Real-time Scheduling on Multiprocessors

Joël Goossens Sanjoy Baruah Shelby Funk

Joël Goossens
Université Libre de Bruxelles
Département d'Informatique CP 212
B-1050 Bruxelles, Belgium
E-mail: Joel.Goossens@ulb.ac.be

Sanjoy Baruah & Shelby Funk
The University of North Carolina
Chapel Hill, North Carolina
USA
E-mail: {baruah,shelby}@cs.unc.edu

Abstract

We have recently been studying the scheduling of real-time systems upon uniform and identical multiprocessor platforms. In particular, we have been exploring the use of EDF-scheduling in such systems; this paper summarizes some of our recent findings in this field.

It has previously been shown that EDF is not optimal for multiprocessors in the sense that it is upon uniprocessors. Nevertheless, EDF remains a good algorithm to use in multiprocessor systems – we formally justify this statement, by obtaining conditions which permit us to determine whether a task system can be scheduled to meet all deadlines using EDF upon a given uniform multiprocessor platform, provided we know it to be feasible upon some other uniform multiprocessor platform.

We apply this result to obtain a test for determining whether a *periodic task system* can be scheduled using EDF upon any given uniform multiprocessor platform. For the problem of scheduling periodic task systems upon identical multiprocessors, we propose a new priority-driven scheduling algorithm that is provably superior to EDF for this purpose.

keywords: Multiprocessors scheduling; periodic tasks; Earliest Deadline First.

1 Introduction

In **hard-real-time** systems, there are certain basic units of work, known as **jobs**, which must be executed in a timely manner. In one popular model of hard-real-time systems, each job is assumed to be characterized by three parameters – an *arrival time*, an *execution requirement*, and a *deadline*, with the interpretation that the job must be executed for an amount equal to its execution requirement between its arrival time and its deadline.

The scheduling of hard-real-time systems has been much studied, particularly upon **uniprocessor** platforms — upon machines in which there is exactly one shared processor available, and all the jobs in the system are required to execute on this single shared processor. In such systems, it have been proven that the earliest deadline first scheduling algorithm (EDF) is an optimal scheduling algorithm in the sense that if a real-time system can be scheduled such that all jobs complete by their deadlines, then EDF will also schedule the system such that all jobs complete by their deadlines.

In **multiprocessor** platforms there are several processors available upon which these jobs may execute. Not very much is known about real-time scheduling upon multiprocessors; for example, no multiprocessor analog to the uniprocessor optimality of EDF is known. In this paper, we will be studying the scheduling of hard-real-time systems on multiprocessor platforms, under the following assumptions:

Job preemption is permitted. That is, a job executing on a processor may be preempted prior to completing execution, and its execution may be resumed later. We assume that there is no penalty associated with such preemption.

Job migration is permitted. That is, a job that has been preempted on a particular processor may resume execution on the same or a different processor. Once again, we assume that there is no penalty associated with such migration.

Job parallelism is forbidden. That is, each job may execute on at most one processor at any given instant in time.

A taxonomy of multiprocessor platforms. 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.

Our results. We have recently been studying the scheduling of real-time systems upon uniform and identical multiprocessor platforms. In particular, we have been exploring the use of EDF-scheduling in such systems; this paper summarizes some of our recent findings in this field.

- It has previously been shown that EDF is not optimal for multiprocessors in the sense that it is upon uniprocessors. Nevertheless, EDF remains a good algorithm to use in multiprocessor systems – we formally justify this statement, by obtaining conditions (Theorem 2) which permit us to determine whether a task system can be scheduled to meet all deadlines using EDF upon a given uniform multiprocessor platform, provided we know it to be feasible upon some other uniform multiprocessor platform.
- We apply this result (Section 3.1) to obtain a test for determining whether a *periodic task system* [8] can be scheduled using EDF upon any given uniform multiprocessor platform.
- For the problem of scheduling periodic task systems upon identical multiprocessors, we propose (Section 4) a new priority-driven scheduling algorithm that is provably superior to EDF for this purpose.

