The non-preemptive scheduling of periodic tasks upon multiprocessors*

Sanjoy K. Baruah

The University of North Carolina

Abstract

The non-preemptive scheduling of periodic task systems upon processing platforms comprised of several identical processors is considered. The exact problem has previously been proven intractable even upon single processors; sufficient conditions are presented here for determining whether a given periodic task system will meet all deadlines if scheduled non-preemptively upon a multiprocessor platform using the earliest-deadline first scheduling algorithm.

Keywords: Non-preemptive scheduling; Periodic tasks; Multiprocessor systems; Earliest-deadline first; Feasibility analysis.

1 Introduction

Over the years, the **periodic task** model [7, 6] has proven remarkably useful for the modelling of recurring processes that occur in hard-real-time computer application systems. A periodic task τ_i is characterized by a 3-tuple of parameters – an initial arrival time $a(\tau_i)$, an execution requirement $e(\tau_i)$ and a period $p(\tau_i)$. Such a task generates an infinite sequence of *jobs*, each of which has an

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

execution requirement equal to $e(\tau_i)$, and a deadline that occurs $p(\tau_i)$ time units after its arrival time. The first such job generated by τ_i becomes available for execution at time-instant $a(\tau_i)$; consecutive jobs generated by τ_i become available exactly $p(\tau_i)$ time units apart. A periodic task system is comprised of a number of such periodic real-time tasks; with respect to a given scheduling algorithm A, a periodic task system is said to be *A-feasible* upon a specified processor platform if A schedules the system upon the specified platform to meet all deadline of all jobs. A periodic task system is said to be *feasible* upon a specified processor platform and under specified restrictions if there is an optimal algorithm observing the specified restrictions that can schedule the jobs generated by the tasks upon the specified platform in such a manner that all jobs of all tasks complete by their deadlines.

Non-preemptive scheduling. Scheduling algorithms may be either preemptive or nonpreemptive. In *nonpreemptive* scheduling, a job must be executed to completion once it starts execution; in *preemptive* scheduling, on the other hand, it is permitted that an executing job be interrupted prior to completion and its execution resumed later. Much of the prior research on the scheduling of periodic task systems has focused upon preemptive scheduling; with non-preemptive scheduling receiving considerably less attention. Jeffay, Stanat, and Martel [5] showed that the problem of determining whether a given periodic task system is non-preemptively feasible upon a single processor is intractable – NP-hard in the strong sense. Cai and Kong [2] further strengthened this intractability result by showing that the problem remains NP-hard in the strong sense even if all the periodic tasks have the same initial arrival time, and if all task periods are *harmonic* (i.e., each period exactly divides all periods larger than itself). Jeffay et al. [5] defined the notion of *non-concrete* periodic task systems, in which task start times are not know in advance. I.e., each periodic task in a non-concrete task system is specified by just two parameters, the execution

requirement and the period, with the initial arrival time remaining unknown until run-time. A non-concrete periodic task system is deemed feasible if it is feasible for every possible combination of start times. Howell and Venkatrao [4] showed that no polynomial-time scheduling algorithm could be an *optimal* scheduling algorithm for non-concrete periodic task systems, in the sense of always meeting all deadlines of all feasible non-concrete periodic task systems, unless P=NP. Jeffay et al. considered algorithms for scheduling non-concrete periodic task systems under the restriction that the algorithm is not allowed to idle a processor if there is some job awaiting execution (such scheduling algorithms are said to be "work-conserving"), and showed [5] that the non-preemptive earliest deadline first scheduling algorithm (Algorithm $\mathsf{EDF}_{\mathsf{np}}$), which chooses for execution at each instant in time the job needing execution with the smallest value of its deadline parameter, is an optimal scheduling algorithm for non-concrete periodic task systems that are feasible under the work-conserving restriction.

This research. As demonstrated by the results of Jeffay et al. [5] and Cai and Kong [2] discussed above, exact feasibility-analysis of periodic task systems upon non-preemptive systems is highly intractable, even upon uniprocessors. Consequently, our goal here is to obtain *sufficient*, rather than exact, conditions for feasibility, and our algorithm of choice is the non-preemptive earliest deadline first scheduling algoithm (Algorithm $\mathsf{EDF}_{\mathsf{np}}$). Our main contribution is a very efficient test, taking time linear in the representation of the input, that is sufficient for determining whether a given periodic task system will meet all deadlines if scheduled upon a specified number of identical processors by the multiprocessor extension of Algorithm $\mathsf{EDF}_{\mathsf{np}}$. Our test is particularly effective for a particular class of periodic task systems – those in which the largest execution requirement parameter is significantly smaller than the smallest period parameter.

