Robustness Results Concerning EDF Scheduling upon Uniform Multiprocessors

Sanjoy Baruah, *Member*, *IEEE*, Shelby Funk, *Student Member*, *IEEE*, and Joël Goossens

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 earliest-deadline-first (EDF) scheduling of hard-real-time systems upon uniform multiprocessor machines is considered. It is known that online algorithms tend to perform very poorly in scheduling such hard-real-time systems on multiprocessors; resource-augmentation techniques are presented here that permit online algorithms in general (EDF in particular) to perform better than may be expected given these inherent limitations. It is shown that EDF scheduling upon uniform multiprocessors is robust with respect to both job execution requirements and processor computing capacity.

Index Terms—Uniform multiprocessors, hard-real-time systems, resource augmentation, earliest deadline first, robustness.

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.

In this paper, we study the scheduling of hard-real-time systems upon **uniform multiprocessor** platforms. In contrast to *identical* multiprocessors, in which it is assumed that all processors are 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. We consider the scheduling of hard-real-time systems on uniform multiprocessor platforms which allow for *preemptions* and *interprocessor migrations* (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), but forbid *job-level parallelism*—at any instant in time, each job may be executing upon at most one processor.

1.1 Online Scheduling on Multiprocessors

Online scheduling algorithms make scheduling decisions at each time-instant based upon the characteristics of the jobs that have arrived thus far, with no knowledge of jobs that may arrive in the future. Several uniprocessor online scheduling algorithms, such as the earliest deadline first scheduling algorithm (EDF) [5], [1] and the Least Laxity algorithm [7] are known to be **optimal** in the sense that if a set of jobs can be scheduled such that all jobs complete by their deadlines (such real-time instances are known as **feasible** instances), then these algorithms will also schedule these sets of jobs to meet all deadlines. For multiprocessor systems, however, no online scheduling algorithm can be optimal: This was shown for the simplest (identical) multiprocessor model by Dertuozos and Mok [2] and Hong and Leung [3], and the techniques from [2], [3] can be directly extended to the more general uniform machine model.

An important advance in the study of online scheduling upon multiprocessors was obtained by Phillips et al. [8], who explored the use of *resource-augmentation* techniques for the online scheduling of real-time jobs.[1] Phillips et al. investigated whether an online algorithm, if provided with faster processors than necessary for feasibility, could perform better than is implied by the above nonoptimality results. In [8], they showed that the obvious extension to the Earliest Deadline First algorithm for identical multiprocessors could make the following performance guarantee:

> If a real-time instance is feasible on m identical processors, then the same instance will be scheduled to meet all deadlines by EDF on m identical processors *in which the individual processors are* $(2 - \frac{1}{m})$ *times as fast* as in the original system.

That is, EDF is an online algorithm that can guarantee to successfully schedule on m identical processors any set of jobs that is feasible on m processors $m/(2m-1)$ times as fast as those available to EDF.

1.2 This Research

In this paper, we explore the issue of online scheduling of hard-real-time systems on uniform multiprocessors. Upon multiprocessor platforms, it turns out that it is often far

- *S. Baruah and S. Funk are with the Department of Computer Science, CB 3175, University of North Carolina, Chapel Hill, NC 27599-3175. E-mail: {baruah, shelby}@cs.unc.edu.*
- *J. Goossens is with the Département d'Informatique, Université Libre de Bruxelles, Boulevard du Triomphe-C.P. 212, 1050 Bruxelles, Belgium. E-mail: joel.goossens@ulb.ac.be.*

Manuscript received 14 Jan. 2002; revised 14 Jan. 2003; accepted 23 Feb. 2003. For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 115697.

1. Resource augmentation as a technique for improving the performance on online scheduling algorithms was formally proposed by Kalyanasundaram and Pruhs [4].

easier (from a computational complexity perspective) to determine feasibility than it is to check whether a specific algorithm will meet all deadlines. In designing an EDF-schedulability test for uniform multiprocessors, therefore, our approach is as follows: Given the specifications of a uniform multiprocessor platform π_o, we generalize techniques from [8] to obtain a condition (Theorem 1) upon the specifications of any other uniform multiprocessor platform π_1 such that, if π_1 satisfies this condition, then *any hard-real-time task system feasible on π_o will meet all deadlines when scheduled on π_1 using EDF*—in this manner, the issue of determining EDF-schedulability upon π_1 is reduced to the (usually, simpler) problem of determining feasibility upon π_o. We use this condition as the basis for designing a sufficient algorithm to determine whether any instance feasible upon π_o will meet all deadlines if scheduled upon π_1 using EDF, for any given π_o and π_1.

One of the benefits of having provably optimal online scheduling algorithms (such as EDF, in the uniprocessor case) available is that they permit a separation of the concerns of *feasibility-analysis* and *runtime scheduling* during the design of real-time application systems—provided that the system designer can guarantee that a proposed design is feasible, the optimal online scheduling algorithm can be used as a runtime scheduling algorithm which will guarantee to meet all deadlines. In such frameworks, runtime scheduling can be looked upon as a kind of "service" that is provided by a real-time platform to an application system, subject to the following contract—if the application system guarantees to only generate feasible collections of jobs, the service-provider (i.e., the real-time platform) guarantees to meet all deadlines. We believe that our results should be viewed in the same context and therein lies their significance: Provided that an application system only generates collections of jobs that are feasible on a particular platform π_o, our results permit us to derive the specifications of another system π_1 such that the runtime platform π_1 will meet all deadlines if the EDF online scheduling algorithm is used. Hence, once again the system-designer can focus upon designing feasible systems (although the platform upon which feasibility must be guaranteed is a "virtual" one, π_o, and not the actual runtime platform π_1 upon which the system will actually execute). Given the results of Dertouzos and Mok [2] and Hong and Leung [3], this is the best kind of result we can hope to obtain since no uniform multiprocessor online algorithm for scheduling hard-real-time systems can possibly be optimal.

The remainder of this paper is organized as follows: In Section 2, we define our real-time system and processor models and specify how EDF is implemented upon uniform multiprocessors. In Section 3, we present (and prove the correctness of) a sufficient condition (Theorem 1) for ensuring that any instance feasible upon platform π_o is successfully scheduled upon platform π_1, for any given π_o and π_1. In Section 4, we prove that EDF upon uniform multiprocessors is *robust* with respect to processing platform—if an instance is successfully scheduled by EDF upon a particular platform, then it is successfully scheduled by EDF upon any platform that is more powerful. In Section 5, we exploit our knowledge of the robustness of EDF to propose a sufficient algorithm for determining whether any instance feasible upon platform π_o is successfully scheduled upon platform π_1, for any given π_o and π_1.

2 Definitions and Assumptions

A hard-real-time system is specified by specifying an instance of jobs with hard-real-time requirements and the processor platform upon which the jobs are to execute. We will assume that a hard-real-time **instance** is modeled 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 $[r_j, d_j)$. A job instance is defined to be **feasible** upon a specified processing platform if it is possible to schedule the job instance upon the platform such that all job deadlines are met.

In this paper, we are considering the scheduling of job instances upon uniform multiprocessor platforms which permit job preemptions and interprocessor migrations (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), but prohibit any job from executing upon more than one processor at any instant in time. We now introduce some notation for describing 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 ith fastest processor in π is denoted by $s_i(\pi)$.*
- *The total computing capacity of all the processors in π is denoted by $S(\pi)$: $S(\pi) \stackrel{\text{def}}{=} \sum_{i=1}^{m(\pi)} s_i(\pi)$.*

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

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

Very loosely speaking, this parameter $\lambda(\pi)$ of a uniform multiprocessor system π intuitively measures the "degree" by which π differs from an identical multiprocessor platform (in the sense that $\lambda(\pi) = (m-1)$ if π is comprised of m identical processors and becomes progressively smaller as the speeds of the processors differ from each other by greater amounts; in the extreme, if $s_1 > 0$ and $s_2 = s_3 = \cdots = s_m = 0$ would have $\lambda(\pi) = 0$).