Organization of this document. The remainder of this paper is organized as follows. In Section 2, we describe our system, our task/job model, and formally specify our EDF implementation for multiprocessor platforms. In Section 3, we present our major results concerning the on-line scheduling of real-time jobs upon uniform multiprocessor platforms using EDF. In Section 3.1, we apply this technique to the scheduling of periodic task systems on uniform multiprocessor platforms. In Section 4, we apply the theory developed in Section 3 to obtain utilization-based EDF-schedulability bounds for periodic task systems upon identical multiprocessors. We propose a new priority-driven scheduling algorithm which is provably superior to EDF for the scheduling of periodic real-time task systems upon identical multiprocessors. In Section 5, we summarize the results presented in this paper.

2 Definitions and assumptions

Throughout this paper, we will consider the scheduling of hard-real-time systems upon a uniform multiprocessor platform comprised of m processors (since identical multiprocessors are a special case of uniform multiprocessors, our results apply to identical multiprocessors as well – where more specialized results are derivable for the identical multiprocessor case, we will point this out explicitly).

2.1 Processor model

We begin with some basic notation used to describe a uniform multiprocessor platform π.

Definition 1 Let π denote an m-processor multiprocessor platform with processor speeds $s_1, \ldots, s_m$, where $s_i \geq s_{i+1}$ for $i = 1, \ldots, m-1$. The following attributes of π are defined:

$m(\pi)$**:** denotes the number of processors in π: $m(\pi) \stackrel{\text{def}}{=} m$.

$s_i(\pi)$**:** denotes the speed of the i^{th} fastest processor of π: $s_i(\pi) \stackrel{\text{def}}{=} s_i$.

$S(\pi)$**:** denotes π's cumulative processing power: $S(\pi) \stackrel{\text{def}}{=} \sum_{i=1}^{m(\pi)-1} s_i(\pi)$.

$\lambda(\pi)$**:** We define an additional parameter $\lambda(\pi)$ as follows:

$$\lambda(\pi) \stackrel{\text{def}}{=} \max_{k=1}^{m(\pi)-1} \frac{\sum_{i=k+1}^{m(\pi)} s_i(\pi)}{s_k(\pi)} \ . \tag{1}$$

■

Intuitively, the parameter $\lambda(\pi)$ measures the degree of "identicalness" of π — the closer π is to being an identical system, the larger the value of $\lambda(\pi)$. This value has an upper bound of $(m(\pi) - 1)$, which occurs when π consists of $m(\pi)$ identical processors. At the other extreme, $\lambda(\pi)$ is arbitrarily small when the processors have very different speeds. For example, $\lambda(\pi) < \epsilon$ for the $m(\pi)$-processor platform where $s_{i+1}(\pi) \leq \frac{\epsilon}{m(\pi)} \cdot s_i(\pi)$ for $i = 1, \ldots, m(\pi) - 1$.

2.2 Job model

We will assume that a hard-real-time system may be modelled as an arbitrary collection of individual **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 $[r_j, d_j)$.

Periodic tasks. The periodic task model [8] has proven very useful for the modelling and analysis of real-time computer application systems. In this model, each recurring real-time process is modelled as a **periodic task**, and is characterized by two parameters – an **execution requirement** and a **period**. (While the execution time may be any non-negative number, deadlines are assumed here to be non-negative rational numbers.) In this section, we will study the scheduling of real-time systems that can be completely modelled as finite collections of such periodic tasks. Accordingly, we will model a real-time system $\tau \stackrel{\text{def}}{=} \{T_1, T_2, \ldots, T_n\}$ as being comprised of a collection of n periodic tasks. We will assume that all the system parameters – the number of tasks in the system, and the execution-requirement and period parameters of each task – are a priori known. Each periodic task generates an infinite sequence of jobs which need to be executed by the system. A periodic task $T_i = (e_i, p_i)$ with execution-requirement parameter e_i and period parameter p_i generates a job at each instant $k \cdot p_i$, which needs to execute for e_i units by a deadline of $(k+1) \cdot p_i$, for all non-negative integers k. (In the remainder of this section, we

will often use the symbol τ itself to denote the infinite collection of jobs generated by the tasks in periodic task system τ.)