Organization. The remainder of this paper is organized as follows. In Section 2, we formally define the periodic real-time task model, and introduce terminology and definitions that will be used in later sections. In Section 3, we present, and prove the correctness of, an efficient feasibility-test that is sufficient (although not necessary) for determining whether a given periodic task system will meet all deadlines if scheduled upon a multiprocessor platform using the non-preemptive earliest deadline first scheduling algorithm. In Section 4, we discuss some ramifications of our results from Section 3, and some straightforward extensions.

2 System model

A real-time task τ_i is characterized by a 3-tuple of parameters – an initial arrival time $a(\tau_i)$, an execution requirement $e(\tau_i)$ and a period $p(\tau_i)$. Such a real-time task may be periodic or sporadic; in either case, it generates an infinite sequence of jobs, each of which has an execution requirement equal to $e(\tau_i)$, and a deadline that occurs $p(\tau_i)$ time units after its arrival time. The first such job generated by τ_i "arrives" – i.e., becomes available for execution – at time-instant $a(\tau_i)$; the constraints placed upon the inter-arrival separation between consecutive jobs generated by τ_i differ depending upon whether τ_i is a periodic or a sporadic task:

- If τ_i is a periodic task, consecutive jobs generated by τ_i arrive exactly $p(\tau_i)$ time units apart.
- If τ_i is a sporadic task, consecutive jobs generated by τ_i arrive at least $p(\tau_i)$ time units apart.

Let $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ denote a periodic or sporadic real-time task systems. For each task τ_i, define its *utilization* $U(\tau_i)$ to be the ratio of its execution requirement to its period: $U(\tau_i) \stackrel{\text{def}}{=} e(\tau_i)/p(\tau_i)$. We define the utilization of task system to be the sum of the utilizations of all tasks in the system. Without loss of generality, we assume that $\min_{\tau_i \in \tau}\{a_i\} = 0$; i.e., that the first job generated by τ arrives at time-instant 0.

We consider the problem of scheduling τ on m identical processors, $m > 1$; without loss of generality, we assume that the processors have speed one (i.e., each processor can complete one unit of execution per time unit). Since we are studying non-preemptive scheduling, we require that each job execute contiguously upon a single processor: once a job begins execution upon a processor, it continues to execute upon that particular processor until it completes.

We now introduce some more terminology and notation concerning periodic and sporadic task systems; this terminology and notation is used in the following sections.

$$U_{\mathsf{sum}}(\tau) \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau} U(\tau_i) \tag{1}$$

$$U_{\mathsf{max}}(\tau) \stackrel{\text{def}}{=} \max_{\tau_i \in \tau} U(\tau_i) \tag{2}$$

$$e_{\mathsf{max}}(\tau) \stackrel{\text{def}}{=} \max_{\tau_i \in \tau} e(\tau_i) \tag{3}$$

$$p_{\mathsf{min}}(\tau) \stackrel{\text{def}}{=} \min_{\tau_i \in \tau} p(\tau_i) \tag{4}$$

That is, $U_{\mathsf{sum}}(\tau)$ denotes the utilization of task system τ, while $U_{\mathsf{max}}(\tau)$ denotes the largest utilization of any task in τ. The quantity $e_{\mathsf{max}}(\tau)$ denotes the largest execution requirment of any task in τ; since this represents the minimum blocking that may be experienced in non-preemptive schedules for τ, $e_{\mathsf{max}}(\tau)$ plays a crucial role in determining the feasibility of τ in non-preemptive environments. Very loosely speaking, $V(\tau_i, \tau)$, defined below, plays the role of the task utilization of task τ_i, when τ_i is subject to being blocked for an amount of time equal to $e_{\mathsf{max}}(\tau)$.

$$V(\tau_i, \tau) \stackrel{\text{def}}{=} \frac{e(\tau_i)}{p(\tau_i) - e_{\mathsf{max}}(\tau)} \tag{5}$$

