Global mixed-criticality scheduling on multiprocessors

Haohan Li Sanjoy Baruah
The University of North Carolina at Chapel Hill

Abstract— **The scheduling of mixed-criticality implicit-deadline sporadic task systems on identical multiprocessor platforms is considered, when inter-processor migration is permitted. A scheduling algorithm is derived and proved correct, and its properties investigated. Theoretical analysis (in the form of both a speedup factor and sufficient schedulability conditions) as well as extensive simulation experiments serve to demonstrate its effectiveness.**

Keywords. Mixed-criticality system; multiprocessors; global scheduling; sporadic task system; speedup factor.

I. Introduction

There is an increasing trend in embedded systems towards integrating multiple functionalities on a common platform, as is evidenced by industry-wide initiatives such as IMA (Integrated Modular Avionics) for aerospace, and AUTOSAR (AUTomotive Open System ARchitecture) for the automotive industry. Such integrated platforms may co-host applications of different degrees of importance or criticalities, with some of the more safety-critical applications subject to mandatory certification by statutory *Certification Authorities* (CAs.) Real-time scheduling theory has only recently begun addressing the resulting challenge of implementing safety-critical functionalities that are subject to certification, and less critical functionalities that do not need to be certified, on a common platform (see, e.g, [12], [7], [5], [4]).

In order to certify a system as being correct, the certification authority (CA) mandates certain assumptions about the worst-case behavior of the system during run-time; these assumptions are typically far more conservative than the assumptions that the system designer would use during the process of designing, implementing, and testing the system if subsequent certification were not required. (For instance, the worst-case execution time (WCET) estimate used by the CA to characterize a complex piece of code is likely to be more pessimistic (i.e., larger) than the WCET estimate used by the system designer.) However, while the CA is only concerned with the correctness of the safety-critical part of the system the system designer is responsible for ensuring that the entire system is correct, including the non-critical parts. The scheduling problem then becomes one of coming up with a single scheduling strategy that meets two separate goals: (i) certification of the high criticality jobs under more pessimistic assumptions and (ii) feasibility of *all* the jobs (including the low criticality ones) under the designer's, less pessimistic, assumptions.

This research. To our knowledge, most prior algorithmic research on mixed-criticality scheduling for certifiability has focused on uniprocessor systems. However, safety-critical (and other) embedded systems are increasingly coming to be implemented on *multicore* CPU's, which are typically scheduled using multiprocessor scheduling algorithms rather than uniprocessor ones. Furthermore, as these multicore CPU's become more complex and sophisticated, their behavior becomes less uniform and predictable and shows greater variation; as a consequence, the pessimism that a CA's more conservative WCET-analysis tool expresses vis a vis a system-designer's tool is only going to increase and become more pronounced. Motivated by these and other reasons, we seek to study mixed-criticality scheduling on multiprocessors in this paper. Specifically, we consider the global preemptive scheduling of mixed-criticality implicit-deadline sporadic task systems upon identical multiprocessor platforms – these terms will be defined, and the precise scope of our work will be delineated, in Section II. Our approach extends a uniprocessor mixed-criticality scheduling algorithm called EDF-VD [6] to multiprocessors, by applying a previously-proposed multiprocessor global scheduling algorithm called fpEDF [2] that was designed for non mixed-criticality systems.

Organization. The remainder of this paper is organized as follows. In Section II we formally define the workload and platform models assumed in this work. In Section III we briefly describe some prior work that we will be using — specifically, the multiprocessor global scheduling algorithm fpEDF [2], that was designed for non mixed-criticality systems. In Section IV we provide an overview of our algorithm; a detailed description follows in Section V. We provide a worst-case characterization of the algorithm's performance in Section VI. In Section VII we describe some pragmatic improvements that can be made to the algorithm described in Sections IV-V; although these improvements do not change the worst-case characterization of our algorithm, they do significantly improve its performance on typical task systems. We illustrate this fact by presenting the results of some simulation experiments in Section VIII.