We define the *utilization* u_i of task T_i to be the ratio of its execution requirement to its period: $u_i \stackrel{\text{def}}{=} e_i/p_i$. Without loss of generality, we assume that the tasks in τ are indexed according to non-increasing utilization: $u_i \geq u_{i+1}$ for all $i,\ 1 \leq i < n$.

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

We define a uniform multiprocessor scheduling algorithm to be **work-conserving** if and only if it satisfies the following conditions:

- No processor is idled while there are active jobs awaiting execution.
- If at some instant there are fewer than m active jobs awaiting execution (recall that m denotes the number of processors in the uniform multiprocessor platform), then the active jobs are executed upon the fastest processors. That is, it is the case that at any instant t if the j'th-slowest processor is idled by the work-conserving scheduling algorithm, then the k'th-slowest processor is also idled at instant t, for all $k > j$.

EDF on uniform processors. Recall that the earliest deadline first scheduling algorithm (EDF) chooses for execution at each instant in time the currently active job[s] that have the smallest deadlines. In this research, we assume that EDF is implemented upon uniform multiprocessor systems according to the following rules:

1. No processor is idled while there is an active job awaiting execution.
2. When fewer than m jobs are active, they are required to execute upon the fastest processors while the slowest are idled.
3. Higher priority jobs are executed on faster processors. More formally, if the j'th-slowest processor is executing job J_g at time t under our EDF implementation, it must be the case that the deadline of J_g is not greater than the deadlines of jobs (if any) executing on the $(j+1)$'th-, $(j+2)$'th-, ..., m'th-slowest processors.

The first two conditions above imply that EDF is a work-conserving scheduling algorithm.

EDF is known to be an **optimal** scheduling algorithm in uniprocessor systems — a task set τ is feasible on a uniprocessor π if and only if it is EDF-schedulable on π. Unfortunately, EDF is *not* optimal on multiprocessors since there are tasks

sets that are feasible on some multiprocessors but will miss deadlines if EDF is used [1]. There are nevertheless significant advantages to using EDF for scheduling on multiprocessor platforms. While it is beyond the scope of this paper to describe in detail these advantages, some important ones are listed below:

- Very efficient implementations of EDF have been designed (see, e.g., [9]).
- It can be shown that when a set of jobs is scheduled using EDF, then the total number of *preemptions* is bounded from above by the number of jobs in the set.
- It can be similarly be shown that the total number of *interprocessor migrations* that of individual jobs is bounded from above by the number of jobs.

3 Theoretical foundations

It follows from the result of Hong and Leung [7] that no uniform multiprocessor online scheduling algorithm can be optimal. Suppose that a given set of jobs is known to be feasible on a given m-processor uniform multiprocessor platform π. In this section, we obtain conditions upon some other m-processor uniform multiprocessor platform π' under which the EDF scheduling algorithm guarantees to meet all deadlines for this set of jobs on π'. In Lemma 1 below, we first prove a general result that relates the amount of work done at each instant in time by any work-conserving scheduling algorithm (such as EDF) executing on π' with the amount of work done by an optimal scheduling algorithm executing on π, when both algorithms are executing the same set of jobs. We use this lemma in Theorem 2 to determine conditions under which EDF executing on π' will meet all deadlines of a set of jobs known to be feasible on π.

First, some additional notation.

Definition 2 ($\mathbf{W(A, \pi, I, t)}$.) 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 2 below) upon the uniform multiprocessor platforms π and 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. This condition is expressed as a constraint on the parameter $\lambda(\pi')$ of the uniform multiprocessor platform π'. Condition 2 expresses the additional computing capacity needed by π' (i.e., the amount by which the total computing capacity of π' must exceed that of π) in terms of this $\lambda(\pi')$ parameter, and the speed of the fastest processor in π — the smaller the value of $\lambda(\pi')$ (the more π' deviates from being an identical multiprocessor), the smaller the amount of this excess processing capacity needed.