$V_{\mathsf{sum}}(\tau)$ and $V_{\mathsf{max}}(\tau)$ play analogous roles to $U_{\mathsf{sum}}(\tau)$ and $U_{\mathsf{max}}(\tau)$, with the "V" parameter described (Equation 5) above playing the role of task utilization:

$$V_{\mathsf{sum}}(\tau) \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau} V(\tau_i, \tau) \tag{6}$$

$$V_{\mathsf{max}}(\tau) \stackrel{\text{def}}{=} \max_{\tau_i \in \tau} V(\tau_i, \tau) \tag{7}$$

3 $\mathsf{EDF}_{\mathsf{np}}$-feasibility analysis

In this section, we consider periodic task systems τ that satisfy the following constraint:

$$V_{\mathsf{sum}}(\tau) \leq m - (m-1) \times V_{\mathsf{max}}(\tau) \tag{8}$$

We prove (Lemma 1) that all such periodic task systems can be scheduled preemptively by an optimal scheduling algorithm in such a manner that all jobs of all tasks complete at least $e_{\mathsf{max}}(\tau)$ time units prior to their deadlines. We then show (Lemma 2) that Algorithm $\mathsf{EDF}_{\mathsf{np}}$ while executing τ will have performed at least as much work upon the k earliest-deadline jobs in τ by time $t+e_{\mathsf{max}}(\tau)$ as the optimal scheduling algorithm performs on these same jobs by time t, for all $t \geq 0$. We use these results to prove, in Theorem 1, that Algorithm $\mathsf{EDF}_{\mathsf{np}}$ meets all deadlines for τ. In Corollary 1, we use algebraic simplification techniques upon Condition 8 to obtain a utilization-based sufficient feasibility condition for periodic task systems τ in which $e_{\mathsf{max}}(\tau) \leq \rho \cdot p_{\mathsf{min}}(\tau)$, for some constant $\rho < 1$.

Lemma 1 Let τ denote a periodic task system satisfying Condition 8:

$$V_{\mathsf{sum}}(\tau) \leq m - (m-1) \times V_{\mathsf{max}}(\tau)$$

There is a processor-sharing schedule — henceforth denoted $\mathsf{opt}(\tau)$ — for τ upon m processors in which each job completes at least $e_{\mathsf{max}}(\tau)$ time units prior to its deadline.

Proof: The proof is by construction: the schedule $\mathsf{opt}(\tau)$ is constructed as follows. Let j denote any job generated by any task $\tau_i \in \tau$, with arrival time at a_j and deadline at $a_j + p(\tau_i)$. We assign a share $V(\tau_i, \tau)$ to job j over the interval $[a_j, a_j + p(\tau_i) - e_{\mathsf{max}}(\tau))$.

- In the task model, at most one job of each task may be active at any given instant. Hence, the total processor share thus allocated at any instant is $\leq \sum_{\tau_i \in \tau} V(\tau_i, \tau)$, which is, by definition, $\leq V_{\mathsf{sum}}(\tau)$. By Condition 8 above, this is $\leq m$; hence, such a schedule can indeed be constructed upon m processors.

- By definition, $V(\tau_i, \tau) = e(\tau_i)/(p(\tau_i) - e_{\mathsf{max}}(\tau))$; hence, the total amount of execution afforded job j over the interval $[a_j, a_j + p(\tau_i) - e_{\mathsf{max}}(\tau))$ is equal to $e(\tau_i)$, and it therefore completes execution $e_{\mathsf{max}}(\tau)$ time units prior to its deadline of $a_j + p(\tau_i)$.

Thus, we have constructed the schedule $\mathsf{opt}(\tau)$ for τ upon m processors, in which each job completes at least (in fact, exactly) $e_{\mathsf{max}}(\tau)$ time units prior to its deadline. ■

Some more definitions:

Definition 1 Let $\mathsf{EDF}_{\mathsf{np}}(\tau)$ denote the schedule generated by the non-preemptive earliest-deadline first scheduling algorithm while scheduling τ upon m processors.

Definition 2 For any task system τ and any integer $k \geq 0$, let $J(\tau, k)$ denote the k jobs generated by τ that are accorded the highest priority by the non-preemptive earliest-deadline first scheduling algorithm while scheduling τ. That is, $J(\tau, k)$ is the set of k earliest-deadline jobs generated by τ; in case of tied deadlines, ties are broken in the same manner as by Algorithm $\mathsf{EDF}_{\mathsf{np}}$.

Definition 3 For any schedule S, any subset J of the jobs being scheduled in S, and any time-instant $t \geq 0$, let $W(S, J, t)$ denote the amount of work done in schedule S upon jobs in J over the interval $[0, t)$.

