predictable Scheduling Algorithms

Ha and Liu (1993, 1994; Ha, 1995) have studied the issue of predictability in the multiprocessor scheduling of real-time systems from the following perspective.

Let us define a job $J_j = (r_j, e_j, d_j)$ as being characterized by an arrival time r_j, an execution requirement e_j, and a deadline d_j, with the interpretation that this job needs to execute for e_j units over the interval $[r_j, d_j)$.

DEFINITION 2 (*Predictability*) *Let A denote a scheduling algorithm, and $I = \{J_1, J_2, \ldots, J_n\}$ any set of n jobs, $J_j = (r_j, e_j, d_j)$. Let f_j denote the time at which job J_j completes execution when I is scheduled using algorithm A.*

Now, consider any set $I' = \{J'_1, J'_2, \ldots, J'_n\}$ of n jobs obtained from I as follows. Job J'_j has an arrival time r_j, an execution requirement $e'_j \le e_j$, and a deadline d_j (i.e., job J'_j has the same arrival time and deadline as J_j, and an execution requirement no larger than

J_j's). Let f_j' denote the time at which job J_j completes execution when I is scheduled using algorithm A. Scheduling algorithm A is said to be predictable if and only if for any set of jobs I and for any such I' obtained from *I, it is the case that $f_j' \leq f_j$ for all j.*

Informally, Definition 2 recognizes the fact that the specified execution-requirement parameters of jobs are typically only upper bounds on the actual execution-requirements during run-time, rather than the exact values. For a predictable scheduling algorithm, one may determine an upper bound on the completion-times of jobs by analyzing the situation under the assumption that each job executes for an amount equal to the upper bound on its execution requirement; it is guaranteed that the actual completion time of jobs will be no later than this determined value.

Since a periodic task system generates a set of jobs, Definition 2 may be extended in a straightforward manner to algorithms for scheduling periodic task systems: an algorithm for scheduling periodic task systems is predictable iff for any periodic task systems $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ it is the case that the completion time of each job when every job of τ_i has an execution requirement exactly equal to C_i is an upper bound on the completion time of that job when every job of τ_i has an execution requirement of at most C_i, for all i, $1 \leq i \leq n$.

The result from the work of Ha and Liu (1993, 1994) and (Ha, 1995) that we will be using can be stated as follows.

THEOREM 1 *(Ha and Liu) Any preemptive priority-driven scheduling algorithm is predictable.*

2.2. Scheduling Periodic Task Systems on Uniform Multiprocessors

Although our goal in this research is to study the scheduling of periodic task systems upon identical multiprocessor platforms—multiprocessor machines in which all the processors are identical—we find it useful to introduce a more general model of multiprocessor machines—the uniform multiprocessor platform.

DEFINITION 3 (*Uniform multiprocessors*) *A uniform multiprocessor platform is comprised of several processors. Each processor P is characterized by a single parameter—a speed (or computing capacity)* $\mathsf{speed}(P)$, *with the interpretation that a job that executes on processor P for t time units completes* $\mathsf{speed}(P) \times t$ *units of execution*[2].

Let π denote a uniform multiprocessor platform. We introduce the following notation:

$$s_\pi \stackrel{\text{def}}{=} \max_{P \in \pi} \{\mathsf{speed}(P)\}$$

$$S_\pi \stackrel{\text{def}}{=} \sum_{P \in \pi} \mathsf{speed}(P)$$

That is, s_π denotes the computing capacity of the fastest processor in π, and S_π the total computing capacity of all the processors in π.

The problem of determining whether a particular periodic task system is feasible on a given uniform multiprocessor platform has been studied (see, for example, Funk et al., 2001). The following theorem was proved in Funk et al. (2001).

THEOREM 2 *Let τ denote a periodic task system. There is a uniform multiprocessor platform π upon which τ is feasible, which satisfies the following two properties:*

- *The fastest processor in π has computing capacity equal to the largest utilization of any task in τ, i.e.,*
$$s_\pi = \max_{\tau_i \in \tau} \left\{ \frac{C_i}{T_i} \right\} \tag{1}$$
- *The cumulative computing capacity of π is equal to the utilization of τ, i.e.,*
$$S_\pi = \sum_{\tau_i \in \tau} \frac{C_i}{T_i} \tag{2}$$

3. EDF-scheduling on Identical Multiprocessors

In this section, we develop a theoretical framework that permits us to relate the feasibility of a real-time system upon a particular multiprocessor platform to its EDF-schedulability upon a different real-time platform. This framework is more general than is needed for our purposes—while we are interested in periodic task systems only, the results we derive here hold for any arbitrary collection of jobs (and not just those generated by periodic tasks); also, these jobs need not all be known beforehand to EDF, but can be revealed on-line.

In the context of uniprocessor scheduling, a work-conserving scheduling algorithm is defined to be one that never idles the (single) processor while there is any active job awaiting execution. This definition extends in a rather straightforward manner to the identical multiprocessor case:

DEFINITION 4 (*Work-conserving scheduling algorithms*) *An algorithm for scheduling on identical multiprocessors is defined to be work-conserving if it never leaves any processor idle while there remain active jobs awaiting execution.*

Note that EDF is a work-conserving algorithm by this definition.

Some additional notation:

DEFINITION 5 ($W(A, \pi, I, t)$) *Let I denote any set 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.