Lemma 1 ([2]) Let π and π' denote uniform multiprocessor platforms. Let A denote any $m(\pi)$-processor uniform multiprocessor scheduling algorithm, and A' any *work-conserving* $m(\pi')$-processor uniform multiprocessor scheduling algorithm. If the following condition is satisfied by platforms π and π':

$$S(\pi') \geq \lambda(\pi') \cdot s_1(\pi) + S(\pi) \tag{2}$$

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

$$W(A', \pi', I, t) \geq W(A, \pi, I, t) \ . \tag{3}$$

■

Lemma 1 allows us to reason about the total execution of work-conserving algorithms. The next theorem shows that we can use this knowledge to deduce whether a work-conserving algorithm can feasibly schedule a task set: it states that any collection of jobs I that is feasible on a uniform multiprocessor platform π will be scheduled to meet all deadlines by Algorithm EDF on any platform π' satisfying Condition 2 of Lemma 1.

Theorem 2 ([2]) Let I denote an instance of jobs that is feasible on an uniform multiprocessor platform π. Let π' denote another uniform multiprocessor platform. If Condition 2 of Lemma 1 is satisfied by platforms π and π':

$$S(\pi') \geq \lambda(\pi') \cdot s_1(\pi) + S(\pi)$$

then I will meet all deadlines when scheduled using the EDF algorithm executing on π'.

■

As an immediate corollary to Theorem 2 above, we obtain the result of Phillips, Stein, Torng, and Wein [10] concerning EDF-scheduling on identical multiprocessors:

Corollary 3 If a set of jobs is feasible on an identical m-processor platform, then the same set of jobs will be scheduled to meet all deadlines by EDF on an identical m-processor platform in which the individual processors are $(2 - \frac{1}{m})$ times as fast as in the original system.

■

Theorem 2 characterizes a uniform multiprocessor platform π' according to its parameter "$\lambda(\pi')$" (as defined in Equation 1), and relates the EDF-feasibility of a system, known to be feasible on some platform π, to the cumulative capacities of π and π', the speed $s_1(\pi)$ of the fastest processor in π, and this parameter $\lambda(\pi')$ of platform π'. Theorem 4 below asserts that, for this particular characterization of a uniform multiprocessor system, the bound represented by Theorem 2 is a tight one and EDF is optimal in the sense that no other on-line scheduling algorithm can make a better guarantee:

Theorem 4 ([2]) There exist uniform multiprocessor platforms π and π' and an instance I of hard-real-time jobs such that

- Instance I is feasible on platform π,
- The relationship between the various parameters of π and π' is as follows

$$S(\pi') = \lambda(\pi') \cdot s_1(\pi) + S(\pi) - \epsilon \ . \tag{4}$$

  (where ϵ is assumed to be an arbitrarily small positive number; i.e., the condition of Theorem 2 is not satisfied by an arbitrarily small amount ϵ.)

- Instance I is infeasible on uniform multiprocessor platform π'.

■

3.1 EDF-scheduling of periodic tasks

In this section, we apply the theory developed in the previous section to the scheduling of periodic task, systems on uniform multiprocessor platforms.

Lemma 5 ([2]) Let $\tau = \{T_1, \ldots, T_n\}$ denote a collection of periodic tasks indexed according to non-increasing utilization (i.e., $u_i \geq u_{i+1}$ for all i, $1 \leq i < n$, where $u_i \stackrel{\text{def}}{=} \frac{e_i}{p_i}$). τ is feasible on $\pi = [u_1, \ldots, u_n]$.

Theorem 6 ([2]) Let π' denote a uniform multiprocessor platform. Periodic task system τ will meet all deadlines when scheduled on π' using EDF, if the following condition holds

$$S(\pi') \quad \geq \lambda(\pi') \cdot u_1 + U_n. \tag{5}$$

Where $U_n \stackrel{\text{def}}{=} \sum_{i=1}^{n} u_i$.

Proof. Immediately follows from Theorem 2 and Lemma 5. ■

We now illustrate the use of Theorem 6 by an example.

Example 7 Consider a task system τ comprised of five tasks:

$$\tau = \{(15, 10), (4, 5), (12, 20), (6, 15), (2, 10)\};$$

for this system, $u_1 = 1.5$, $u_2 = 0.8$, $u_3 = 0.6$, $u_4 = 0.4$, and $u_5 = 0.2$. Suppose that τ is to be EDF-scheduled on the uniform multiprocessor platform $\pi' = [3, 1, 0.5]$ — will all deadlines be met?