2.1 Work-Conserving Scheduling Algorithms

In the context of uniprocessor scheduling, a work-conserving scheduling algorithm is defined to be one that never idles the (sole) 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[2] jobs awaiting execution.
- If, at some instant, there are fewer active jobs awaiting execution than processors available, then the active jobs are executed upon the fastest processors. That is, it is the case that, at any instant t, if the jth slowest processor is idled by the work-conserving scheduling algorithm, then the kth slowest processor is also idled at instant t, for all $k > j$.

2.2 EDF on Uniform Processors

Recall that the earliest deadline first scheduling algorithm (EDF) [5], [1] chooses, for execution at each instant in time, the currently active job(s) that have the smallest deadlines. Upon uniform multiprocessors, we require that EDF implementations satisfy the following additional rules:

1. EDF is implemented as a work-conserving scheduling algorithm. (Because of our qualification of the term "active," it follows that we are assuming that, in our implementations of EDF, a job that fails to meet its deadline will continue to contend for execution and will continue to be scheduled until its execution requirement is satisfied.)
2. Higher priority jobs are executed on faster processors. More formally, if the jth 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 upon the $(j+1)$th, $(j+2)$th, ... slowest processors.
3. **EDF is implemented in a strictly deterministic manner**, that is, jobs are prioritized strictly according to deadlines and ties among deadlines are broken according to some—predetermined—deterministic rule (e.g., according to job identifiers). Hence, given any two jobs J_a and J_b, we can unequivocally determine that either J_a has greater priority than J_b, or that J_b has greater priority than J_a.

A real-time instance is said to be **EDF-feasible** upon platform π if it is successfully scheduled to meet all deadlines by EDF upon platform π.

We conclude this section with additional definitions we will be needing throughout the remainder of this paper.

2. The use of the word "active" here deserves further explanation. Traditionally, a job is considered active in a schedule between its arrival time and its deadline, while it has not received enough execution to satisfy its execution requirement. For reasons that will become clear later, we adopt a slightly different definition of active in this document: A job is **active** in a schedule between its arrival time and the instant at which it receives enough execution to satisfy its execution requirement, irrespective of whether its deadline has elapsed or not. In other words, we do not consider a job inactive once its deadline has elapsed if it has not received enough execution—it continues to contend for execution until it has executed for an amount equal to its execution requirement. (We are primarily interested here in schedules in which all jobs do indeed complete by their deadlines: In such schedules, this definition of "active" corresponds to the traditional definition since no job fails to obtain enough execution by its deadline.)

Definition 2 ($\mathbf{W}(\mathbf{A}, \pi, \mathbf{I}, \tilde{\mathbf{I}}, \mathbf{t})$). *Let I denote any set of jobs, $\tilde{I} \subseteq I$ any subset of I, and π any uniform multiprocessor platform. For any algorithm A and time instant $t \geq 0$, $W(A, \pi, I, \tilde{I}, t)$ denotes the total amount that jobs in $\tilde{I}$ have been executed by algorithm A over the interval $[0, t)$, while executing I on π.*

If we specify four, rather than five, parameters to the function $W(.)$, it is understood to mean that the third and fourth parameters are equal:

Definition 3 ($\mathbf{W}(\mathbf{A}, \pi, \mathbf{I}, \mathbf{t})$). *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.*

Equivalently,

$$W(A, \pi, I, t) \equiv W(A, \pi, I, I, t).$$

Definition 4. *For all uniform multiprocessor platforms π and for all x such that there is a processor of speed x in π, let $\pi \setminus \{x\}$ denote the uniform multiprocessor obtained from π by removing a processor of speed x from π.*

3 EDF Scheduling on Uniform Multiprocessors

Recall that it follows from the results of Dertuozos and Mok [2] and Hong and Leung [3] 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 uniform multiprocessor platform π_o. In this section, we generalize the techniques of Phillips et al. [8] to obtain conditions upon some other uniform multiprocessor platform π_1 under which the EDF scheduling algorithm guarantees to meet all deadlines for this set of jobs on π_1. 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_1 with the amount of work done by an optimal scheduling algorithm executing on π_o, when both algorithms are executing the same set of jobs. We use this lemma in Theorem 1 to determine conditions under which EDF executing on π_1 will meet all deadlines of a set of jobs known to be feasible on π_o.

Lemma 1 below specifies a condition (Condition 2 below) upon the uniform multiprocessor platforms π_o and π_1 under which any work-conserving algorithm A_1 (such as EDF) executing on π_1 is guaranteed to complete at least as much work by each instant in time t as any other algorithm A_o (including an optimal algorithm) executing on π_o, when both algorithms are executing on any set of jobs I. This condition is expressed as a constraint on the parameter $\lambda(\pi_1)$ of the uniform multiprocessor platform π_1. Condition 2 expresses the additional computing capacity needed by π_1 (i.e., the amount by which the total computing capacity of π_1 must exceed that of π_o) in terms of this $\lambda(\pi_1)$ parameter and the speed of the fastest processor in π_o—the smaller the value of $\lambda(\pi_1)$ (the more π_1 deviates from being an identical

multiprocessor), the smaller the amount of this excess processing capacity needed.

Lemma 1. *Let π_o and π_1 denote uniform multiprocessor platforms. Let A_o denote any uniform multiprocessor scheduling algorithm and A_1 any* work-conserving *uniform multiprocessor scheduling algorithm. If the following condition is satisfied by platforms π_o and π_1:*

$$S(\pi_1) \geq \lambda(\pi_1) \cdot s_1(\pi_o) + S(\pi_o), \tag{2}$$

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

$$W(A_1, \pi_1, I, t) \geq W(A_o, \pi_o, I, t). \tag{3}$$

Proof. The proof is by contradiction. Suppose that it is not true, i.e., there is some collection of jobs I and some time-instant by which work-conserving algorithm A_1 executing on π_1 has performed strictly less work than some other algorithm A_o executing on π_o. Let $J_a = (r_a, c_a, d_a)$ denote some job in I with the earliest arrival time such that there is some time-instant t_o satisfying

$$W(A_1, \pi_1, I, t_o) < W(A_o, \pi_o, I, t_o),$$

and the amount of work done on job J_a by time-instant t_o in A_1 is strictly less than the amount of work done of J_a by time-instant t_o in A_o.

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

$$W(A_1, \pi_1, I, r_a) \geq W(A_o, \pi_o, I, r_a).$$

Therefore, the amount of work done by A_o over $[r_a, t_o)$ is strictly greater than the amount of work done by A_1 over the same interval.

Let x_ℓ denote the cumulative length of time over the interval $[r_a, t_o)$ during which A_1 is executing on ℓ processors, $1 \leq \ell \leq m(\pi_1)$. Hence,

$$t_o - r_a = x_1 + x_2 + \cdots + x_{m(\pi_1)}.$$

We make the following two observations:

- Since A_1 is a work-conserving scheduling algorithm, job J_a, which has not completed by instant t_o in the schedule generated by A_1, must be executing at all time-instants during which some processor is idled by A_1. During the instants at which ℓ processors are nonidling, $\ell < m(\pi_1)$, all these nonidled processors have computing capacity $\geq s_\ell(\pi_1)$—this follows from the definition of "work-conserving" and the fact that A_1 is a work-conserving algorithm. Therefore, it follows that job J_a has executed for at least $(\sum_{j=1}^{m(\pi_1)-1} x_j s_j(\pi_1))$ units by time t_o in the schedule generated by A_1 on π_1, while it could have executed for at most $s_1(\pi_o)(\sum_{j=1}^{m(\pi_1)} x_j)$ units in the schedule generated by Algorithm A_o on π_o. We therefore have

$$\sum_{j=1}^{m(\pi_1)-1} x_j s_j(\pi_1) < s_1(\pi_o)\left(\sum_{j=1}^{m(\pi_1)} x_j\right). \tag{4}$$