Lemma 1 below specifies a condition (Condition (3)) upon any uniform multiprocessor platform π and any identical platform π' under which any work-conserving algorithm A' (such as EDF) executing on π' is guaranteed to complete at least as much work by each instant in time t as any other algorithm A (including an optimal algorithm) executing on π, when both algorithms are executing on any set of jobs I.

Lemma 1 *Let π denote a uniform multiprocessor platform with cumulative processor-capacity S_π, and in which the fastest processor has computing capacity s_π, $s_\pi < 1$. Let π' denote an identical multiprocessor platform comprised of m' unit-capacity processors. Let A denote any uniform multiprocessor scheduling algorithm, and A' any work-conserving m'-processor identical multiprocessor scheduling algorithm. If the following condition is satisfied:*

$$m' \geq \frac{S_\pi - s_\pi}{1 - s_\pi} \tag{3}$$

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

$$W(A', \pi', I, t) \geq W(A, \pi, I, t) \tag{4}$$

Proof: The proof is by contradiction. Suppose then that it is not true; i.e., there is some time-instant by which a work-conserving algorithm A' executing on π' has performed strictly less work than some other algorithm A executing on π. Let $J_j \in I$ denote a job with the earliest arrival time such that there is some time-instant t_0 satisfying

$$W(A', \pi', I, t_0) < W(A, \pi, I, t_0)$$

and the amount of work done on job J_j by time-instant t_0 in A' is strictly less than the amount of work done on J_j by time-instant t_0 in A.

By our choice of r_j, it must be the case that

$$W(A', \pi', I, r_j) \geq W(A, \pi, I, r_j)$$

Therefore, the amount of work done by A over $[r_j, t_0)$ is strictly more than the amount of work done by A' over the same interval.

Let x denote the cumulative length of time over the interval $[r_j, t_0)$ during which A' is executing on all m' processors; let $y \overset{\text{def}}{=} ((t_0 - r_j) - x)$ denote the length of time over this interval during which A' idles some processor.

We make the following two observations.

- Since A' is a work-conserving scheduling algorithm, job J_j, which has not completed by instant t_0 in the schedule generated by A', must have executed for at least y time units by time t_0 in the schedule generated by A'; while it could have executed for at most $(x + y)$ time units in the schedule generated by A; therefore,

$$(x + y) \cdot s_\pi > y \tag{5}$$

- The amount of work done by A' over $[r_j, t_0)$ is at least

  $(m'x + y)$

  while the amount of work done by A over this interval is at most

  $S_\pi \cdot (x + y)$

  therefore, it must be the case that

  $$S_\pi \cdot (x + y) > (m'x + y) \tag{6}$$

  Adding $(m' - 1)$ times Inequality 5 to Inequality 6, we get

  $$\begin{aligned}&(m' - 1)(x + y) \cdot s_\pi + S_\pi(x + y) > (m' - 1) \cdot y + (m'x + y)\\ &\quad \equiv ((m' - 1)s_\pi + S_\pi) \cdot (x + y) > m' \cdot (x + y)\\ &\quad \equiv (m' - 1)s_\pi + S_\pi > m'\\ &\quad \equiv S_\pi - s_\pi > m' - m's_\pi\\ &\quad \equiv \frac{S_\pi - s_\pi}{1 - s_\pi} > m'\end{aligned}$$

  which is a contradiction of Condition (3). ■

The following theorem applies Lemma 1 to the case where the work-conserving algorithm A' of Lemma 1 is Algorithm EDF, and algorithm A of Lemma 1 is an optimal (offline) scheduler.

THEOREM 3 *Let π denote an m-processor uniform multiprocessor platform with cumulative processor-capacity S_π, and in which the fastest processor has computing capacity s_π, $s_\pi < 1$. Let I denote an instance of jobs that is feasible on π. Let π' denote an identical multiprocessor platform comprised of m' unit-capacity processors. If Condition (3) of Lemma 1 is satisfied, then I will meet all deadlines when scheduled using the* EDF *algorithm executing on π'.*

Proof: As a consequence of π and π' satisfying Condition (3), it follows from Lemma 1 that the work done at any time-instant t by EDF scheduling I on π' is at least as much as the work done by that time-instant t by an optimal scheduling algorithm executing I on π:

$$W(\text{EDF}, \pi', I, t) \geq W(\text{OPT}, \pi, I, t) \quad \text{for all } t \geq 0$$

where OPT denotes an algorithm that generates a schedule for I which meets all deadlines on π—since I is assumed feasible on π, such a schedule exists.

We now prove by induction that I is scheduled by EDF to meet all deadlines on π'. The induction is on the number of jobs in I. Specifically, let $I_k \overset{\text{def}}{=} \{J_1, \ldots, J_k\}$ denote the k jobs of I with the highest EDF-priority.

Base case. Since I_0 denotes the empty set, I_0 can clearly be scheduled by EDF to meet all deadlines on π'.

Induction step. Assume that EDF can schedule I_k on π' for some k and consider the EDF-generated schedule of I_{k+1} on π'. Note that $I_k \subset I_{k+1}$ and that the J_{k+1} does not effect the scheduling decisions made by EDF on the jobs $\{J_1, J_2, \ldots, J_k\}$ while it is scheduling I_{k+1}. That is, the schedule generated by EDF for $\{J_1, J_2, \ldots, J_k\}$ while scheduling I_{k+1}, is identical to the schedule generated by EDF while scheduling I_k; hence by the induction hypothesis, these k highest priority jobs $\{J_1, J_2, \ldots, J_k\}$ of I_{k+1} all meet their deadlines. It remains to prove that J_{k+1} also meets its deadline.