II. Model and definitions

As stated in Section I above, we are studying the global preemptive scheduling of mixed-criticality implicit-deadline sporadic task systems on a multiprocessor platform consisting of multiple identical processors. By *global* scheduling, we mean that any job may execute on any processor, and that

a preempted job may later resume execution on any processor. In common with much prior research on multiprocessor scheduling, we assume that there is no cost associated with preemptions and inter-processor migration.

To our knowledge, this is one of the first papers to consider mixed-criticality scheduling on multiprocessors. The only prior publications that we are aware of on this topic are

1) [6, Section 4], which considers (1) the global multiprocessor scheduling of a finite collection of independent *jobs*; and (2) the *partitioned* multiprocessor scheduling of implicit-deadline sporadic task systems.
2) [11], which considers a very simply workload model (all jobs have the same release time and deadline) and deals primarily with implementation issues.

We now formally define the mixed-criticality (henceforth often referred to as **MC**) workload model that is used in this paper, and explain terms and concepts used throughout the remainder of this document. As with traditional (i.e., non MC) real-time systems, we will model a MC real-time system τ as consisting of a finite specified collection of MC sporadic tasks, each of which will generate a potentially infinite sequence of MC jobs.

MC jobs. Each job is characterized by a 5-tuple of parameters: $J_i = (a_i, d_i, \chi_i, c_i(\text{LO}), c_i(\text{HI}))$, where

- $a_i \in R^+$ is the release time.
- $d_i \in R^+$ is the deadline. We assume that $d_i \geq a_i$.
- $\chi_i \in \{\text{LO}, \text{HI}\}$ denotes the criticality of the job. A HI-criticality job (a J_i with $\chi_i = \text{HI}$) is one that is subject to certification, whereas a LO-criticality job (a J_i with $\chi_i = \text{LO}$) is one that does not need to be certified.
- $c_i(\text{LO})$ specifies the worst case execution time (WCET) estimate of J_i that is used by the system designer (i.e., the WCET estimate at the LO criticality level).
- $c_i(\text{HI})$ specifies the worst case execution time (WCET) estimate of J_i that is used by the certification authorities (i.e., the WCET estimate at the HI criticality level). We assume that
  - $c_i(\text{HI}) \geq c_i(\text{LO})$ (i.e., the WCET estimate used by the system designer is never more pessimistic than the one used by the CA), and
  - $c_i(\text{HI}) = c_i(\text{LO})$ if $\chi_i = \text{LO}$ (i.e., a LO-criticality job is aborted if it executes for more than its LO-criticality WCET estimate[1]).

The MC job model has the following semantics. Job J_i is released at time a_i, has a deadline at d_i, and needs to execute for some amount of time γ_i. However, *the value of γ_i is not known beforehand, but only becomes revealed by actually executing the job until it* signals *that it has completed execution*. If J_i signals completion without exceeding $c_i(\text{LO})$ units of execution, we say that it has exhibited LO-criticality behavior; if it signals completion after executing for more than $c_i(\text{LO})$ but no more than $c_i(\text{HI})$ units of execution, we say that it has exhibited HI-criticality behavior. If it does not signal completion upon having executed for $c_i(\text{HI})$ units, we say that its behavior is erroneous.

[1]We assume that the run-time system provides support for ensuring that jobs do not execute for more than a specified amount; see, e.g., [4] for a discussion of this issue.

	χ_k	$C_k(\text{LO})$	$C_k(\text{HI})$	T_k
τ_1	LO	2	2	6
τ_2	HI	1	2	10
τ_2	HI	2	10	20

TABLE I
AN EXAMPLE MIXED-CRITICALITY IMPLICIT-DEADLINE SPORADIC TASK SYSTEM.