■

In Lemma 2 below, we use techniques very similar to the ones introduced by Phillips et al. [8] to show that $\mathsf{EDF}_{\mathsf{np}}(\tau)$ never "lags" $\mathsf{opt}(\tau)$ in terms of the work done on high-priority jobs by too much. More precisely, we prove that $\mathsf{EDF}_{\mathsf{np}}(\tau)$ does at least as much work upon the jobs in $J(\tau, k)$ by time $t + e_{\mathsf{max}}(\tau)$ as $\mathsf{opt}(\tau)$ does by time t, for all $t \geq 0$ and for all $k \geq 0$.

Lemma 2 Let τ denote a task system satisfying Condition 8 above:

$$V_{\mathsf{sum}}(\tau) \leq m - (m-1) \times V_{\mathsf{max}}(\tau) \ .$$

For all $k \geq 0$ and all time-instants $t \geq 0$

$$W(\mathsf{EDF}_{\mathsf{np}}(\tau), J(\tau, k), t + e_{\mathsf{max}}(\tau)) \geq W(\mathsf{opt}(\tau), J(\tau, k), t) \tag{9}$$

Proof: Our proof is by contradiction: we will assume that Inequality 9 above is violated, and prove that this contradicts the assumption that τ satisfies Condition 8.

Let t_o denote the earliest value of t at which Inequality 9 is violated, and consider some job $j \in J(\tau, k)$ such that j has received less service by time $t_o + e_{\mathsf{max}}(\tau)$ in $\mathsf{EDF}_{\mathsf{np}}(\tau)$ than by time t_o in $\mathsf{opt}(\tau)$:

$$W(\mathsf{EDF}_{\mathsf{np}}(\tau), \{j\}, t_o + e_{\mathsf{max}}(\tau)) < W(\mathsf{opt}(\tau), \{j\}, t_o) \ .$$

(Since the total amount of work done on all jobs in $J(\tau,k)$ over $[0, t_o + e_{\mathsf{max}}(\tau))$ is strictly less in $\mathsf{EDF}_{\mathsf{np}}(\tau)$ that the total amount of work done on all jobs in $J(\tau,k)$ over $[0,t_o)$ in $\mathsf{opt}(\tau)$, there must exist some such job $j \in J(\tau,k)$ such that j receives strictly less service.) Let a_j denote the arrival time of job j.

By our choice of t_o as the first instant at which Inequality 9 is violated, it must be the case (since $a_j < t_o$) that

$$W(\mathsf{EDF}_{\mathsf{np}}(\tau), J(\tau,k), a_j + e_{\mathsf{max}}(\tau)) \geq W(\mathsf{opt}(\tau), J(\tau,k), a_j).$$

Therefore, the amount of work done on jobs in $J(\tau,k)$ in $\mathsf{opt}(\tau)$ over $[a_j, t_o)$ is strictly greater than the amount of work done on jobs in $J(\tau,k)$ in $\mathsf{EDF}_{\mathsf{np}}(\tau)$ over the interval $[a_j + e_{\mathsf{max}}(\tau), t_o + e_{\mathsf{max}}(\tau))$.

By time-instant $r_j + e_{\mathsf{max}}(\tau)$, all jobs that were executing at time-instant r_j in schedule $\mathsf{EDF}_{\mathsf{np}}(\tau)$ will have completed execution. Over the interval $[a_j + e_{\mathsf{max}}(\tau), t_o + e_{\mathsf{max}}(\tau))$, therefore, $\mathsf{EDF}_{\mathsf{np}}(\tau)$ executes j in preference to jobs not in $J(\tau,k)$ since it assigns job j greater priority than all such jobs. Hence at all instants within this interval $[a_j + e_{\mathsf{max}}(\tau), t_o + e_{\mathsf{max}}(\tau))$ during which $\mathsf{EDF}_{\mathsf{np}}(\tau)$ is not executing j, it is executing jobs in $J(\tau,k)$ upon all processors. Let x denote the cumulative length of time over the interval $[a_j + e_{\mathsf{max}}(\tau), t_o + e_{\mathsf{max}}(\tau))$ during which $\mathsf{EDF}_{\mathsf{np}}(\tau)$ is executing jobs in $J(\tau,k)$ upon all m processors; let $y \stackrel{\text{def}}{=} (t_o - a_j) - x)$ denote the length of time over this interval during which $\mathsf{EDF}_{\mathsf{np}}(\tau)$ either idles some processor, or executes some job not in $J(\tau,k)$.