Let us now turn our attention to the schedules generated by OPT executing on π. Since I is assumed to be feasible on π, it follows that I_{k+1} is also feasible on π and hence OPT will schedule I_{k+1} on π to meet all deadlines. That is,

$$W(\text{OPT}, \pi, I_{k+1}, d_{k+1}) = \sum_{i=1}^{k+1} c_i$$

where d_{k+1} denotes the latest deadline of a job in I_{k+1}. By Lemma 1

$$W(\text{EDF}, \pi', I_{k+1}, d_{k+1}) \geq W(\text{OPT}, \pi, I_{k+1}, d_{k+1}) = \sum_{i=1}^{k+1} c_i$$

Since the total execution requirement of all the jobs in I_{k+1} is $\Sigma_{i=1}^{k+1} c_i$ it follows that job J_{k+1} meets its deadline.

We have thus shown that EDF successfully schedules all the jobs of I_{k+1} to meet their deadlines on π'. The theorem follows. ■

4. EDF-Scheduling of Periodic Task Systems

In this section and the next, we apply the theory developed in Section 3 above to study the priority-driven scheduling of periodic task systems on identical multiprocessor platforms. In this section, we study the EDF-scheduling of periodic task systems on multiprocessor platforms; in Section 5, we consider the problem of scheduling periodic task systems upon multiprocessor platforms when we are not restricted to using EDF, but rather may use any priority-driven algorithm.

The results of Section 3 are applicable to on-line scheduling—the characteristics of jobs need not be known prior to their arrival times. Although scheduling a periodic task system is not an on-line problem in the sense that all task parameters are assumed known beforehand, the results in Section 3 nevertheless turn out to be useful towards developing a framework for scheduling periodic task systems on multiprocessors.

Let $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ denote a periodic task system. Let $u_i \overset{\text{def}}{=} (C_i/T_i)$ denote the utilization of task T_i for each i, $1 \leq i \leq n$, and let $U(\tau) \overset{\text{def}}{=} \Sigma_{i=1}^{n} u_i$ denote the utilization of task system τ. We require that $u_i \leq 1$ for all i, $1 \leq i \leq n$. Without loss of generality, we assume that tasks are indexed according to non-increasing utilization; i.e., $u_i \geq u_{i+1}$ for all i, $1 \leq i \leq n$. We introduce the notation $\tau^{(i)}$ to refer to the task system comprised of the $(n - \mathrm{i} + 1)$ minimum-utilization tasks in τ:

$$\tau^{(i)} \overset{\text{def}}{=} \{\tau_i, \tau_{i+1}, \ldots, \tau_n\}$$

(According to this notation, $\tau \equiv \tau^{(1)}$.)

By a direct application of Theorems 2 and 3, we obtain below a sufficient condition for a periodic task system to be successfully scheduled by EDF. By Theorem 2, periodic task system τ is feasible on some uniform multiprocessor platform π with cumulative computing capacity $S_\pi = U(\tau)$, in which the fastest processor has speed $s_\pi = u_1$. Hence by Theorem 3, we obtain the following theorem:

THEOREM 4 *Periodic task system τ can be* EDF*-scheduled upon an identical multiprocessor platform comprised of m unit-capacity processors, provided*

$$m \geq \left\lceil \frac{U(\tau) - u_1}{1 - u_1} \right\rceil \tag{7}$$

(Note that, as $u_1 \to 1$, the right-hand side of Inequality 7 approaches ∞. However, the number of processors needed for EDF to successfully schedule τ cannot exceed the number of tasks n; hence the right-hand side of Inequality 7 could be replaced by $\min(n, \lceil (U(\tau) - u_1/1 - u_1) \rceil)$. For reasons of algebraic simplicity, we do not make this explicit in the remainder of this paper.)

Theorem 5 follows by algebraic simplification of Equation (7):

THEOREM 5 *Periodic task system τ can be* EDF*-scheduled upon m unit-speed identical processors, provided its cumulative utilization is bounded from above as follows:*

$$U(\tau) \leq m - u_1 \cdot (m - 1) \tag{8}$$

It turns out that the bounds of Theorem 4 and 5 are in fact tight:

THEOREM 6 *Let m denote any positive integer > 1, u_1 any real number satisfying $0 < u_1 < 1$, and ε an arbitrarily small positive real number, $\varepsilon \ll u_1$.* EDF *cannot schedule some periodic task systems with cumulative utilization $m - u_1(m - 1) + \varepsilon$ in which the largest-utilization task has utilization equal to u_1, upon m unit-speed processors.*

Proof: Let P denote some positive number. We construct a task set τ as follows.

1. Task τ_1 has execution requirement $C_1 = u_1 \cdot p$ and period $T_1 = p$.

2. Tasks $\tau_2, \tau_2, \ldots, \tau_n$ all have period $T_i = p$ and execution requirement $C_i = C$ for all i, $1 < i \leq n$, satisfying

   $$(n - 1) \cdot C = m \cdot (1 - u_1) \cdot p + m\delta$$

   where $\delta = (p \cdot \varepsilon)/m$. Furthermore, n is chosen such that $n - 1$ is a multiple of m, and is large enough so that $C \ll u_1 \cdot p$.