MC implicit-deadline sporadic tasks. Each implicit-deadline sporadic task in the MC model is characterized by a 4-tuple of parameters: $\tau_k = (\chi_k, C_k(\text{LO}), C_k(\text{HI}), T_k)$, with the following interpretation. Task τ_k generates a potentially infinite sequence of jobs, with successive jobs being released at least T_k time units apart. Each such job has a deadline that is T_k time units after its release. The criticality of each such job is χ_k, and it has LO-criticality and HI-criticality WCET's of $C_k(\text{LO})$ and $C_k(\text{HI})$ respectively.

A MC ***sporadic task system*** is specified as a finite collection of such sporadic tasks. As with traditional (non-MC) systems, such a MC sporadic task system can potentially generate infinitely many different MC instances (collections of jobs), each instance being obtained by taking the union of one sequence of jobs generated by each sporadic task.

Utilizations. The *utilization* of a (regular, i.e., non-MC) implicit-deadline sporadic task denotes the ratio of its WCET to its period; the utilization of a task system denotes the sum of the utilizations of all the tasks in the system. We now define analogous concepts for mixed-criticality sporadic task systems. That is, we let $U_k(\text{LO})$ and $U_k(\text{HI})$ denote the LO-criticality and HI-criticality utilizations of task τ_k:

$$U_k(\text{LO}) := C_k(\text{LO})/T_k \text{ and } U_k(\text{HI}) := C_k(\text{HI})/T_k$$

Let $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ denote a MC implicit-deadline sporadic task system. For each of x and y in $\{\text{LO}, \text{HI}\}$, we define a utilization parameter as follows:

$$U_x^y(\tau) = \sum_{\tau_i \in \tau \wedge \chi_i = x} U_i(y) \quad (1)$$

Thus for example, $U_{\text{HI}}^{\text{LO}}(\tau)$ denotes the sum of the utilizations of the HI-criticality tasks in τ, under the assumption that each job of each task executes for no more than its LO-criticality WCET.

Example 1: Consider the task system depicted in Table I. For this task system,

$$\begin{aligned} U_{\text{LO}}^{\text{LO}}(\tau) &= 2/6 = 0.33 \\ U_{\text{LO}}^{\text{HI}}(\tau) &= 2/6 = 0.33 \\ U_{\text{HI}}^{\text{LO}}(\tau) &= 1/10 + 2/20 = 0.2 \\ U_{\text{HI}}^{\text{HI}}(\tau) &= 2/10 + 10/20 = 0.7 \end{aligned}$$

■

Scheduling MC sporadic task systems. A particular implicit-deadline sporadic task system may generate different

instances of jobs during different runs. Furthermore, during any given run each job comprising the instance may exhibit LO-criticality, HI-criticality, or erroneous behavior. We define an algorithm for scheduling implicit-deadline sporadic task system τ to be *correct* if it is able to schedule every instance generated by τ such that

- If all jobs exhibit LO-criticality behavior, then all jobs receive enough execution between their release time and deadline to be able to signal completion; and
- If *any* job exhibits HI-criticality behavior, then all HI-criticality jobs receive enough execution between their release time and deadline to be able to signal completion.

Note that if any job exhibits HI-criticality behavior, we do not require any LO-criticality jobs (including those that may have arrived before this happened) to complete by their deadlines. This is an implication of the requirements of certification: informally speaking, the system designer fully expects that all jobs will exhibit LO-criticality behavior, and hence is only concerned that they behave as desired under these circumstances. The CA, on the other hand, allows for the possibility that some jobs may exhibit HI-criticality behavior, and requires that all HI-criticality jobs nevertheless meet their deadlines.

III. Algorithm fpEDF [2]

Algorithm fpEDF [2] is a global EDF-based algorithm for scheduling systems of non mixed-criticality implicit-deadline sporadic tasks upon identical multiprocessor platforms. The mixed-criticality scheduling algorithm that we present in the following sections is based on Algorithm fpEDF; therefore, we describe it here and provide a brief overview of some results from [2].