We make the following two observations.

- Since $\mathsf{EDF}_{\mathsf{np}}$ is a work-conserving scheduling algorithm, job j, which has not completed by instant $t_o + e_{\mathsf{max}}(\tau)$ in the schedule $\mathsf{EDF}_{\mathsf{np}}(\tau)$, must have executed for at least y time units by time $t_o + e_{\mathsf{max}}(\tau)$ in $\mathsf{EDF}_{\mathsf{np}}(\tau)$; while it executed for an amount no larger than $(x+y)V_{\mathsf{max}}(\tau)$

over $[a_j, t_o)$ in schedule $\mathsf{opt}(\tau)$; therefore,

$$(x + y) \times V_{\mathsf{max}}(\tau) > y \tag{10}$$

- The amount of work upon jobs in $J(\tau, k)$ done in $\mathsf{EDF}_{\mathsf{np}}(\tau)$ over $[a_j + e_{\mathsf{max}}(\tau), t_o + e_{\mathsf{max}}(\tau))$ is at least $(mx + y)$, while the amount of work done upon these jobs in $\mathsf{opt}(\tau)$ over the interval $[a_j, t_o)$ is at most $(x + y)V_{\mathsf{sum}}(\tau)$. Therefore, it is the case that

$$(x + y)V_{\mathsf{sum}}(\tau) > mx + y \tag{11}$$

Adding $(m - 1)$ times Inequality 10 to Inequality 11, we get

$$\begin{aligned}
&(x + y)(m - 1) \times V_{\mathsf{max}}(\tau) + (x + y) \times V_{\mathsf{sum}}(\tau) > (m - 1)y + mx + y \\
\equiv\ & (x + y)((m - 1)V_{\mathsf{max}}(\tau) + V_{\mathsf{sum}}(\tau)) > m(x + y) \\
\equiv\ & (m - 1)V_{\mathsf{max}}(\tau) + V_{\mathsf{sum}}(\tau) > m \\
\equiv\ & V_{\mathsf{sum}}(\tau) > m - (m - 1)V_{\mathsf{max}}(\tau)
\end{aligned}$$

which contradicts the assumption that Condition 8 holds. ■

By Lemma 1 above, it follows that all jobs in $J(\tau, k)$ complete at least $e_{\mathsf{max}}(\tau)$ time units before the latest deadline of any job in $J(\tau, k)$. Hence by Lemma 2 above, all these jobs will complete by the latest deadline of any job in $J(\tau, k)$; from this observation, it follows that the job with the k'th-smallest deadline will meet its deadline provided all earlier-deadline jobs have completed by *their* deadlines. A proof by induction suggests itself; this is formalized in the proof of Theorem 1 below.

Theorem 1 Any task system τ satisfying Condition 8 above:

$$V_{\mathsf{sum}}(\tau) \leq m - (m-1) \times V_{\mathsf{max}}(\tau)$$

is successfully scheduled by $\mathsf{EDF}_{\mathsf{np}}$ to meet all deadlines.

Proof: The proof is by induction on the jobs, ordered by deadline (with tied deadlines resolved according to the same rule used by Algorithm $\mathsf{EDF}_{\mathsf{np}}$).

The base case concerns the empty set of jobs — $\mathsf{EDF}_{\mathsf{np}}(\tau)$ trivially meets all deadlines in the empty set of jobs.

For the inductive step, for each integer $k \geq 1$ we assume that the jobs with the first $k-1$ deadlines have completed by their deadlines in $\mathsf{EDF}_{\mathsf{np}}(\tau)$, and prove that the k'th-earliest deadline job generated by τ completes by its deadline in $\mathsf{EDF}_{\mathsf{np}}(\tau)$.

Let d_k denote the deadline of the k'th earliest deadline job generated by τ.

- By Lemma 1, $\mathsf{opt}(\tau)$ completes all jobs in $J(\tau, k)$ by time-instant $d_k - e_{\mathsf{max}}(\tau)$.
- By Lemma 2, $\mathsf{EDF}_{\mathsf{np}}(\tau)$ performs at least as much work on $J(\tau, k)$ by time instant d_k as $\mathsf{opt}(\tau)$ does by time-instant $d_k - e_{\mathsf{max}}(\tau)$. But since $\mathsf{opt}(\tau)$ *completes* all these jobs by time-instant $d_k - e_{\mathsf{max}}(\tau)$, it must be the case that $\mathsf{EDF}_{\mathsf{np}}(\tau)$ also completes all these jobs by time-instant d_k.