The largest-utilization task in τ is τ_1, which has utilization equal to $(u_1 \cdot p)/p = u_1$. The cumulative utilization of tasks in τ is given by

$$\begin{aligned} U(\tau) &= \frac{u_1 \cdot p}{p} + \frac{C_2}{p} + \frac{C_3}{p} + \cdots + \frac{C_n}{p} \\ &= u_1 + \frac{(n-1) \cdot C}{p} \\ &= u_1 + m \cdot (1 - u_1) + m\frac{\delta}{p} \\ &= m - u_1 \cdot (m-1) + \varepsilon \end{aligned}$$

Now consider the scheduling of the first jobs of each task, and suppose that EDF breaks ties such that τ_1's first job is selected last for execution.[3] Then EDF schedules the jobs of tasks $\tau_2, \tau_3, \ldots$ before scheduling τ_1's job; these jobs of $\tau_2, \tau_3, \ldots$ consume all m processors over the interval $[0, (1-u_1) \cdot p + \delta)$, and τ_1's job can only begin execution at time-instant $(1-u_1) \cdot p + \delta$. Therefore τ_1's job's completion time is $((1-u_1) \cdot p + \delta + u_1 \cdot p) = (p + \delta)$, and it misses its deadline. Thus, the τ we have constructed above is a periodic task system with utilization $U(\tau) = m - u_1(m-1) + \varepsilon$, which EDF fails to successfully schedule upon m unit-speed processors. The theorem follows. ■

Phillips et al. (1997) had proved that any instance of jobs feasible upon m unit-capacity multiprocessors can be EDF-scheduled upon m processors each of capacity $(2 - (1/\mathrm{m}))$. For periodic task systems, we see below (Theorem 7) that this follows as a direct consequence of the results above.

Lemma 2 *Any periodic task system* $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ *satisfying*

1. $u_i \leq m/(2m-1)$ *for all* i, $1 \leq i \leq n$, *and*

2. $U(\tau) \leq m^2/(2m-1)$

will be scheduled by Algorithm EDF *to meet all deadlines on m unit-capacity processors.*

Proof sketch: By Theorem 4, τ can be EDF-scheduled on $\lceil (U(\tau) - u_1)/(1-u_1) \rceil$ unit-capacity processors; by substituting for $U(\tau)$ and u_1, we obtain

$$\begin{aligned} \left\lceil \frac{U(\tau) - u_1}{1 - u_1} \right\rceil &= \left\lceil \frac{m^2/(2m-1) - m/(2m-1)}{1 - m/(2m-1)} \right\rceil = \left\lceil \frac{m/(2m-1) \cdot (m-1)}{(2m-1-m)/2m-1} \right\rceil \\ &= \left\lceil \frac{m(m-1)}{(m-1)} \right\rceil = \lceil m \rceil = m \end{aligned}$$ ■

THEOREM 7 *Any periodic task system that is feasible upon m unit-capacity processors will be scheduled by Algorithm* EDF *to meet all deadlines on* $m(2 - 1/m)$*-capacity processors.*

Proof sketch: Suppose that periodic task system $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ is feasible upon m unit-capacity processors. It must be the case that

1. $u_i \leq 1$ for all i, $1 \leq i \leq n$—i.e., no individual task needs more than an entire processor, and

2. $U(\tau) \leq m$—i.e., the cumulative computing requirement of τ does not exceed the capacity of the platform.

The theorem now follows directly from Lemma 2, by scaling all utilizations and processor-speeds by a factor of $(2 - 1/m)$. ■

5. Priority-Driven Scheduling of Periodic Task Systems

If we are not tied to using EDF, but can instead use any priority-driven scheduling algorithm, we can often schedule a periodic task system τ upon fewer than the $\lceil (U \times (\tau) - u_1)/(1 - u_1) \rceil$ processors mandated by Theorems 4 and 6. Recall that tasks in τ are indexed according to non-increasing utilization (i.e., $u_i \geq u_{i+1}$ for all i, $1 i < n$), and consider the following priority-driven scheduling algorithm:

Algorithm $\mathsf{EDF}^{(k)}$ assigns priorities to jobs of tasks in τ according to the following rule:
For all $i < k$, τ_i's jobs are assigned highest priority (ties broken arbitrarily)—this is trivially achieved within an EDF implementation by setting all deadlines of τ_i equal to $-\infty$.
For all $i \geq k$, τ_i's jobs are assigned priorities according to EDF.

That is, Algorithm $\mathsf{EDF}^{(k)}$ assigns highest priority to jobs generated by the $k - 1$ tasks in τ that have highest utilizations, and assigns priorities according to deadline to jobs generated by all other tasks in τ. (Thus, ''pure'' EDF is $\mathsf{EDF}^{(1)}$.)

THEOREM 8 *Periodic task system τ will be scheduled to meet all deadlines on m unit-speed processors by Algorithm $\mathsf{EDF}^{(k)}$, where*

$$m = (k-1) + \left\lceil \frac{U(\tau^{(k+1)})}{1 - u_k} \right\rceil \tag{9}$$

Proof: By Theorem 2, $\tau^{(k)}$ is feasible on some uniform multiprocessor platform with cumulative computing capacity $U(\tau^{(k)})$, in which the fastest processor has speed u_k. Hence, by Theorem 3, $\tau^{(k)}$ can be EDF-scheduled upon an identical multiprocessor platform comprised of $\hat{m}$ unit-capacity processors, where

$$\hat{m} \stackrel{\text{def}}{=} \left\lceil \frac{U(\tau^{(k+1)})}{1 - u_k} \right\rceil$$