Suppose that "regular" (i.e., non-MC) implicit-deadline sporadic task system τ is to be scheduled on m unit-speed processors. During run-time all jobs of tasks in τ that have utilization greater than one-half are assigned highest priority, and the remaining tasks' jobs are assigned priorities according to their deadlines (as in "regular" EDF).

Let $U_{\mathsf{sum}}(\tau)$ (respectively, $U_{\mathsf{max}}(\tau)$) denote the sum of the utilizations (resp., the largest utilization) of the tasks in τ. The *schedulable utilization* of a multiprocessor scheduling algorithm on m speed-s processors is defined to be the largest number such that any task system τ with $U_{\mathsf{sum}}(\tau)$ no larger than this number and $U_{\mathsf{max}}(\tau)$ no larger than s, is correctly scheduled by the algorithm on the m processors. The following result characterizes the performance guarantees made by Algorithm fpEDF:

Theorem 1 (Theorem 4 in [2]): Algorithm fpEDF has a schedulable utilization of $(m + 1)/2$ upon m unit-speed processors. ■

Since preemptive uniprocessor EDF is known [10] to have a schedulable utilization equal to the speed of the processor on which it is implemented, the contrapositive of Theorem 1 yields the following corollary.

Corollary 1: If a task system cannot be scheduled by Algorithm fpEDF on m unit-speed processors, then it cannot be scheduled by preemptive uniprocessor EDF on a processor of speed $(m + 1)/2$. ■

IV. An overview of our scheduling algorithm

We now have the foundations in place to describe our proposed algorithm for the global scheduling of implicit-deadline sporadic mixed-criticality task systems on preemptive identical multiprocessor platforms. In this section, we provide a high-level overview of the algorithm, and attempt to communicate the intuition behind our algorithm design by means of a very simple example. We will fill in the details with a more comprehensive description in Section V, and prove some important properties in Section VI. We reiterate that our algorithm is a generalization to multiprocessor platforms, of the preemptive uniprocessor algorithm EDF-FD first proposed in [6].

Let $\tau = \{\tau_1, \ldots, \tau_n\}$ denote the MC implicit-deadline sporadic task system that is to be scheduled on m unit-speed preemptive processors. Our approach to scheduling τ can be thought of as a three-phased one:

1) There is an initial pre-processing phase that occurs prior to run-time.
2) During run-time, jobs are initially dispatched in the expectation that the behavior of the system is going to be a LO-criticality one: no job will execute for more than its LO-criticality WCET.
3) If some job does execute beyond its LO-criticality WCET without signaling that it has completed execution, the dispatching algorithm is modified accordingly and the algorithm enters its optional third phase.

We now discuss each of the three phases.

During the *pre-processing phase*, a schedulability test is performed to determine whether τ can be successfully scheduled by our algorithm or not. If τ is deemed schedulable, then an additional parameter, which we call a *modified period* denoted $\hat{T}_i$, is computed for each HI-criticality task $\tau_i \in \tau$. We will see that it is always the case that $\hat{T}_i \leq T_i$.

Initial run-time scheduling is done according to the Algorithm fpEDF described in Section III above. Since fpEDF is defined for regular, rather than mixed-criticality, task systems, we must map the mixed-criticality tasks in τ to regular tasks. This is done as follows: each LO-criticality task $\tau_k = (\chi_k, C_k(\text{LO}), C_k(\text{HI}), T_k)$ in τ is mapped to a regular implicit-deadline task $(C_k(\text{LO}), T_k)$, while each HI-criticality task $\tau_k = (\chi_k, C_k(\text{LO}), C_k(\text{HI}), T_k)$ in τ is mapped to a regular implicit-deadline task $(C_k(\text{LO}), \hat{T}_k)$, where the $\hat{T}_k$'s are the modified periods computed during the pre-processing phase. It follows from the *sustainability* property [3], [1] of Algorithm fpEDF that if Algorithm fpEDF is able to schedule this regular implicit-deadline sporadic task system then it is able to schedule all LO-criticality behaviors of the MC implicit-deadline task system τ.