■

For families of task systems for which it can be *a priori* guaranteed that the largest execution requirement of any task is less than ρ times the smallest period of any task for some constant $\rho < 1$, Theorem 1 yields a simple utilization-based $\mathsf{EDF}_{\mathsf{np}}$-feasibility condition:

Corollary 1 Let τ denote any task system, and let ρ be a constant such that

$$e_{\mathsf{max}}(\tau) \leq \rho \times p_{\mathsf{min}}(\tau) \ . \tag{12}$$

A sufficient condition for τ to be $\mathsf{EDF}_{\mathsf{np}}$-feasible is that

$$U_{\mathsf{sum}}(\tau) \leq m(1-\rho) - (m-1) \times U_{\mathsf{max}}(\tau) \ . \tag{13}$$

Proof: Observe that

$$\begin{aligned} V(\tau_i, \tau) &\stackrel{\text{def}}{=} \frac{e(\tau_i)}{p(\tau_i) - e_{\mathsf{max}}(\tau)} \\ &\leq \frac{e(\tau_i)}{p(\tau_i) - \rho \times p_{\mathsf{min}}(\tau)} \\ &\leq \frac{e(\tau_i)}{p(\tau_i) - \rho \times p_i} \\ &= U(\tau_i) \left(\frac{1}{1-\rho} \right) \end{aligned}$$

Hence, by the definitions of $V_{\mathsf{sum}}(\tau)$ and $V_{\mathsf{max}}(\tau)$,

$$V_{\mathsf{sum}}(\tau) \leq U_{\mathsf{sum}}(\tau) \left(\frac{1}{1-\rho} \right) \tag{14}$$

and

$$V_{\mathsf{max}}(\tau) \leq U_{\mathsf{max}}(\tau) \left(\frac{1}{1-\rho} \right) \ . \tag{15}$$

By Theorem 1,

$$V_{\mathsf{sum}}(\tau) \leq m - (m-1) \times V_{\mathsf{max}}(\tau)$$

is sufficient for ensuring that τ is $\mathsf{EDF}_{\mathsf{np}}$-feasible upon m processors. But

$$
\begin{aligned}
& V_{\mathsf{sum}}(\tau) \leq m - (m-1) \times V_{\mathsf{max}}(\tau) \\
\Leftarrow \quad & U_{\mathsf{sum}}(\tau) \left(\frac{1}{1-\rho}\right) \leq m - (m-1) \times U_{\mathsf{max}}(\tau) \left(\frac{1}{1-\rho}\right) \quad \text{(By(14)and(15)above)} \\
\equiv \quad & U_{\mathsf{sum}}(\tau) \leq m(1-\rho) + (m-1) \times U_{\mathsf{max}}(\tau)
\end{aligned}
$$

which is as stated in the corollary. ■

Inequality 13 is syntactically very similar to the utilization-based feasibility condition for preemptive earliest-deadline based scheduling (see, e.g., [3]:

$$U_{\mathsf{sum}}(\tau) \leq m - (m-1) \times U_{\mathsf{max}}(\tau) \ . \tag{16}$$

The added penalty we pay in non-preemptive scheduling is that the "capacity" of each processor is reduced by a factor ρ; in a sense, this penalty is the overhead associated with non-preemptivity. For task systems in which ρ is relatively small, the advantages of non-preemptive scheduling (less overhead in terms of context-switch cost, saving state, etc.) may well be worth this additional loss of capacity. It is also noteworthy that as faster processors become available it is not unreasonable to expect ρ to become smaller for a certain class of applications – those in which deadline constraints are drawn from the external environment, and are hence less likely to be influenced by increase in processor speed. Hence it may well be that non-preemptive scheduling will become more popular in the future as processor speeds increase.

4 Conclusions and Discussion

The goal of the research reported here has been to determine feasibility conditions for non-preemptive multiprocessor scheduling upon mltiprocessors. Previous research [5, 4, 2] has revealed that exact feasibility-analysis of non-preemptive periodic task systems is a highly intractable problem even upon a single processor; hence our effort has been in determining sufficient, rather than exact, feasibility conditions. To this end, we have analyzed the non-preemptive EDF-based scheduling of periodic task systems upon identical multiprocessors, and have eeveloped a sufficient test, which run in time linear in the representation of the task system, for determining whether a periodic task system that is scheduled using the non-preemptive earliest deadline first scheduling algorithm (Algorithm $\mathsf{EDF}_{\mathsf{np}}$) will meet all deadlines. Below, we make explicit some additional features of our $\mathsf{EDF}_{\mathsf{np}}$-feasibility analysis algorithm.