It follows from the definition of m (Equation (9)) that

$$m = (k-1) + \hat{m}$$

Now, consider the task system $\tilde{\tau}$ obtained from τ by replacing each task in $(\tau \setminus \tau^{(k)})$ by a task with the same period, but with utilization equal to one:

$$\tilde{\tau} \overset{\text{def}}{=} \bigcup_{j=1}^{k-1} \{(T_j, T_j)\} \quad \bigcup \tau^{(k)}$$

Let us consider the scheduling of $\tilde{\tau}$ by Algorithm $\mathsf{EDF}^{(k)}$, on m unit-capacity processors (where m is as defined in Equation (9)). Notice that Algorithm $\mathsf{EDF}^{(k)}$ will assign identical priorities to corresponding tasks in τ and $\tilde{\tau}$ (where the notion of ''corresponding'' is defined in the obvious manner). Also notice that when scheduling $\tilde{\tau}$, Algorithm $\mathsf{EDF}^{(k)}$ will devote $(k-1)$ processors exclusively to the $(k-1)$ tasks that generate jobs of highest priority (since each has a utilization equal to unity) and will be executing EDF on the jobs generated by the remaining tasks (the tasks in $\tau^{(k)}$) upon the remaining $\hat{m}$ processors. As we have seen above, Algorithm EDF schedules the tasks in $\tau^{(k)}$ upon $\hat{m}$ processors to meet all deadlines; hence, Algorithm $\mathsf{EDF}^{(k)}$ schedules $\tilde{\tau}$ upon m processors to meet all deadlines of all jobs.

Finally, notice that an execution of Algorithm $\mathsf{EDF}^{(k)}$ upon m processors on task system τ can be considered to be an instantiation of a run of Algorithm $\mathsf{EDF}^{(k)}$ upon m processors on task system $\tilde{\tau}$ in which some jobs—the ones generated by tasks whose jobs are assigned highest priority—do not execute to their full execution requirement. Since Algorithm $\mathsf{EDF}^{(k)}$ is a predictable scheduling algorithm, it follows by the result of Ha and Liu (Theorem 1) that each job of each task during the execution of Algorithm $\mathsf{EDF}^{(k)}$ on task system τ completes no later than the corresponding job during the execution of Algorithm $\mathsf{EDF}^{(k)}$ on task system $\tilde{\tau}$. And, we have already seen above that no deadlines are missed during the execution of Algorithm $\mathsf{EDF}^{(k)}$ on task system $\tilde{\tau}$. ■

By Theorem 4, $\lceil (U(\tau) - u_1)/(1 - u_1) \rceil$ unit-capacity processors suffice to guarantee that all deadlines of periodic task system τ are met, if τ is scheduled using EDF. As the following corollary states, we can often make do with fewer than $\lceil (U(\tau) - u_1)/(1 - u_1) \rceil$ processors if we are not restricted to using the EDF scheduling algorithm, but may instead choose one of the priority-driven algorithms Algorithm $\mathsf{EDF}^{(k)}$, for some k, $1 \le k < n$.

Corollary 1 Periodic task system τ will be scheduled to meet all deadlines on

$$m_{\min}(\tau) \overset{\text{def}}{=} \min_{k=1}^{n} \left\{ (k-1) + \left\lceil \frac{U(\tau^{(k+1)})}{1 - u_k} \right\rceil \right\} \tag{10}$$

unit-capacity processors by a priority-driven scheduling algorithm.

Proof: Let $k_{\min}(\tau)$ denote the smallest value of k that minimizes the right-hand side of Equation (10):

$$m_{\min}(\tau) \equiv (k_{\min}(\tau) - 1) + \left\lceil \frac{U(\tau^{(k_{\min}(\tau)+1)})}{1 - u_{k_{\min}(\tau)}} \right\rceil$$

It follows directly from Theorem 8 that τ can be scheduled to meet all deadlines upon $m_{\min}(\tau)$ unit-speed processors by the priority-driven algorithm Algorithm $\mathsf{EDF}^{(k_{\min}(\tau))}$. ■

Algorithm PriD. Based upon Corollary 1 above, we propose the following priority-driven scheduling algorithm for scheduling periodic task systems upon identical multiprocessors: Given a periodic task system $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ with $u_i \leq u_{i+1}$ for all i, $1 \leq i > n$, Algorithm PriD computes $m_{\min}(\tau)$ according to Equation (10), and schedules τ by Algorithm $\mathsf{EDF}^{(k_{\min}(\tau))}$.

Example 1 Consider a task system τ comprised of five tasks:

$$\tau = \{(9, 10), (14, 19), (1, 3), (2, 7), (1, 5)\}$$

for this system, $u_1 = 0.9$, $u_2 = 14/19 \approx 0.737$, $u_3 = 1/3$, $u_4 = 2/7 \approx 0.286$, and $u_5 = 0.2$; $U(\tau)$ consequently equals ≈ 2.457.

It may be verified that for this task system, the right-hand side of Equation (10) is minimized for $k = 3$; hence, $k_{\min}(\tau) = 3$ and $m_{\min}(\tau)$ equals

$$\begin{aligned}(3-1) &+ \left\lceil \frac{0.286 + 0.2}{1 - 0.334} \right\rceil \\ &= 2 + \left\lceil \frac{0.486}{0.667} \right\rceil \\ &= 3\end{aligned}$$

That is, τ can be scheduled to meet all deadlines by Algorithm $\mathsf{EDF}^{(3)}$ on 3 processors.