By Equation 1, the value of $\lambda_{\pi'}$ for the uniform multiprocessor platform π' is

$$\lambda_{\pi'} = \max(\frac{1 + 0.5}{3}, \frac{0.5}{1}) = \frac{1}{2} \ ,$$

the total computing capacity is

$$3 + 1 + 0.5 = 4.5 \ ,$$

and $U_n = 3.5$.

The Condition 5 is therefore

$$\begin{aligned} 4.5 &\geq 0.5 \cdot 1.5 + 3.5 \\ \equiv 4.5 &\geq 0.75 + 3.5 \\ \equiv 4.5 &\geq 4.25 \end{aligned}$$

and τ can consequently be scheduled by EDF to meet all deadlines on π'. ■

4 Priority-driven algorithms

In this section, we extend the theory developed in Section 3 to obtain utilization-based EDF-schedulablity bounds for periodic task systems upon identical multiprocessors.

Lemma 8 ([3]) Let π denote a *uniform* multiprocessor platform, and π' 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:

$$m' \geq \frac{S(\pi) - s_1(\pi)}{1 - s_1(\pi)} \tag{6}$$

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

$$W(A', \pi', I, t) \geq W(A, \pi, I, t) \ . \tag{7}$$

■

The following theorem applies Lemma 8 to the case where the work-conserving algorithm A' of Lemma 8 is Algorithm EDF, and algorithm A of Lemma 8 is an optimal (offline) scheduler.

Theorem 9 ([3]) Let π denote a uniform multiprocessor platform. 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 6 of Lemma 8 is satisfied, then I will meet all deadlines when scheduled using the EDF algorithm executing on π'.

4.1 EDF-scheduling of periodic task systems

Lemma 8 and Theorem 9 above 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, these results nevertheless turn out to be useful towards developing a framework for scheduling periodic task systems on multiprocessors.

Recall that $\tau = \{T_1, T_2, \ldots, T_n\}$ denotes a periodic task system comprised of n tasks, indexed in order of non-increasing utilization. We introduce the notation $\tau^{(i)}$ to refer to the task system comprised of the $(n - i + 1)$ minimum-utilization tasks in τ:

$$\tau^{(i)} \stackrel{\text{def}}{=} \{T_i, T_{i+1}, \ldots, T_n\} .$$

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

By a direct application Theorem 9, we obtain below a sufficient condition for a periodic task system to be successfully scheduled by EDF. By Theorem 9, 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 9, we obtain the following theorem:

Theorem 10 ([3]) Periodic task system τ can be EDF-scheduled upon an identical multiprocessor platform comprised of m unit-capacity processors, provided

$$m \geq \left\lceil \frac{U(\tau) - u_1}{1 - u_1} \right\rceil \tag{8}$$

■

(Note that, as $u_1 \to 1$, the right-hand side of Inequality 8 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 8 could be replaced by $\min(n, \left\lceil \frac{U(\tau)-u_1}{1-u_1} \right\rceil)$. For reasons of algebraic simplicity, we do not make this explicit in the remainder of this paper.)

Theorem 11 follows by algebraic simplification of Equation 8:

Theorem 11 ([3]) Periodic task system τ can be EDF-scheduled upon m unit-speed identical processors, provided its cumulative utilization is bounded from above as follows:

$$U(\tau) \leq m - u_1 \cdot (m - 1) . \tag{9}$$

■

It turns out that the bounds of Theorem 10 and 11 are in fact tight:

Theorem 12 ([3]) 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, $\epsilon \ll u_1$. EDF cannot schedule some periodic task systems with cumulative utilization $m - u_1(m - 1) + \epsilon$ in which the largest-utilization task has utilization equal to u_1, upon m unit-speed processors.

4.2 Priority-driven scheduling of periodic task systems

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 [5, 6, 4], who proposed the following definition:

Definition 3 (Priority-driven algorithms [6].) 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 the *least laxity algorithm* is not.