  Multiplying both sides of (4) above by $\lambda(\pi_1)$ and noting that

$$\begin{aligned}\left(x_j s_j(\pi_1)\lambda(\pi_1)\right) &\geq \left(x_j s_j(\pi_1)\frac{\sum_{k=j+1}^{m(\pi_1)} s_k(\pi_1)}{s_j(\pi_1)}\right)\\ &= x_j\left(\sum_{k=j+1}^{m(\pi_1)} s_k(\pi_1)\right),\end{aligned}$$

  we obtain

$$\sum_{j=1}^{m(\pi_1)-1}\left(x_j\left(\sum_{k=j+1}^{m(\pi_1)} s_k(\pi_1)\right)\right) < s_1(\pi_o)\lambda(\pi_1)\left(\sum_{j=1}^{m} x_j\right). \tag{5}$$

- The total amount of work done by A_1 executing on π_1 during $[r_a, t_o)$ is given by

$$\sum_{j=1}^{m}\left(x_j\sum_{k=1}^{j} s_k(\pi_1)\right),$$

  while the total amount of work done by A_o executing on π_o during this same interval is bounded from above by the capacity of π_o and is, hence, $\leq (\sum_{j=1}^{m} x_j)\cdot S(\pi_o)$. We thus obtain the inequality

$$\sum_{j=1}^{m}\left(x_j\sum_{k=1}^{j} s_k(\pi_1)\right) < \left(\sum_{j=1}^{m(\pi_1)} x_j\right)\cdot S(\pi_o). \tag{6}$$

Adding (5) and (6), we obtain

$$\begin{aligned}&\sum_{j=1}^{m}\left(x_j\sum_{k=1}^{j} s_k(\pi_1)\right) + \sum_{j=1}^{m(\pi_1)-1}\left(x_j\left(\sum_{k=j+1}^{m(\pi_1)} s_k(\pi_1)\right)\right)\\ &< \left(\sum_{j=1}^{m} x_j\right)(s_1(\pi_o)\lambda(\pi_1) + S(\pi_o))\\ &\equiv x_{m(\pi_1)}S(\pi_1) + \sum_{j=1}^{m(\pi_1)-1}\left[x_j(S(\pi_1))\right]\\ &< \left(\sum_{j=1}^{m(\pi_1)} x_j\right)(s_1(\pi_o)\lambda(\pi_1) + S(\pi_o))\\ &\equiv x_{m(\pi_1)}S(\pi_1) + \sum_{j=1}^{m(\pi_1)-1}(x_j S(\pi_1))\\ &< \left(\sum_{j=1}^{m(\pi_1)} x_j\right)(s_1(\pi_o)\lambda(\pi_1) + S(\pi_o))\\ &\equiv S(\pi_1)\cdot\left(\sum_{j=1}^{m(\pi_1)} x_j\right)\\ &< \left(\sum_{j=1}^{m(\pi_1)} x_j\right)(s_1(\pi_o)\lambda(\pi_1) + S(\pi_o))\\ &\equiv S(\pi_1) < s_1(\pi_o)\lambda(\pi_1) + S(\pi_o),\end{aligned}$$

which contradicts the assumption made in the statement of the lemma (2). □

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 certain

particular work-conserving algorithms can feasibly schedule a task set: It states that any collection of jobs I that is feasible on a uniform multiprocessor platform π_o will be scheduled to meet all deadlines by Algorithm EDF on any platform π_1 satisfying Condition 2 of Lemma 1.

Theorem 1. *Let I denote an instance of jobs that is feasible on uniform multiprocessor platform π_o. Let π_1 denote another uniform multiprocessor platform. If Condition 2 of Lemma 1 is satisfied by platforms π_o and π_1:*

$$S(\pi_1) \geq s_1(\pi_o) \cdot \lambda(\pi_1) + S(\pi_o),$$

then I will meet all deadlines when scheduled using the EDF algorithm executing on π_1.

Proof. As a consequence of π_o and π_1 satisfying Condition 2, it follows from Lemma 1 that the work done at any time-instant t by EDF scheduling I on π_1 is at least as much as the work done by that time-instant t by an optimal scheduling algorithm executing I on π_o:

$$W(\text{EDF}, \pi_1, I, t) \geq W(\text{opt}, \pi_o, I, t) \text{ for all } t \geq 0,$$

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

We now prove by induction that I is scheduled by EDF to meet all deadlines on π_1. The induction is on the number of jobs in I. Specifically, let $I_k \stackrel{\text{def}}{=} \{J_1, \ldots, J_k\}$ denote the k jobs of I with the highest EDF-priority.

Base case. Since I_o denotes the empty set, I_o can clearly be scheduled by EDF to meet all deadlines on π_1.

Induction step. Assume that EDF can schedule I_k on π_1 for some k and consider the EDF-generated schedule of I_{k+1} on π_1. 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_o. Since I is assumed to be feasible on π_o, it follows that I_{k+1} is also feasible on π_o and, hence, opt will schedule I_{k+1} on π_o to meet all deadlines. That is,

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

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

$$W(\text{EDF}, \pi_1, I_{k+1}, d_{k+1}) \geq W(\text{opt}, \pi_o, I_{k+1}, d_{k+1}) = \sum_{i=1}^{k+1} c_i.$$

Since the total execution requirement of all the jobs in I_{k+1} is $\sum_{i=1}^{k+1} c_i$, 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_1. The theorem follows. □

As an immediate corollary to Theorem 1 above, we obtain the result of Phillips et al. [8] concerning EDF-scheduling on identical multiprocessors:

Corollary 1. *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.*

Proof. If we require the platforms π_o and π_1 of the statement of Theorem 1 to each be comprised of m identical multiprocessors of speeds s_o and s_1 respectively, the parameter $\lambda(\pi_1)$ evaluates to $(m-1)$:

$$\lambda(\pi_1) \stackrel{\text{def}}{=} \max_{j=1}^{m} \left\{ \frac{\sum_{k=j+1}^{m} s_k(\pi_1)}{s_j(\pi_1)} \right\} = \max_{j=1}^{m} \left\{ \frac{(j-1)s_1}{s_1} \right\} = m - 1.$$

The condition $S(\pi_1) \geq \lambda(\pi_1) \cdot s_1(\pi_o) + S(\pi_o)$ from the statement of Theorem 1 is hence equivalent to

$$\begin{aligned} S(\pi_1) &\geq \lambda(\pi_1) \cdot s_1(\pi_o) + S(\pi_o) \\ &\equiv m \cdot s_1 \geq (m-1) \cdot s_o + m \cdot s_o \\ &\equiv m \cdot s_1 \geq (m - 1 + m) \cdot s_o \\ &\equiv s_1 \geq \left(2 - \frac{1}{m}\right) \cdot s_o, \end{aligned}$$

from which the corollary follows. □

As another direct corollary of Theorem 1, we obtain the well-known result that EDF is optimal upon uniprocessors:

Corollary 2. *If an instance is feasible upon a processing platform π_o with (total) computing capacity $S(\pi_o)$, then the instance is EDF-feasible upon a single processor of computing capacity $S(\pi_o)$.*

Proof. Suppose that real-time instance I is feasible upon platform π_o. By definition of the parameter λ, $\lambda(\pi_1) = 0$ for any uniform multiprocessor platform π_1 comprised of one processor (i.e., with $m(\pi_1) = 1$). Hence, for I to be EDF-feasible upon a uniprocessor π_1, the condition of Theorem 1 is equivalent to

$$\begin{aligned} &S(\pi_1) \geq s_1(\pi_o) \cdot \lambda(\pi_1) + S(\pi_o) \\ &\equiv S(\pi_1) \geq s_1(\pi_o) \cdot 0 + S(\pi_o) \\ &\equiv S(\pi_1) \geq S(\pi_o). \end{aligned}$$