By contrast, Theorem 4 can only guarantee that all deadlines will be met upon $\lceil (U(\tau) - u_1)/(1 - u_1) \rceil \approx \lceil 1.557/0.1 \rceil = 16$ processors, if τ were scheduled using EDF.

It is noteworthy that with respect to runtime complexity (and hence with respect to scheduling overhead, scalability, etc.) the behavior of Algorithm PriD is identical to that of EDF; therefore, all the positive characteristics of EDF in these respects continue to hold for Algorithm PriD as well.

6. Experimental Evaluation

Above, we proposed a new priority-driven scheduling algorithm—Algorithm PriD—and proved that this algorithm often makes better use of available computing resources than ‘‘pure’’ EDF. We now experimentally evaluate Algorithm PriD and compare its performance with that of EDF.

In our experiments we shall study our technique based on randomly chosen systems. We are cognizant that it is in general very difficult to draw accurate conclusions regarding the benefits of a proposed technique from ‘‘simulations’’, since these benefits often depend in a non-obvious way upon the many parameters of the real-time system—in particular on the (distribution of the) system characteristics (the number of tasks, the load of the system, etc.). It is of course not possible to consider all distributions of real-time systems in our simulations; moreover, it is difficult to determine which distributions are reasonable, and which are not. For some of our simulation experiments, we have therefore made use of the pseudo-random task set generator developed by Ripoll et al. (1996) for evaluating a feasibility-analysis algorithm, which they have very generously made available to us. Workloads generated by the Ripoll et al. generator have been widely used for experimentally evaluating real-time scheduling algorithms, and these experiments have been revealed to the larger research community for several years now. We believe that using this task generator provides a context for our simulation results, and allows them to be compared with other results performed by other researchers.

We use the pseudo-random periodic task set generator proposed by Ripoll and colleagues, with the same parameters as in Ripoll et al. (1996) except the utilization factor of the system which is uniformly drawn from interval $[1, 10]$, the computation times are uniformly chosen from the interval $[1, 20]$, the deadlines from the interval $[2, 170]$, and the periods from the interval $[3, 670]$. Figure 1 shows the average (i.e., the arithmetic mean) number of processors needed by Algorithms PriD and EDF as a function of total utilization $U(\tau)$.

Mixed systems. The experiments described above indicate that Algorithm PriD tends to require fewer processors than pure EDF in general, in order to schedule a given periodic task system. The benefits of Algorithm PriD seem to be even more significant if the utilizations of the tasks are less homogeneous than is the case for task systems generated

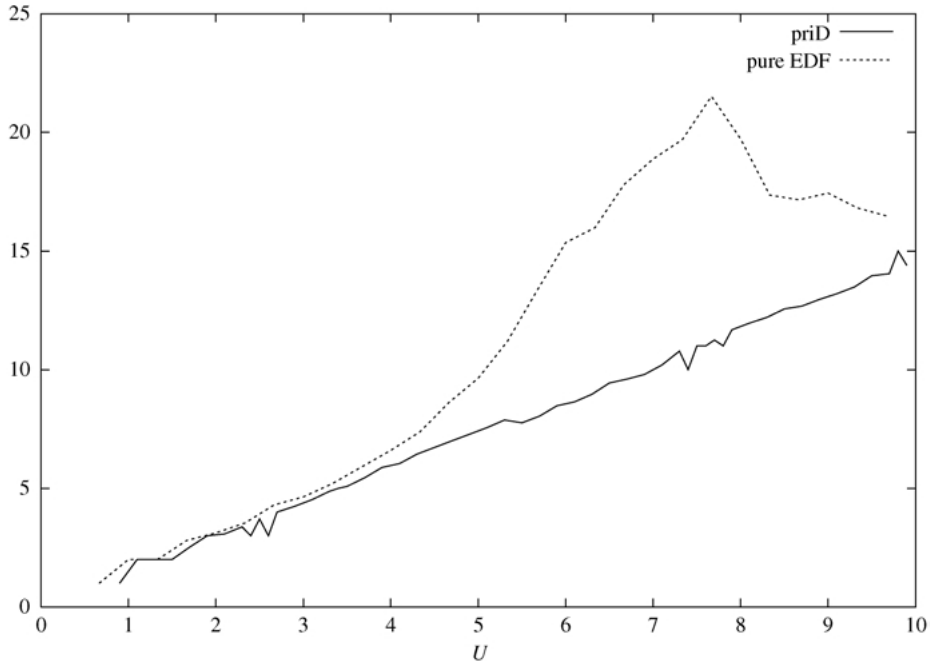

Figure 1. Average number of processors needed, as a function of total utilization $U(\tau)$.

by the Ripoll et al. generator (Ripoll et al. 1996), but instead tend to be clustered around two different values; i.e., most tasks in the set of tasks can be classified into two categories: ''heavy'' and ''light'' tasks.