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 $\lceil (U(\tau) - u_1)/(1 - u_1) \rceil$ processors mandated by Theorems 10 and 12. Recall that tasks in τ are indexed according to non-increasing utilization (i.e., $u_i \geq u_{i+1}$ for all i, $1 \leq i < n$), 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 $i < k$, T_i's jobs are assigned highest priority (ties broken arbitrarily) — this is trivially achieved within an EDF implementation by setting all deadlines of T_i equal to $-\infty$.

For all $i \geq k$, T_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 13 ([3]) Periodic task system τ will be scheduled to meet all deadlines on m unit-speed processors by Algorithm $\mathsf{EDF}^{(k)}$, where

$$m = (k - 1) + \left\lceil \frac{U(\tau^{(k+1)})}{1 - u_k} \right\rceil \tag{10}$$

Corollary 14 ([3]) Periodic task system τ will be scheduled to meet all deadlines on

$$m_{\min}(\tau) \stackrel{\text{def}}{=} \min_{k=1}^{n} \left\{ (k - 1) + \left\lceil \frac{U(\tau^{(k+1)})}{1 - u_k} \right\rceil \right\} \tag{11}$$

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

Algorithm PriD. Based upon Corollary 14 above, we propose the following priority-driven scheduling algorithm for scheduling periodic task systems upon identical multiprocessors: Given a periodic task system $\tau = \{T_1, T_2, \ldots, T_n\}$ with $u_i \leq u_{i+1}$ for all i, $1 \leq i < n$, Algorithm PriD computes $m_{\min}(\tau)$ according to Equation 11, and schedules τ by Algorithm $\mathsf{EDF}^{(k_{\min}(\tau))}$.

Example 15 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 11 is minimized for $k = 3$; hence, $k_{\min}(\tau) = 3$ and $m_{\min}(\tau)$ equals

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

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

By contrast, Theorem 10 can only guarantee that all deadlines will be met upon $\left\lceil \frac{U(\tau)-u_1}{1-u_1} \right\rceil \approx \lceil 1.557/0.1 \rceil = 16$ processors, if τ were scheduled using EDF.

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. [11] 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 [11] except the utilization factor of the system which is uniformly drawn from interval $[1, 10]$, the computation times are

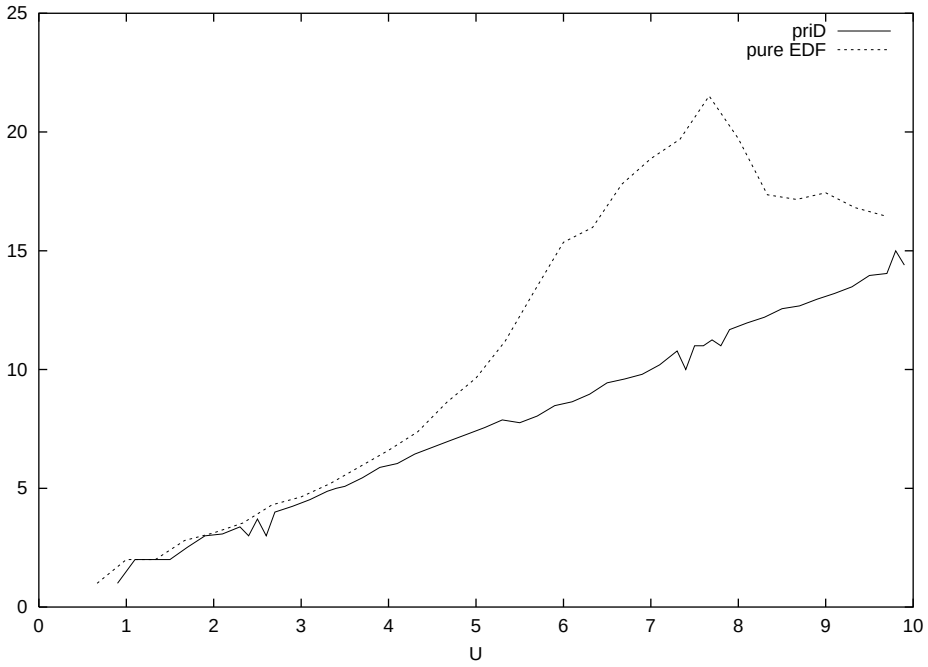

Figure 1: Average number of processors needed, as a function of total utilization $U(\tau)$.