□

Theorem 1 characterizes a uniform multiprocessor platform π_1 according to its parameter "$\lambda(\pi_1)$" (as defined in (1)) and relates the EDF-feasibility of a system, known to be feasible on some platform π_o, to the cumulative capacities of π_o and π_1, the speed s_1 of the fastest processor in π_o, and this parameter $\lambda(\pi_1)$ of platform π_1. Theorem 2 below asserts that, for this particular characterization of a uniform multiprocessor system, the bound represented by Theorem 1 is a tight one and EDF is optimal in the sense that no other online scheduling algorithm can make a better guarantee:

Theorem 2. *There exist uniform multiprocessor platforms π_o and π_1 and an instance I of hard-real-time jobs such that*

- *Instance I is feasible on platform π_o,*
- *The relationship between the various parameters of π_o and π_1 is as follows:*

$$S(\pi_1) = \lambda(\pi_1) \cdot s_1(\pi_o) + S(\pi_o) - \epsilon. \quad (7)$$

(where ϵ is assumed to be an arbitrarily small positive number, i.e., the condition of Theorem 1 is not satisfied by an arbitrarily small amount ϵ).

- *Instance I is infeasible on uniform multiprocessor platform π_1.*

Proof. Let $\eta = \frac{\epsilon}{2}$. Consider a system $\pi_o = [1, 1]$ comprised of two identical unit-speed processors, an instance $I = \{(0, 1, 1), (0, 1, 1)\}$ of two jobs each arriving at time zero and needing to execute for one unit by a deadline equal to one, and a platform $\pi_1 = [\frac{2-\eta}{1+\eta}, \frac{\eta(2-\eta)}{1+\eta}]$ comprised of one processor of speed $\frac{2-\eta}{1+\eta}$ and another processor of speed $\frac{\eta(2-\eta)}{1+\eta}$. Observe that

$$\begin{aligned}\lambda(\pi_1) \cdot s_1(\pi_o) + S(\pi_o) - \epsilon &= \eta + 2 - \epsilon = \frac{\epsilon}{2} + 2 - \epsilon \\ &= 2 - \eta = S(\pi_1).\end{aligned}$$

Therefore, (7) is satisfied; however, it is easy to see that there is no way to meet both jobs' deadlines upon π_1 since instance I requires two units of execution over the interval $[0, 1)$, while platform π_1 can accommodate only $(2 - \eta)$ units of execution over this interval. □

4 The Robustness of EDF upon Uniform Multiprocessors

In choosing a scheduling framework for use in real-time application systems, an important requirement is that the framework be *stable* to "positive" changes in the operating environment. For instance, a real-time system that has been analyzed assuming a certain workload and found to meet all deadlines should continue to meet all deadlines if the workload turns out to be less than assumed during analysis; similarly, with respect to platform computing capacity, if all deadlines are met upon a given processor platform and we replace the platform with a more powerful one, it is desirable that all deadlines continue to be met. Such properties of a scheduling framework are often referred to as robustness properties and frameworks that possess these properties are said to be **robust**.

Robustness of this form is very desirable in hard-real-time scheduling frameworks. For instance, robustness with respect to **variation in workload** is implicitly assumed in many feasibility analysis algorithms based upon *worst-case execution time*; this is the principle that allows us to consider just one (or a few) runtime behaviors of a real-time system, instead of infinitely many possible behaviors, during system analysis to determine whether all deadlines will be met. Robustness with respect to **platform capacity** is desirable for several reasons:

- In a robust scheduling framework, we can perform system analysis with only lower bounds upon the computing capacity of (components of) the platform, rather than the exact capacity. (While this may seem a nonissue at first—after all, it is not unreasonable to assume that hardware components of a real-time systems will be exactly calibrated—the fact is that computing capacity of a platform is subject to minor fluctuations during runtime, such as that caused by variations in the voltage supplied to a CPU.)
- Hardware components in a robust framework may be upgraded without requiring rigorous reanalysis of the entire system.
- The design of a feasible system is generally easier if the framework is robust—if the design process is considered as a state-space search problem, there are *regions* of feasible solutions in the design space that include each feasible platform and all more powerful ones. In nonrobust frameworks, each feasible solution may be a single point in the design space and, hence, generally more difficult to find.

With respect to workload, it follows directly from the proofs of Lemma 1 and Theorem 1 that EDF upon uniform multiprocessors is indeed robust: If an instance is proven to meet all deadlines assuming worst-case execution requirements, then it follows that all deadlines will be met, even if some of the jobs have actual execution requirements less than their worst-case requirements.

While Theorem 1 provides a simple sufficient test for determining if a real-time instance is EDF-feasible upon a given uniform multiprocessor platform, we show below (Example 1) that this test does not indicate whether EDF-scheduling upon multiprocessors is robust or not; indeed, this test itself is not robust in the sense that, while an instance may be shown to be EDF-feasible upon a particular platform using this test, the test may fail to show that this same instance is EDF-feasible upon a more powerful platform.

Example 1. Consider the two uniform multiprocessor platforms π' and π'', with $m(\pi') = 5$, $m(\pi'') = 4$, and the following processor speeds:

π': $s_1(\pi') = 5$; $s_2(\pi') = 1$; $s_3(\pi') = 1$; $s_4(\pi') = 1$; and $s_5(\pi') = 1$

π'': $s_1(\pi'') = 5$; $s_2(\pi'') = 1$; $s_3(\pi'') = 1$; and $s_4(\pi'') = 1$

It may be verified that $S(\pi') = 9.0$ and $\lambda(\pi') = 3.0$; while $S(\pi'') = 8$ and $\lambda(\pi'') = 2$.

Suppose that a collection of hard-real-time jobs I is known to be feasible upon a uniform multiprocessor platform π_o satisfying $S(\pi_o) = 5.5$ and $s_1(\pi_o) = 1.25$. Is I EDF-feasible upon π' and π''?

Substituting in (2) for π', we obtain

$$\begin{aligned}&S(\pi') \geq s_1(\pi_o) \cdot \lambda(\pi') + S(\pi_o) \\ &\equiv 9.0 \geq 1.25 \cdot 3 + 5.5 \\ &\equiv 9.0 \geq 9.25 \\ &\equiv \text{false}.\end{aligned}$$

For π'', on the other hand,

$$\begin{aligned} & S(\pi'') \geq s_1(\pi_o) \cdot \lambda(\pi'') + S(\pi_o) \\ & \equiv 8 \geq 1.25 \cdot 2 + 5.5 \\ & \equiv 8 \geq 8 \\ & \equiv \text{true}. \end{aligned}$$

That is, the test of Theorem 1 reveals that the real-time instance I is EDF-feasible upon π'', but does not show that I is EDF-feasible upon π'.

Intuitively, however, π' is a more "powerful" machine than π'', in the example above, as formalized by the following definition:

Definition 5. *Uniform multiprocessor platform π_1* **dominates** *uniform multiprocessor platform π_2 (denoted as $\pi_1 \stackrel{\triangleright}{-} \pi_2$) if*

1. $m(\pi_2) \leq m(\pi_1)$ *and*
2. $\forall i,\ 1 \leq i \leq m(\pi_2),\ s_i(\pi_2) \leq s_i(\pi_1)$.

(That is, π_2 contains no more processors than π_1 does and the ith fastest processor in π_2 is no faster than the ith fastest processor in π_1, for all i.)