In this set of experiments, we can no longer use the task-generator proposed by Ripoll and colleagues (which generates systems of a given total utilization). Our task-generation methodology is instead as follows: we choose values for the average utilization of the heavy tasks $\bar{x}_1$, and the average utilization of the light tasks $\bar{x}_2$, and generate task systems comprised of 50 tasks as follows:

1. $n_1 \leftarrow 0$;
2. Generate[4] n_1 heavy tasks:
   $u_i \leftarrow \overline{\text{normal}}(\bar{x}_1, \sigma_1) \quad i = 1, \ldots, n_1$;
3. Generate $50 - n_1$ light tasks:
   $u_i \leftarrow \overline{\text{normal}}(\bar{x}_2, \sigma_2) \quad i = n_1 + 1, \ldots, n$;
4. Re-order the utilization factors such that $u_1 \geq u_2 \geq \cdots \geq u_n$;
5. $m_{\min} \leftarrow \min_{k=1}^{n} \left\{ (k-1) + \left\lceil U(\tau^{(k+1)})/1 - u_k \right\rceil \right\}$;

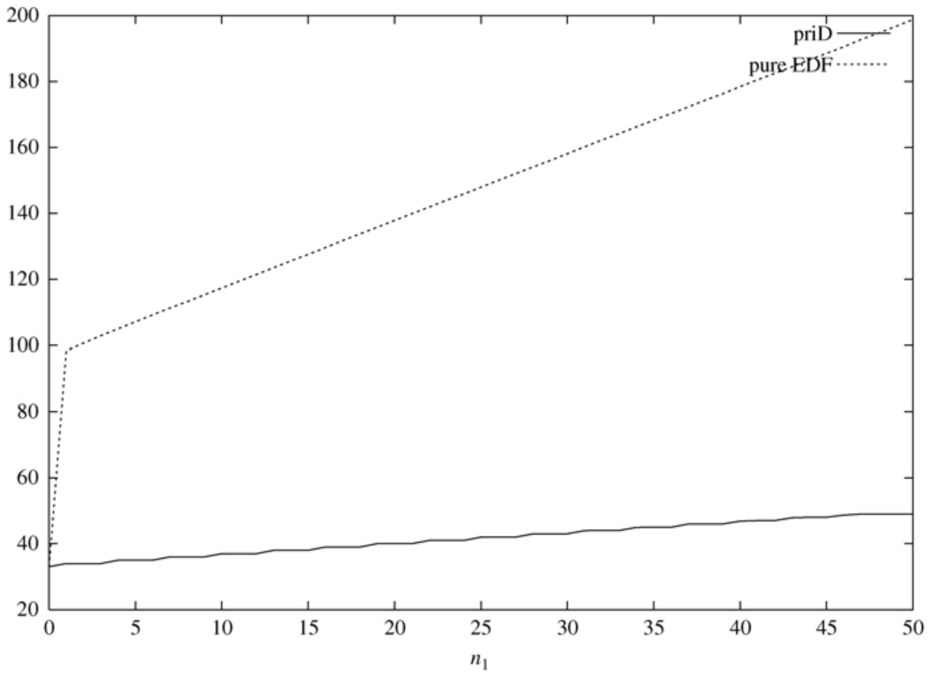

Figure 2. Average number of processors needed, as a function of number of heavy tasks ($\bar{x}_1 = 0.8$; $\bar{x}_2 = 0.4$).

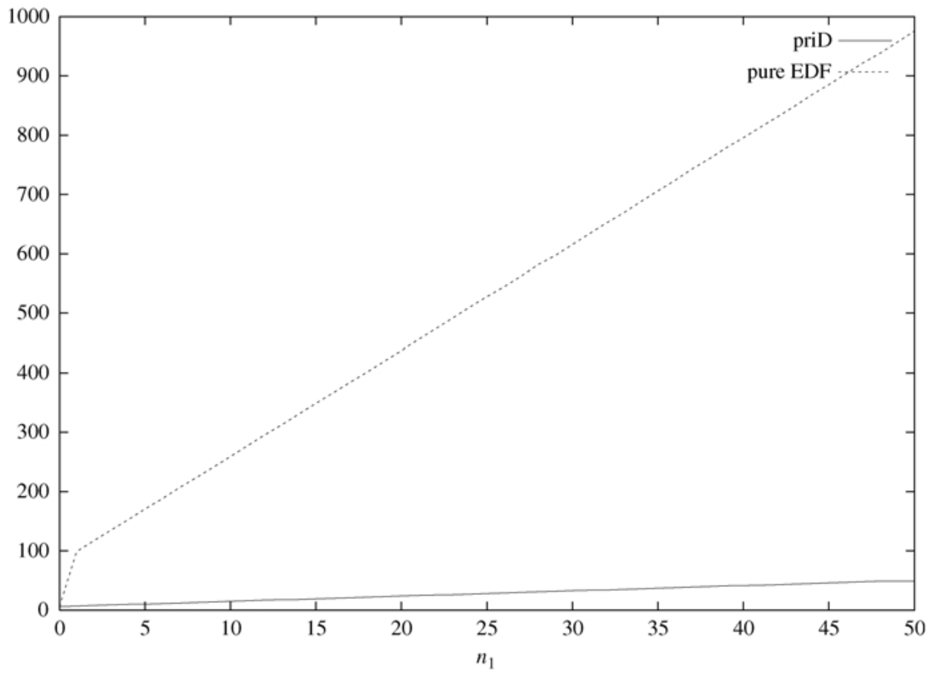

Figure 3. Average number of processors needed, as a function of number of heavy tasks ($\bar{x}_1 = 0.95$; $\bar{x}_2 = 0.1$).

```
6. $\tilde{m}_{\min} \leftarrow \min\{m \mid m > 0 \wedge U(\tau) \le m^2/2 \cdot m - 1\}$;
7. $n_1 \leftarrow n_1 + 1$;
8. if $n_1 \le 50$ repeat from step 2.
```

Figures 2 and 3 show the average number of processors needed by Algorithms PriD and EDF, as a function of the number of heavy tasks n_1. These graphs are obtained by applying the above algorithm to a large number of randomly chosen utilization factors. Figure 2 corresponds to the case where $\bar{x}_1 = 0.8$ and $\bar{x}_2 = 0.4$. Figure 3 corresponds to the case where $\bar{x}_1 = 0.95$ and $\bar{x}_2 = 0.1$. In both cases, we observe that Algorithm PriD compares more favorably to EDF, than was the case with task-systems generated according to the Ripoll et al. task-generator (Ripoll et al., 1996).