Extensions to other task models. While we restricted our analysis in this paper to systems comprised of concrete [5] periodic task systems only, in which each periodic task is characterized by three parameters initial arrival time, exeuction requirement, and period, it is straightforward to extend our result to apply to

- systems comprised of **sporadic** tasks, in which the period parameter signifies the minimum, rather than the exact, separation between arrivals of successive jobs of the task, and
- **non-concrete** [5] periodic/ sporadic task systems, in which the initial arrival time of each task is left unspecified and it is incumbent upon the feasibility-analysis algorithm to ensure that the system meets all deadlines for all combinations of arrival times.

Robustness. A scheduling framework is said to be **robust** with respect to workload if it is the case that decreasing the execution times of individual jobs in a system which is known to meet all

deadlines does not result in some deadlines being missed. Such robustness with respect to workload is extremely desirable, since this property allows the system designer to consider just the worst-case behavior of a real-time system, instead of infinitely many possible behaviors, during analysis to determine whether the system will indeed meet all deadlines. Unfortunately, the non-preemptive scheduling of periodic tasks is *not* robust if we are restricted to using work-conserving scheduling algorithms; for example, a periodic task system that is known to be $\mathsf{EDF}_{\mathsf{np}}$-feasible may miss deadlines if the actual execution requirements of some jobs turn out to be strictly less than the worst-case execution requirement (as specified in the generating task's execution requirement parameter). Hence, exact feasibility-analysis tests for periodic task systems under the work-conserving restriction are necessarily non-robust; while of interest from a theoretical perspective, such tests are of limited practical significance since system designers tend to to characterize their workload by obtaining upper-bounds upon, rather than exact values of, job execution requirements. Of more practical significance would be feasibility tests which are robust in the sense that if a periodic task system is guaranteed by this test to meet all deadlines, then it is guaranteed to continue to meet all deadlines even if some of the execution requirements are decreased. It is not difficult to see that the sufficient feasibility tests presented in this paper are indeed robust: if a periodic task system τ passes these tests, then it is guaranteed that $\mathsf{EDF}_{\mathsf{np}}(\tau)$ will meet all deadlines provided all jobs have execution requirements no more than the worst-case execution requirements of the periodic tasks that generate them.

Liquid tasks. Abdelzaher et al. [1] have defined a real-time system of **liquid tasks** to be a real-time system in which the smallest deadline of any job in the system is orders of magnitude greater than the largest execution requirement of any job, and have identified a large class of applications which may be modelled as liquid tasks. Corollary 1 implies that our feasibility test is

particularly suited for use upon systems that can be modelled as liquid task systems: since the ratio $e_{\mathsf{max}}(\tau)/p_{\mathsf{min}}(\tau)$ approaches zero for such systems, the achievable utilization for non-preemptive EDF-scheduling approaches that of preemptive EDF-scheduling, which in turn approaches the platform capacity (since $U_{\mathsf{max}}(\tau)$ also $\rightarrow 0$). This observation strongly argues in favour of scheduling periodic liquid task systems non-preemptively.

References

[1] Abdelzaher, T., Andersson, B., Jonsson, J., Sharma, V., and Nguyen, M. The aperiodic multiprocessor utilization bound for liquid tasks. Submitted for publication, 2002.

[2] Cai, Y., and Kong, M. C. Nonpreemptive scheduling of periodic tasks in uni- and multiprocessor systems. *Algorithmica 15*, 6 (June 1996), 572–599.

[3] Goossens, J., Funk, S., and Baruah, S. Priority-driven scheduling of periodic task systems on uniform multiprocessors. *Real Time Systems*. To appear.

[4] Howell, R., and Venkatrao, M. On non-preemptive scheduling of recurring tasks using inserted idle times. *Information and Computation 117* (1995), 50–62.

[5] Jeffay, K., Stanat, D., and Martel, C. On non-preemptive scheduling of periodic and sporadic tasks. In *Proceedings of the 12th Real-Time Systems Symposium* (San Antonio, Texas, December 1991), IEEE Computer Society Press, pp. 129–139.

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

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

[8] 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.