For a scheduling framework to be robust with respect to platforms, it is necessary that any instance that can be successfully scheduled upon a platform π should be successfully scheduled upon any platform that dominates π. The following theorem asserts that EDF upon uniform multiprocessors makes such a guarantee and, hence, is robust:

Theorem 3. *Let I denote an instance of hard-real-time jobs that is EDF-feasible upon a uniform multiprocessor platform π_o. Let π denote any uniform multiprocessor platform such that $\pi \stackrel{\triangleright}{-} \pi_o$. Then, I is EDF-feasible upon π.*

Proof. In order to prove this theorem, we will prove that, for all time-instants t and for all jobs $J \in I$,

$$W(\text{EDF}, \pi, I, \{J\}, t) \geq W(\text{EDF}, \pi_o, I, \{J\}, t). \tag{8}$$

That is, at each instant in time, each job receives at least as much execution when scheduled by EDF on π as when scheduled by EDF on π_o. The theorem then immediately follows: Since I is EDF-feasible upon π_o, each job $J \in I$ receives its entire execution requirement by its deadline when EDF-scheduled upon π_o; consequently, it follows that each job $J \in I$ receives its entire execution requirement by its deadline when EDF-scheduled upon π.

The proof of (8) is by contradiction. Suppose that it is not true, i.e., there is some collection I of jobs that is EDF-feasible upon π_o, some job $J \in I$, and some time-instant by which EDF executing I on π has executed J for strictly less than EDF executing I on π_o. Let t_o denote the earliest such instant and $J_a = (r_a, c_a, d_a)$ denote some job in I such that

$$W(\text{EDF}, \pi, I, \{J_a\}, t_o) < W(\text{EDF}, \pi_o, I, \{J_a\}, t_o).$$

Let σ_o and σ denote the schedules generated by EDF executing I on π_o and π, respectively.

Let Δ denote an arbitrarily small positive real number: $\Delta \to 0^+$. Since t_o is, by definition, the earliest instant at which $W(\text{EDF}, \pi, I, \{J_a\}, t_o) < W(\text{EDF}, \pi_o, I, \{J_a\}, t_o)$, it must be the case that J_a is selected for execution upon the kth fastest processor over $[t_o - \Delta, t_o)$ in σ_o for some $k \leq m(\pi_o)$, but is not selected for execution upon any of the k fastest processors over the same interval in σ. Hence, there must be some job J_b, with priority strictly greater than J_a, that is selected for execution during $[t_o - \Delta, t_o)$ in σ, but not in σ_o. For this to be true, it must be the case that J_b completes execution by instant $t_o - \Delta$ in σ_o but not in σ, from which it follows that

$$W(\text{EDF}, \pi, I, \{J_b\}, t_o - \Delta) < W(\text{EDF}, \pi_o, I, \{J_b\}, t_o - \Delta).$$

But, this contradicts the definition of t_o as the earliest instant at which some job is executed less in σ than in σ_o.□

5 An EDF-Feasibility Analysis Algorithm

In this section, we exploit our knowledge that EDF-scheduling upon uniform multiprocessors is robust with respect to platform computing capacity to derive an algorithm for determining whether a real-time instance, known to be feasible upon some uniform multiprocessor platform π_o, is EDF-feasible upon a given uniform multiprocessor platform π. This algorithm, like the EDF-feasibility test of Theorem 1, is a *sufficient*, rather than exact, test; however, this algorithm is a marked improvement over the EDF-feasibility test of Theorem 1 in that it is far less likely to report false negatives.[3] (In Example 1, for instance, this algorithm correctly determines that the real-time instance considered is EDF-feasible upon platform π', while we have already seen that the test of Theorem 1 fails to determine this fact.) In addition, this EDF-feasibility analysis algorithm is robust: If the algorithm determines that some real-time instance is EDF-feasible upon a particular platform π, then it is guaranteed to determine that the instance is EDF-feasible upon all platforms π' satisfying $\pi' \stackrel{\triangleright}{-} \pi$. In Section 5.1 below, we first derive the theory necessary for developing our algorithm; the algorithm itself is described in Section 5.2.

5.1 Clean Domination

In this section, we will develop the theory necessary to design our EDF-feasibility analysis algorithm. For the remainder of this section, let us assume that we are given that some real-time job instance I is feasible upon uniform multiprocessor platform π_0 and let

$$\begin{aligned} a &\stackrel{\text{def}}{=} s_1(\pi_0) \ - \text{the speed of the fastest processor in } \pi_0 \\ b &\stackrel{\text{def}}{=} S(\pi_0) - \text{the cumulative speed of } \pi_0. \end{aligned}$$

We have available to us a uniform multiprocessor platform π and wish to determine whether I is EDF-feasible upon π.

Suppose that one cannot directly conclude from Theorem 1 that I is feasible upon π, i.e., it turns out that $(S(\pi) < \lambda(\pi) \cdot a + b)$. By Theorem 3, we could instead attempt to identify a uniform multiprocessor platform π_1, for which

3. A feasibility-analysis test reports a **false negative** if it reports that a feasible system is not known to be so and it reports a **false positive** if it reports that an infeasible system is feasible. False negatives may lead to inefficient use of resources; however, false positives are far worse in that they may result in unacceptable deadline-misses during runtime. Hence, false-negative feasibilty-analysis algorithms are commonly used if an exact algorithm is not available, but false-positive algorithms are not useful.

$$(\pi \stackrel{\triangleright}{-} \pi_1) \text{ and } (S(\pi_1) \geq \lambda(\pi_1) \cdot a + b),$$

however, there are, in general, infinitely many π_1 for which $\pi \stackrel{\triangleright}{-} \pi_1$ holds and it is impractical to exhaustively test all such machines for the above condition. The following theorem narrows our search: It shows that if there is a π_1 dominated by π upon which our real-time instance can be shown, by Theorem 1, to be EDF-feasible, then there is another machine —π_2—also dominated by π upon which the real-time instance can be shown to be EDF-feasible, which further satisfies the property that there is an index k such that:

1. The $(k-1)$ fastest processors in π_2 are as fast as possible (subject to $\pi \stackrel{\triangleright}{-} \pi_2$):
$$\forall \ell : 1 \leq \ell < k : s_\ell(\pi_2) = s_\ell(\pi).$$
2. The speed of the kth-fastest processor in π_2 satisfies
$$0 \leq s_k(\pi_2) \leq s_k(\pi),$$
and
3. The remaining (slowest) processors in π_2 are of speed zero.

Let us say that such a π_2 is *cleanly dominated* by π, with *clean domination index k*:

Definition 6 ($\stackrel{\star}{\underset{\triangleright}{-}}$). *Uniform multiprocessor platform π* **cleanly dominates** *uniform multiprocessor platform π' (denoted $\pi \stackrel{\star}{\underset{\triangleright}{-}} \pi'$) with* **clean domination index** *k, $1 \leq k \leq m(\pi')$, if and only if*

1. $\pi \stackrel{\triangleright}{-} \pi'$ *and*
2. $$\begin{array}{lll} \forall i, & 1 \leq i < k, & s_i(\pi') = s_i(\pi), \quad \text{and} \\ \forall i, & k < i \leq m(\pi'), & s_i(\pi') = 0. \end{array}$$

Theorem 4. *Given uniform multiprocessor platform π and constants a and b, if there is a uniform multiprocessor machine π_1 such that*

$$(\pi \stackrel{\triangleright}{-} \pi_1) \bigwedge (S(\pi_1) \geq \lambda(\pi_1) \cdot a + b), \tag{9}$$

then there is a uniform multiprocessor machine π_2 such that

$$(\pi \stackrel{\star}{\underset{\triangleright}{-}} \pi_2) \bigwedge (S(\pi_2) \geq \lambda(\pi_2) \cdot a + b). \tag{10}$$

Proof. In order to prove its correctness, we find it convenient to measure how far from being cleanly dominated a uniform multiprocessor machine is by another. Accordingly, let us define a measure of this distance:

Definition 7 ($\ddot{\mathrm{D}}$). *Consider uniform multiprocessor platforms π, π' such that $\pi \stackrel{\triangleright}{-} \pi'$. Let $f \stackrel{\text{def}}{=} \min\{i | (s_i(\pi') < s_i(\pi)\}$ and $\ell \stackrel{\text{def}}{=} \max\{i | (s_i(\pi') > 0\}$. (That is, the fth fastest processor is the fastest processor in π' whose speed is strictly less than the speed of the corresponding processor in π and the ℓth fastest processor is the slowest processor in π' with a nonzero speed.) Define $\ddot{\mathrm{D}}(\pi, \pi')$ as follows:*

$$\ddot{\mathrm{D}}(\pi, \pi') \stackrel{\text{def}}{=} (\ell - f).$$

Lemma 4.1. *If $\pi \stackrel{\triangleright}{-} \pi'$ and $\ddot{\mathrm{D}}(\pi, \pi') = 0$, then $\pi \stackrel{\star}{\underset{\triangleright}{-}} \pi'$.*

Proof of Lemma 4.1. Suppose that the antecedents are true: $\pi \stackrel{\triangleright}{-} \pi'$ and $\ddot{\mathrm{D}}(\pi, \pi') = 0$. Then, $f = \ell$, where f and ℓ are as defined in Definition 7. By setting the clean domination index ("k" of Definition 6) equal to this value f, we see that the conditions of Definition 6 are satisfied. □

We are now ready to prove Theorem 4. Let us assume that the antecedents of the theorem are true: There is a π_1 satisfying Condition 9. We consider the quantity $\ddot{\mathrm{D}}(\pi, \pi_1)$:

- if $\ddot{\mathrm{D}}(\pi, \pi_1) = 0$, then, by Lemma 4.1, π_1 can itself play the role of π_2 in Condition 10 and the consequent of the theorem holds.
- Else, $\ddot{\mathrm{D}}(\pi, \pi_1) > 0$. We will derive below (the attributes of) a multiprocessor platform $\hat{\pi}_1$ such that
$$(\pi \stackrel{\triangleright}{-} \hat{\pi}_1) \bigwedge (S(\hat{\pi}_1) \geq \lambda(\hat{\pi}_1) \cdot a + b) \bigwedge (\ddot{\mathrm{D}}(\pi, \hat{\pi}_1) \leq \ddot{\mathrm{D}}(\pi, \pi_1) - 1).$$
Thus, $\hat{\pi}_1$ is a platform that satisfies the conditions of the antecedent and is "closer" to being cleanly dominated by π in the sense that is measured by the $\ddot{\mathrm{D}}$ parameter. The proof of the theorem then follows by repeatedly applying this argument, with $\hat{\pi}_1$ of one iteration playing the role of π_1 in the succeeding iteration.

If $\ddot{\mathrm{D}}(\pi, \pi_1) = 0$, then $\pi \stackrel{\star}{\underset{\triangleright}{-}} \pi_1$ and we are done. Otherwise, we define a new uniform multiprocessor machine $\hat{\pi}_1$ from π_1, by "transferring" an amount $\min\{s_f(\pi) - s_f(\pi_1), s_\ell(\pi_1)\}$ of processor speed from the ℓth fastest processor to the fth fastest processor; i.e., $\hat{\pi}_1$ consists of $m(\pi_1)$ processors, and

$$s_i(\hat{\pi}_1) = \begin{cases} s_f(\pi_1) + \min\{s_f(\pi) - s_f(\pi_1), s_\ell(\pi_1)\}, & \text{for } i = f \\ s_\ell(\pi_1) - \min\{s_f(\pi) - s_f(\pi_1), s_\ell(\pi_1)\}, & \text{for } i = \ell \\ s_i(\pi_1), & \text{for all other } i. \end{cases} \tag{11}$$

The following facts are easily proven:

Fact 1: $\ddot{\mathrm{D}}(\hat{\pi}_1) \leq \ddot{\mathrm{D}}(\pi_1) - 1$ since at least one of $(s_f(\hat{\pi}_1) = s_f(\pi))$ or $(s_\ell(\hat{\pi}_1) = 0)$ holds;

Fact 2: $S(\hat{\pi}_1) = S(\pi_1)$ since we have simply "transferred" processor speed from one processor to the other; and

Fact 3: $\lambda(\hat{\pi}_1) \leq \lambda(\pi_1)$. To see why this is so, observe that, as defined by (1)

$$\lambda(\pi_1) \equiv \max_{i=1}^{m(\pi_1)} \left\{ \frac{\sum_{j=i+1}^{m(\pi_1)} s_j(\pi_1)}{s_i(\pi_1)} \right\};$$

$$\lambda(\hat{\pi}_1) \equiv \max_{i=1}^{m(\pi_1)} \left\{ \frac{\sum_{j=i+1}^{m(\pi_1)} s_j(\hat{\pi}_1)}{s_i(\hat{\pi}_1)} \right\}.$$

Now, for $\ell < i \leq m(\pi_1)$,

$$\frac{\sum_{j=i+1}^{m(\pi_1)} s_j(\pi_1)}{s_i(\pi_1)} = \frac{\sum_{j=i+1}^{m(\hat{\pi}_1)} s_j(\hat{\pi}_1)}{s_i(\hat{\pi}_1)} = 0$$

since $s_i(\pi_1) = s_i(\hat{\pi}_1) = 0$ for $\ell < i \leq m(\pi_1)$, and the numerators are therefore equal to zero. Similarly, for $1 \leq i < f$,

$$\frac{\sum_{j=i+1}^{m(\pi_1)} s_j(\pi_1)}{s_i(\pi_1)} = \frac{\sum_{j=i+1}^{m(\hat{\pi}_1)} s_j(\hat{\pi}_1)}{s_i(\hat{\pi}_1)}$$

since $\sum_{j=i+1}^{m(\pi_1)} s_j(\pi_1) = \sum_{j=i+1}^{m(\pi_1)} s_j(\hat{\pi}_1)$. Additionally,

$$\frac{\sum_{j=f+1}^{m(\pi_1)} s_j(\pi_1)}{s_f(\pi_1)} > \frac{\sum_{j=f+1}^{m(\hat{\pi}_1)} s_j(\hat{\pi}_1)}{s_f(\hat{\pi}_1)}$$

since $s_f(\pi_1) < s_f(\hat{\pi}_1)$ and $\sum_{j=f+1}^{m(\pi_1)} s_j(\pi_1) > \sum_{j=f+1}^{m(\pi_1)} s_j(\hat{\pi}_1)$.

Thus, it turns out that each term in the "max" defining $\lambda(\pi_1)$ is $\geq$ the corresponding term in the "max" defining $\lambda(\hat{\pi}_1)$; consequently, $\lambda(\hat{\pi}_1) \leq \lambda(\pi_1)$.

The second conjunct of Condition 9 asserts that $S(\pi_1) \geq \lambda(\pi_1) \cdot a + b$. We saw above that $S(\hat{\pi}_1) = S(\pi_1)$ (Fact 2) and $\lambda(\hat{\pi}_1) \leq \lambda(\pi_1)$ (Fact 3). We can therefore conclude that $S(\hat{\pi}_1) \geq \lambda(\hat{\pi}_1) \cdot a + b$ and $\hat{\pi}_1$ is therefore a uniform multiprocessor machine which also satisfies Condition 9 and is closer than π_1 to being cleanly dominated by π (in the sense that $\ddot{\mathrm{D}}(\pi, \hat{\pi}_1) \leq \ddot{\mathrm{D}}(\pi, \pi_1)$). □

5.2 An Algorithm

We now describe a program we have implemented based upon the ideas developed in Section 5.1, for determining whether a real-time instance I, known to be feasible upon a uniform multiprocessor platform in which the fastest processor has computing capacity a and the total computing capacity is b, is EDF-feasible upon a given uniform multiprocessor platform π. The code is given in Fig. 1.