7. Summary

Despite the fact that it is provably non-optimal in multiprocessor systems, there is considerable interest in being able to implement the EDF scheduling algorithm upon multiprocessor platforms. In this paper, we have provided a comprehensive analysis of the EDF-scheduling of periodic task systems upon multiprocessor platforms, by proving a tight utilization-based feasibility condition which depends upon both the total utilization of the system and the maximum utilization of any individual task comprising the system.

We believe that many of the properties of EDF that contribute to its popularity among real-time systems designers—efficient implementations, bounded preemptions and interprocessor migrations, etc.—are in fact satisfied by all priority-driven scheduling algorithms. Accordingly, we have proposed and evaluated here Algorithm PriD, a new priority-driven scheduling algorithm for scheduling periodic task systems upon multiple processors that is provably superior to EDF in the sense that it schedules all periodic task systems that EDF can schedule, and in addition schedules some periodic task systems for which EDF may miss some deadlines.

Notes

1. 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.
2. Observe that identical multiprocessors are a special case of uniform multiprocessors, in which the computing capacities of all processors are equal and generally assumed equal to unity.
3. Alternatively, τ_1's period can be chosen to be infinitesimally larger than p—this would force EDF to schedule τ_1's job last, without changing the value of m.
4. $\overline{\text{normal}}(\bar{x}, \sigma)$ represents a pseudo-random number generator which uses the normal distribution (with an average of $\bar{x}$ and a standard deviation of σ) restricted to values in the interval $(0, 1)$.

Acknowledgment

This research has been supported in part by the National Science Foundation (Grant Nos. CCR-9988327 and ITR-0082866).

References

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

Burchard, A., Liebeherr, J., Oh, Y., and Son, S. H. 1995. Assigning real-time tasks to homogeneous multiprocessor systems. *IEEE Transactions on Computers* 44(12): 1429–1442.

Dertouzos, M. 1974. Control robotics: The procedural control of physical processors. In *Proceedings of the IFIP Congress*, pp. 807–813.

Funk, S., Goossens, J., and Baruah, S. 2001. On-line scheduling on uniform multiprocessors. In *Proceedings of the IEEE Real-Time Systems Symposium*. IEEE Computer Society Press.

Ha, R. 1995. *Validating Timing Constraints in Multiprocessor and Distributed Systems*. Ph.D. thesis, Department of Computer Science, University of Illinois at Urbana-Champaign. Available as Technical Report No. UIUCDCS-R-95-1907.

Ha, R., and Liu, J. W. S. October 1993. Validating timing constraints in multiprocessor and distributed real-time systems. Technical Report UIUCDCS-R-93-1833, Department of Computer Science, University of Illinois at Urbana-Champaign.

Ha, R., and Liu, J. W. S. 1994. Validating timing constraints in multiprocessor and distributed real-time systems. In *Proceedings of the 14th IEEE International Conference on Distributed Computing Systems*. Los Alamitos: IEEE Computer Society Press.

Kalyanasundaram, B., and Pruhs, K. 1995. Speed is as powerful as clairvoyance. In *36th Annual Symposium on Foundations of Computer Science (FOCS'95)*. Los Alamitos: IEEE Computer Society Press, pp. 214–223.

Leung, J. 1989. A new algorithm for scheduling periodic real-time tasks. *Algorithmica* 4: 209–219.

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

Liu, C. L. 1969. Scheduling algorithms for multiprocessors in a hard real-time environment. *JPL Space Programs Summary 37–60* II: 28–31.

Mok, A. 1998. Task management techniques for enforcing ED scheduling on a periodic task set. In *Proceedings of the 5th IEEE Workshop on Real-Time Software and Operating Systems*. Washington DC, pp. 42–46.

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

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

Sanjoy Baruah is an associate professor in the Department of Computer Science at the University of North Carolina at Chapel Hill. He received his Ph.D. from the University of Texas at Austin in 1993. His research and teaching interests are in scheduling theory, real-time and safety-critical system design, and resource-allocation and sharing in distributed computing environments.

Shelby Funk is a Ph.D. candidate in computer science at The University of North Carolina at Chapel Hill. Her research interests are in multiprocessor scheduling scheduling theory, real-time and safety-critical system design, and resource-allocation and sharing in distributed computing environments. Prior to entering graduate school, she worked as a computer programmer at a consulting firm. Ms. Funk has obtained two M.S. degrees, one in mathematics and one in computer science, from The University of North Carolina, and a B.S. degree in mathematics from the University of Maryland at College Park.

Joël Goossens received his M.S. degree in computer science in 1992, his M.S. degree in network and management in 1993 and his Ph.D. degree in computer science in 1999, all from the Université Libre de Bruxelles, Belgium. He is currently Assistant Professor in the Department of Computer Science of the same institution and teaches algorithms and programming, object modeling technique and compiler design. His main interests are presently in real-time scheduling theory and computer integrated manufacturing.