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 by the Ripoll et al. generator [11], 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 $\overline{x}_1$, and the average utilization of the light tasks $\overline{x}_2$, and generate task systems comprised of 50 tasks as follows:

1. $n_1 \leftarrow 0$;
2. Generate[1] n_1 heavy tasks:

[1] $\overline{\text{normal}}(\overline{x}, \sigma)$ represents a pseudo-random number generator which uses the normal distribution (with an average of $\overline{x}$ and a standard deviation of σ) restricted to values in the interval $(0, 1)$.

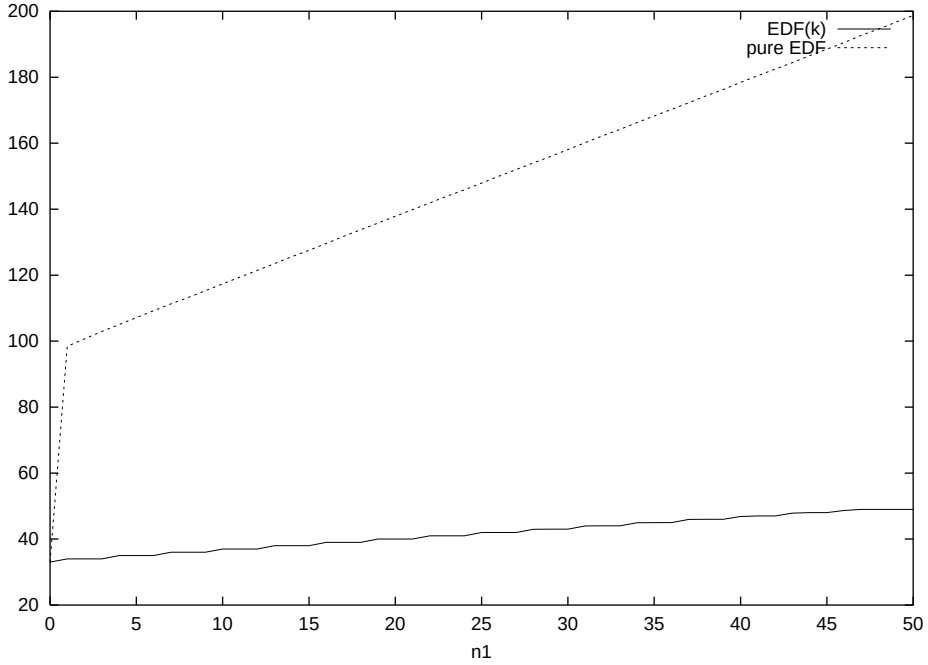

Figure 2: Average number of processors needed, as a function of number of heavy tasks ($\overline{x}_1 = 0.8$; $\overline{x}_2 = 0.4$).

$u_i \leftarrow \overline{\text{normal}}(\overline{x}_1, \sigma_1) \qquad i = 1, \ldots, n_1;$

3. Generate $50 - n_1$ light tasks:
   $u_i \leftarrow \overline{\text{normal}}(\overline{x}_2, \sigma_2) \qquad i = n_1 + 1, \ldots, n;$

4. Re-order the utilization factors such that $u_1 \geq u_2 \geq \cdots \geq u_n$;
5. $m_{\min} \leftarrow \min_{k=1}^{n} \left\{ (k-1) + \left\lceil \frac{U(\tau^{(k+1)})}{1-u_k} \right\rceil \right\}$;
6. $\widetilde{m}_{\min} \leftarrow \min \left\{ m \mid m > 0 \wedge U(\tau) \leq \frac{m^2}{2 \cdot m - 1} \right\}$;
7. $n_1 \leftarrow n_1 + 1$;
8. if $n_1 \leq 50$ 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 $\overline{x}_1 = 0.8$ and $\overline{x}_2 = 0.4$. Figure 3 corresponds to the case where $\overline{x}_1 = 0.95$ and $\overline{x}_2 = 0.1$. 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 [11].