By Theorem 4, it follows that the real-time instance I is EDF-feasible upon π if there is some π_1 satisfying $\pi \overset{\star}{\triangleright} \pi_1$ and

$$S(\pi_1) \geq a\lambda(\pi_1) + b. \tag{12}$$

Our algorithm attempts to determine the parameters of some such π_1; if found, then this π_1 bears witness to the EDF-feasibility of the real-time instance upon π, while the algorithm reports failure if it fails to find such a π_1. This is how it does so. At the top level (the function "main( )" in Fig. 1), the algorithm checks whether there is a π_1 satisfying (12) which is cleanly dominated by π with clean domination index k, for each integer value of k in the range $1 \leq m(\pi)$—this is achieved by the calls to procedure "test(k)."

Each call to "test(k)" starts out with the speed of π_1's kth processor set equal to zero (the speeds of all of π_1's other processors are fixed by the requirement that $\pi \overset{\star}{\triangleright} \pi_1$) and successively increments this speed until either 1) Condition 12 is satisfied at the current speed or 2) the speed exceeds $s_k(\pi)$ (in which case, we can conclude that there is no π_1 satisfying Condition 12 that is cleanly dominated by π with clean domination index k).

For a chosen value of $s_k(\pi_1)$ (initially, zero), "test(k)" computes $S(\pi_1)$ and $\lambda(\pi_1)$ (by calling procedure "compute( )"), according to the relationships:

$$S(\pi_1) = s_1(\pi_1) + s_2(\pi_1) + \cdots + s_k(\pi_1)$$
$$\lambda(\pi_1) = \max_{i=1}^{k-1} \left\{ \frac{\sum_{j=i+1}^{k} s_j(\pi)}{s_i(\pi)} \right\}. \tag{13}$$

In addition, procedure compute( ) determines the index ℓ, corresponding to the value of i that maximizes the right-hand side of (13).

To determine whether π_1 satisfies (12), our algorithm computes the difference between $S(\pi_1)$ and $(a \cdot \lambda(\pi_1) + b)$. If this difference is nonnegative, then it has determined that π_1 with processor-speeds as currently assigned is cleanly dominated by π and satisfies Condition 12; consequently, this π_1 bears witness to the fact that instance I is EDF-feasible upon platform π and we are done. Else, the algorithm increases the computing capacity of the kth processor. Notice that doing so can only cause $\lambda(\pi_1)$ to *in*crease—this follows from the fact that $s_k(\pi_1)$ appears only in the numerators of the terms in the righthand side of (13).

We now attempt to determine the amount by which $s_k(\pi_1)$ should be increased in order that (12) be satisfied . By the definition of the index ℓ, $\lambda(\pi_1)$ prior to the increase is given by

$$\lambda(\pi_1) = \frac{N_\ell}{s_\ell(\pi_1)},$$

where $N_\ell \overset{\text{def}}{=} \sum_{j=\ell+1}^{k} s_j(\pi)$. Let π_1^{new} denote the uniform multiprocessor platform that is identical to π_1, except that $s_k(\pi_1^{\text{new}}) = s_k(\pi_1) + \Delta$. Then, $S(\pi_1^{\text{new}})$ equals $S(\pi_1) + \Delta$, while $\lambda(\pi_1^{\text{new}})$ is $\geq \frac{N_\ell + \Delta}{s_\ell(\pi_1)}$, i.e., $\lambda(\pi_1^{\text{new}}) \geq \lambda(\pi_1) + \frac{\Delta}{s_\ell(\pi_1)}$.

If $\lambda(\pi_1^{\text{new}})$ does in fact equal $\lambda(\pi_1) + \frac{\Delta}{s_\ell(\pi_1)}$, then, in order for (12) to be satisfied by π_1^{new}, we would need that

$$S(\pi_1^{\text{new}}) \geq a\lambda(\pi_1^{\text{new}}) + b$$
$$\text{i.e., } S(\pi_1) + \Delta \geq a\left(\lambda(\pi_1) + \frac{\Delta}{s_\ell(\pi_1)}\right) + b,$$

from which it follows that

$$\Delta \geq (a\lambda(\pi_1) + b - S(\pi_1)) / \left(1 - \frac{a}{s_\ell(\pi_1)}\right). \tag{14}$$

In procedure test( ), therefore, we increase $s_k(\pi_1)$ by an amount equal to

$$\frac{a\lambda(\pi_1) + b - S(\pi_1)}{1 - \frac{a}{s_\ell(\pi_1)}},$$

and then test whether this enhanced platform satisfies (12). There are three possibilities:

- If increasing the value of $s_k(\pi_1)$ by this amount would cause it to *exceed* $s_k(\pi)$, then the resulting π_1^{new} would not be dominated by π. But, we saw above that this is the minimum amount by which $s_k(\pi_1)$ must be increased if Condition 12 is to be satisfied. Hence, we can conclude that it is not possible to have

```
//EDF-FEASIBILITY UPON UNIFORM MULTIPROCESSORS
//
//Determines whether a task system, known to be feasible on a uniform
//xproc platform in which total speed=b and fastest speed=a, is
//edf-feasible on an m-proc platform with speeds s0, s1, ..., s[m-1].
//
//INPUT: Reads in data from standard input, in the following format:
//m
//s0 s1 ... s[m-1]
//ab
//
//ALGORITHM: -- Is based upon the observation that an optimal solution
//exhibits "clean" domination; i.e., it has the fastest (k-1) procs at
//full speed, the next one at (perhaps) partial speed, and the
//remaining procs at zero speed. We exhaustively go through the
//different possible values for k.
//
//COMPLEXITY: No worse than O(m^3)
//
//
//OUTPUT: "Not known to be EDF-feasible" if fails to find suitable
//speeds. Witness solution, if EDF-feasible
//*********************************************************************
#include<iostream>
//All global variables (poor programming practice!!)
#define M 25 //[maximum] number of processors
double s[M+1]; //processor speeds -- input
double a, b; //a: max speed of feas platform -- input
             //b: total speed of feas platform -- input
int m; //actual number of processors -- input

int k; //index being considered in compute() -- try each index as the
 //one defining the "clean domination" point
double s_temp; //speed currently assigned to the k'th processor (in
  //procedure "test()")
double nume[M+1]; //to compute lambda: need sum of lower speeds in
 //numerator
double ratio[M+1]; //to compute lambda: sum of lower speeds divided by
//proc's speed
double S; //S(\pi_1)
double lambda; //\lambda(\pi_1)
int ell; //index at which lambda is defined (the index which maximizes
 //ratio)
double diff; //(lambda(pi_1)*a+b) - S(pi_1) -- the difference
 //that must be made up by increasing s_k(pi_1)

void init(){//reading in the input (and echoing it onto the screen)
  cin>>m; cout<<"Pi has "<<m<<" processors of speeds \n\n";
  for(int i=1; i<=m; i++){
    cin>>s[i]; cout<<"\t"<<s[i];
  }
  cin>>a; cin>>b;
  cout<<"\n\t a and b are "<<a<<" and "<<b<<endl<<endl;
}

void compute(){//for given values of k and s_k(pi_1), computes
               //lambda, ell, and S
  nume[k-1]=s_temp;
  ratio[k-1]=nume[k-1]/s[k-1];
  for(int i=k-2; i>=1; i--){
    nume[i]=nume[i+1]+s[i+1];
    ratio[i]=nume[i]/s[i];
  }
  S=nume[1]+s[1]; //S(pi_1)=numerator used for the 1'th ratio,
 //pluss_1(pi_1)
  for(int i=ell+1; i<k; i++) //iteratively look for the largest
 //ratio, for lambda
    if(ratio[ell]<ratio[i]) ell=i;
  lambda=ratio[ell];
}

bool test(int k){//tests to see whether there is a "clean domination"
 //at the k'th processor
  s_temp=0.0; //we'll start out with the proc speed at 0, and
 //increase it if needed
  ell=1; //will be modified in compute()
  compute();
  diff=lambda*a+b-S;
  while(diff>0){//if diff<=0, then we have edf-feasibility.
 //crucial fact for time-cxty: each time we iterate, either ell
 //increases, or we get diff<=0
    double delta;
    if(s[ell]<=a) return 0; //if diff>0 and s[ell]<=a, we
 //cannot find a solution -- increasing s_k[pi_1] would increase the
 //value of lambda(pi_1) more than it would increase S(pi_1)
    delta=diff/(1-(a/s[ell]));
    s_temp+=delta;
    if(s_temp>s[k]) return 0; //cannot solve
    compute();
    diff=lambda*a+b-S;
  }//end of while
  return 1; //reached here ==> 's diff<=0, and we're done
}

void printOut(){
  cout<<"Is EDF-feasible: The processor speeds are \n\n";
  for(int i=1; i<k; i++) cout<<"\t"<<s[i];
  cout<<"\t"<<s_temp;
  for(int i=k+1; i<=m; i++) cout<<"\t"<<0;
  cout<<"\n\t S(pi_1) = "<<S<<endl;
  cout<<"\t lambda(pi_1) = "<<lambda<<endl;
  cout<<"\t lambda(pi_1)*a+b=";
  cout<<lambda*a<<"+"<<b<<"=";
  cout<<lambda*a+b<<endl;
}

int main(){
  init(); //read in the input
  if(s[1]>=b){//fastest processor used as uniproc suffices
    k=1; //clean domination point is at the 1st proc -- all other
 //procssors have speed 0
    s_temp=b; //the speed of the k'th proc in the clean domination
    S=b; //total speed
    printOut();
    exit(0);
  }
  for(k=2; k<=m; k++) //must use multiprocessor
    if(test(k)){//test(k) checks to see whether it is possible to build
      //a clean domination in which procs numbered >k have speed 0
      printOut();
      exit(0);
    }
  //made it here ==> couldn't find clean domination
  cout<<"Not known to be EDF-feasible\n";
}
```