If some job does execute beyond its LO-criticality WCET without signaling that it has completed execution, we enter the *third phase* of the algorithm, and the following changes occur.

1) All currently-active LO-criticality jobs are immediately discarded; henceforth, no LO-criticality job will receive any execution.
2) Subsequent run-time scheduling of the HI-criticality tasks (including their jobs that are currently active) are done according to Algorithm fpEDF. In order to do so, we must once again map these HI-criticality tasks to regular implicit-deadline tasks. This is done as follows: each HI-criticality MC task $\tau_k = (\chi_k, C_k(\text{LO}), C_k(\text{HI}), T_k)$ in τ is mapped to a regular implicit-deadline task $(C_k(\text{HI}), T_k - \hat{T}_k)$.

We now demonstrate these three phases via a simple example: scheduling a system of three implicit-deadline mixed-criticality tasks on a single preemptive processor. [2]

Example 2: Suppose we wish to schedule the mixed-criticality implicit-deadline sporadic task system depicted in Table I, on a single unit-speed processor. We will see later that the pre-processing phase determines that this task system is schedulable by our algorithm on a single processor ($m = 1$), and computes the following $\hat{T}_k$ parameters for the HI-criticality tasks τ_2 and τ_3:

$$\hat{T}_2 \leftarrow 3; \quad \hat{T}_3 \leftarrow 6 \ .$$

During run time, jobs are initially scheduled by handing off the regular task system $\{(2,6),(1,3),(2,6)\}$ to Algorithm fpEDF. Observe that $U_{\mathsf{sum}} = 2/6 + 1/3 + 2/6 = 1$ for this system; hence according to the optimality of preemptive uniprocessor EDF [10], it is successfully scheduled on a single processor by Algorithm fpEDF.

Now suppose some job executes for more than its LO-criticality WCET. All currently-active jobs of the LO-criticality task τ_1 are immediately discarded, and no future jobs of this task are admitted. Run-time dispatching is done by handing off the regular task system $\{(2, 10-3=7), (10, 20-6=14)\}$ to Algorithm fpEDF. Since $U_{\mathsf{max}} = 10/14 \approx 0.72$ and $U_{\mathsf{sum}} = 2/7 + 10/14 = 1$ for this system it is also successfully scheduled on a single processor by Algorithm fpEDF. In particular, any currently active job of τ_2 (τ_3 respectively) is immediately scheduled as a job with WCET $C_2(\text{HI}) = 2$ ($C_3(\text{HI}) = 10$, resp.) and a deadline that is $T_2 - \hat{T}_2 = 7$ ($T_3 - \hat{T}_3 = 14$, resp.) time-units in the future. ■

V. A DETAILED DESCRIPTION

We now provide a detailed description of our algorithm, specifying what happens during the pre-processing phase –how a system is deemed schedulable or not and how the modified period parameters (the $\hat{T}_k$'s) are computed for schedulable systems– and precisely how run-time job-dispatching decisions are made.

A. The pre-processing step

Consider some execution of the mixed-criticality implicit-deadline task system τ that exhibits HI-criticality behavior, i.e., some job executes beyond its LO-criticality WCET without signaling that it has completed execution. Let t^* denote the first time-instant at which some job executes for more than its LO-criticality WCET without signaling that it has completed — at t^*, therefore, the run-time system gets to know that the current behavior of the system is a HI-criticality one. The idea behind our algorithm is to ensure that there is sufficient computing capacity available between this time-instant t^* and the deadline of each currently-active HI-criticality job, to be able to execute all these jobs for up to their HI-criticality WCET's by their respective deadlines. This is ensured by the manner in which the modified periods (the $\hat{T}_k$ parameters) are computed. We will compute modified period values to ensure that the following two properties P1-P2 are satisfied:

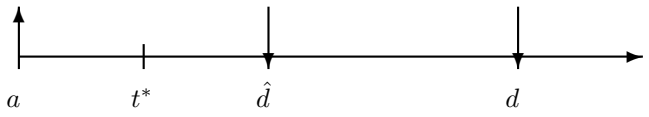


Fig. 1. A HI-criticality job arrives at time a, with deadline at d. It is scheduled by Algorithm fpEDF using the modified deadline $\hat{d}$, which is $\leq d$. If no job executes for more than its LO-criticality WCET, then this job can complete by $\hat{d}$. If only the HI-criticality jobs execute and each executes for up to its HI-criticality WCET, then this job, if a HI-criticality one, can meet its deadline by *only* executing over $[\hat{d}, d)$.

P1. All jobs of all tasks will meet their modified deadlines in any LO-criticality behavior of the system (i.e., if no job executes beyond its LO-criticality WCET). That is, the collection of "regular" (non-MC) tasks

$$\Big(\bigcup_{\chi_i = \text{LO}} \Big\{ (C_i(\text{LO}), T_i) \Big\} \Big) \bigcup \Big(\bigcup_{\chi_i = \text{HI}} \Big\{ (C_i(\text{LO}), \hat{T}_i) \Big\} \Big) \quad (2)$$

is scheduled by Algorithm fpEDF to always meet all deadlines on the available m unit-speed processors.

P2. If each HI-criticality job executes for no more than its HI-criticality WCET and each LO-criticality job does not execute at all then each HI-criticality job can meet its (original) deadline *by beginning execution at or after its modified deadline.* This is ensured by ensuring that the collection of "regular" (non-MC) tasks

$$\bigcup_{\chi_i = \text{HI}} \Big\{ (C_i(\text{HI}), T_i - \hat{T}_i) \Big\} \quad (3)$$

can be scheduled by Algorithm fpEDF to always meet all deadlines on the available m unit-speed processors.

In Section IV, we had stated that run-time scheduling during both the second and the (optional) third phase are done according to Algorithm fpEDF. We observe that in the regular implicit-deadline task systems being scheduled by Algorithm fpEDF during the second and third phases (i.e., prior to, and after, time-instant t^*), the periods of these regular tasks are smaller than or equal to the periods of the MC tasks that are mapped onto them. Hence it follows from the sustainability property of Algorithm fpEDF that all deadlines are met prior to time-instant t^*, and all deadlines are met

[2] Although this is not a multiprocessor example, it serves to illustrate the steps taken by the algorithm in a relatively simple manner.

at "steady state" well after t^* — i.e., once enough time has elapsed beyond t^* that only jobs of HI-criticality tasks that arrived well after t^* are active in the system. It remains to show that jobs that are active at time-instant t^* (see Figure 1), as well as jobs that have arrived soon after t^*, are correctly scheduled as well.

To see why this must be so, we note that each HI-criticality job that had its modified deadline $< t^*$ must have already signaled completion upon executing for at most its LO-criticality WCET, since otherwise the HI-criticality nature of the behavior would have been signaled prior to t^* when such a job failed to signal completion despite having executed for its LO-criticality WCET. Therefore the modified deadline of each HI-criticality job that is *active* —arrived but not yet signaled completion— at time-instant t^* must be $\geq t^*$. By our choice of modified deadlines and the sustainability property [3], [1] of Algorithm fpEDF, Property P2 ensures that all such HI-criticality jobs meet their original deadlines. As for the jobs of each HI-criticality task τ_i that arrive after t^*, they, too are scheduled with a scheduling deadline that is $(T_i - \hat{T}_i)$ after their release times; since they can meet these deadlines under Algorithm fpEDF, it follows that they meet their actual deadlines, which occur T_i time-units after their release times, as well.