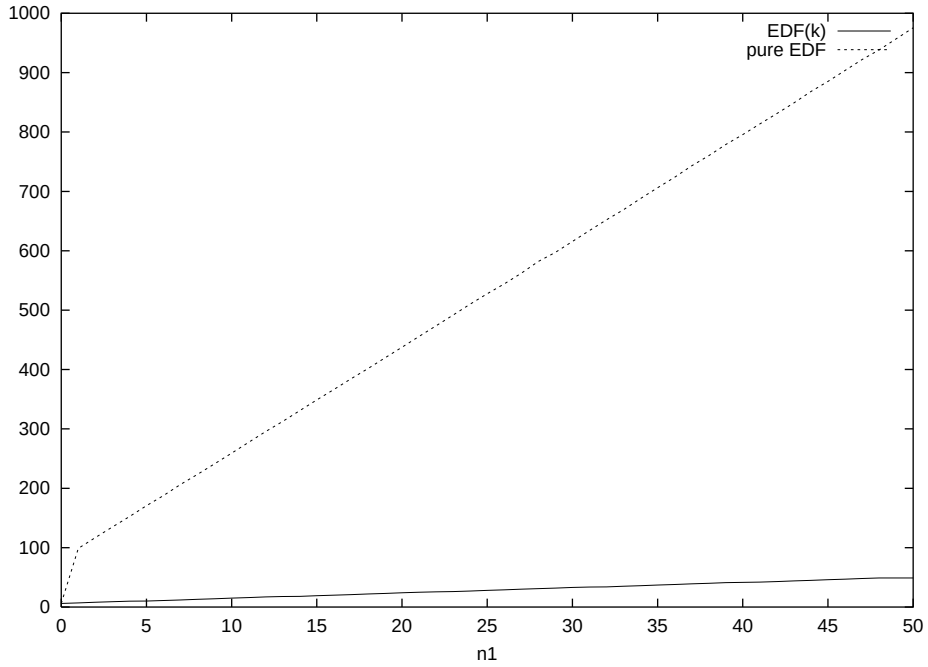

Figure 3: Average number of processors needed, as a function of number of heavy tasks ($\overline{x}_1 = 0.95$; $\overline{x}_2 = 0.1$).

5 Conclusion

In this paper, we summarized some of our recent findings in the field of on-line scheduling on multiprocessors. We have first presented our major result: a ressource augmentation technique to use EDF on multiprocessors and its application to the scheduling of periodic task systems. We have applied the theory to obtain utilization-based EDF-schedulability bounds for periodic task systems upon identical multiprocessors. We proposed a new priority-driven scheduling algorithm which is provably prior to EDF for scheduling of periodic real-time task systems upon identical multiprocessors.

References

[1] Dhall, S. K., and Liu, C. L. On a real-time scheduling problem. *Operations Research 26* (1978), 127–140.

[2] Funk, S., Goossens, J., and Baruah, S. On-line scheduling on uniform multiprocessors. In *Proceedings of the* 22^{nd} *Real-Time Systems Symposium* (London, England, December 2001), IEEE Computer Society Press, pp. 183–192.

[3] Goossens, J., Funk, S., and Baruah, S. Priority-driven scheduling of periodic task systems on multiprocessors. *Real-Time Systems: The International Journal of Time-Critical Computing* (2001). Accepted for publication.

[4] Ha, R. *Validating timing constraints in multiprocessor and distributed systems.* PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 1995. Available as Technical Report No. UIUCDCS-R-95-1907.

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

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

[7] Hong, K., and Leung, J. On-line scheduling of real-time tasks. In *Proceedings of the Real-Time Systems Symposium* (Huntsville, Alabama, December 1988), IEEE, pp. 244–250.

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

[9] Mok, A. Task management techniques for enforcing ED scheduling on a periodic task set. In *Proc. 5th IEEE Workshop on Real-Time Software and Operating Systems* (Washington D.C., May 1988), pp. 42–46.

[10] 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.

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