Fig. 1. C++ implementation of algorithm for determining whether a real-time instance, known to be feasible upon a uniform multiprocessor with fastest processor having speed a and total computing capacity b, is EDF-feasible upon a uniform multiprocessor platform π.

a π_1 cleanly dominated by π with clean domination index k, which satisfies Condition 12.

- If increasing the value of $s_k(\pi_1)$ by this amount causes the λ-parameter of the resulting platform to continue to be defined at the index ℓ, then (12) will indeed be satisfied.
- If the value of ℓ changes (i.e., the index for which the rhs of (14) is maximized is not the same for π_1^{new} as it is for π_1), then $\lambda(\pi_1^{\text{new}})$ is strictly larger than $\lambda(\pi_1) + \frac{\Delta}{s_\ell(\pi_1)}$ and, hence, (12) may once again not be satisfied. In that case, the while loop of procedure test( ) would get executed again since the guard ("while (diff $>$ 0)") would return true and we would once again increment $s_k(\pi_1)$ and repeat the above argument.

Thus, we would iterate through the while loop, increasing the value assigned to the kth fastest processor, only if the third of these cases occurs. And, how many times can this while loop execute for a given value of k? Note that, in order for this third case to occur, the index ℓ which maximizes the rhs of (14) must be *larger* for π_1^{new} than it is for π_1: This follows from the observation that the numerator of each term in the rhs of (14) is increased by the same amount —Δ—but the denominators are in nonincreasing order with index. This observation immediately bounds the number of times we may iterate through the while loop of procedure test( ), increasing $s_k(\pi_1)$ each time, at $k-1$.

5.2.1 Runtime Complexity

The total time complexity of the implementation given in Fig. 1 is $\mathcal{O}(m^3)$, where m denotes the number of processors in the platform π_1 upon which EDF-feasibility is being tested. This follows from the observations that:

- There are m possible values of the index of clean domination k, all of which may need to be tested,
- For each index being tested, we may iterate through the while loop in procedure test( ) at most k times (since each iteration requires a different value of the index ℓ), and
- Each such iteration makes a call to the function compute( ) and each call takes time linear in m.

6 Conclusions

Knowledge of good online scheduling algorithms seems necessary in order to gain a deeper understanding of real-time scheduling within a particular model—witness the pervasive role played by EDF in many of the seminal papers on uniprocessor real-time scheduling theory. While EDF is known to be optimal in the uniprocessor context, it has been shown that no multiprocessor online scheduling algorithm can be optimal.

In this research, we have explored the design of suitable online scheduling algorithms for use in uniform multiprocessor platforms (since identical multiprocessors are a special case of uniform multiprocessors, our results hold for identical multiprocessors as well). Our major conclusion is that, despite its nonoptimality, *EDF is an appropriate algorithm to use for online scheduling on uniform multiprocessors.*

- For a particular (admittedly, very limited) characterization of uniform multiprocessors—by its λ parameter (1) and its cumulative computing capacity—EDF is an optimal algorithm (Theorem 2).
- Many of the advantages of using EDF in the uniprocessor context (efficient implementation [6], the fact that the exact computation requirements of jobs are not needed for making scheduling decisions, bounds upon the number of context switches, etc.) continue to hold in the multiprocessor case.
- A desirable property of scheduling frameworks is that they be robust—reducing the workload or increasing the resources available should not cause an otherwise well-behaved system to misbehave. We have established here the robustness of EDF upon uniform multiprocessor platforms.
- Based upon our robustness result, we have developed an algorithm for determining if a real-time instance is EDF-feasible upon a given uniform multiprocessor platform. Although this algorithm is a sufficient, rather than exact, test, experimental data (not reported in this document) indicates that the test is likely to prove useful in practice, in the design of actual real-time application systems.

Acknowledgments

The research is supported in part by the US National Science Foundation (Grant Nos. CCR-9972105, CCR-9988327, and ITR-0082866).

References

[1] M. Dertouzos, "Control Robotics: The Procedural Control of Physical Processors," *Proc. IFIP Congress,* pp. 807-813, 1974.
[2] M. Dertouzos and A.K. Mok, "Multiprocessor Scheduling in a Hard Real-Time Environment," *IEEE Trans. Software Eng.,* vol. 15, no. 12, pp. 1497-1506, Dec. 1989.
[3] K. Hong and J. Leung, "On-Line Scheduling of Real-Time Tasks," *Proc. Real-Time Systems Symp.,* pp. 244-250, Dec. 1988.
[4] B. Kalyanasundaram and K. Pruhs, "Speed Is as Powerful as Clairvoyance," *Proc. 36th Ann. Symp. Foundations of Computer Science (FOCS '95),* pp. 214-223, Oct. 1995.
[5] C. Liu and J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *J. ACM,* vol. 20, no. 1, pp. 46-61, 1973.
[6] A. Mok, "Task Management Techniques for Enforcing ED Scheduling on a Periodic Task Set," *Proc. Fifth IEEE Workshop Real-Time Software and Operating Systems,* pp. 42-46, May 1988.
[7] A.K. Mok, "Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment," PhD thesis, Laboratory for Computer Science, Massachusetts Inst. of Technology, 1983, Technical Report No. MIT/LCS/TR-297.
[8] C.A. Phillips, C. Stein, E. Torng, and J. Wein, "Optimal Time-Critical Scheduling via Resource Augmentation," *Proc. 29th Ann. ACM Symp. Theory of Computing,* pp. 140-149, May 1997.

Sanjoy Baruah received the PhD degree from the University of Texas at Austin in 1993. He is an associate professor in the Department of Computer Science at the University of North Carolina at Chapel Hill. 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. He is a member of the IEEE and the IEEE Computer Society.

Shelby Funk received two MS degrees, one in mathematics and one in computer science, from the University of North Carolina and the BS degree in mathematics from the University of Maryland at College Park. She is a PhD 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. She is a student member of the IEEE.

Joël Goossens received the MS degree in computer science in 1992, the MS degree in network and management in 1993, and the PhD degree in computer science in 1999, all from the Université Libre de Bruxelles, Belgium. He is currently an 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.