That, then, is the gist of the idea behind the preprocessing phase. It remains to fill in the details, in particular by explaining how the modified deadlines are computed such that the two properties P1 and P2 are satisfied. The pre-processing phase is described in pseudo-code form in Figure 2. We provide an explanation of this pseudo-code below.

Step 1 checks to see whether Algorithm fpEDF can schedule the system if each LO-criticality job executes for up to its LO-criticality WCET, and each HI-criticality job executes for up to its HI-criticality WCET. If so, then the system can be scheduled directly by Algorithm fpEDF; else, Steps 2-3 are executed.

In **Step 2**, a minimum "scaling factor" x is determined, such that if all the HI-criticality tasks have their periods scaled by this factor x then the regular implicit-deadline task system obtained by combining these tasks with the LO-criticality tasks would be successfully scheduled by Algorithm fpEDF. The derivation of the value of x is as follows. According to Theorem 1, Algorithm fpEDF can schedule any task system with total utilization $\leq (m+1)/2$ (recall that m denotes the number of unit-speed processors). Since scaling the period of each HI-criticality task by a factor x is equivalent to inflating its utilization by a factor $1/x$, for ensuring LO-criticality schedulability by fpEDF we therefore need

$$\begin{aligned} & U_{\text{LO}}^{\text{LO}}(\tau) + \frac{U_{\text{HI}}^{\text{LO}}(\tau)}{x} \leq \frac{m+1}{2} \\ \Leftrightarrow \quad & \frac{U_{\text{HI}}^{\text{LO}}(\tau)}{x} \leq \frac{m+1}{2} - U_{\text{LO}}^{\text{LO}}(\tau) \\ \Leftrightarrow \quad & x \geq U_{\text{HI}}^{\text{LO}}(\tau) / \Big(\frac{m+1}{2} - U_{\text{LO}}^{\text{LO}}(\tau) \Big) \end{aligned}$$

This accounts for the first term in the "max." The second term is to ensure that scaling down the period of any HI-criticality task by this factor x does not result in the task having its LO-criticality WCET exceed its scaled-down period (equivalently, the term $\frac{C_i(\text{LO})}{xT_i}$ becoming > 1 for some HI-criticality task τ_i).

Step 3 determines whether the HI-criticality tasks can be scheduled to meet all deadlines by Algorithm fpEDF once the behavior of the system transits to HI-criticality (i.e., after the time-instant t^* at which some job is identified to have executed for more than its LO-criticality WCET). If so, the modified deadline parameters – the $\hat{T}_i$'s – are computed.

B. Run-time dispatching

During the execution of the system, jobs are selected for execution according to the following rules:

1) There is a *criticality level indicator* Γ, initialized to LO.
2) While $(\Gamma \equiv \text{LO})$,
   a) Suppose a job of some task $\tau_i \in \tau$ arrives at time t,.
      - if $\chi_i \equiv$ LO, the job is assigned a deadline equal to $t + T_i$.
      - if $\chi_i \equiv$ HI, the job is assigned a deadline equal to $t + \hat{T}_i$.
   b) At each instant the waiting job with earliest deadline is selected for execution (ties broken arbitrarily).
   c) If the currently-executing job executes for more than its LO-criticality WCET without signaling completion, then the behavior of the system is no longer a LO-criticality behavior, and $\Gamma :=$ HI.
3) Once $(\Gamma \equiv \text{HI})$,
   a) The deadline of each HI-criticality job that is currently active is changed to its release time plus the unmodified relative deadline of the task that generated it. That is, if a job of τ_i that was released at some time t is active, its deadline, for scheduling purposes, is henceforth $t + T_i$.
   b) When a future job of τ_i arrives at some time t, it is assigned a deadline equal to $t + T_i$.
   c) LO-criticality jobs will *not* receive any further execution. Therefore at each instant the earliest-deadline waiting job generated by a HI-criticality task is selected for execution(ties broken arbitrarily).
4) An additional rule could specify the circumstances when Γ gets reset to LO. This could happen, for instance, if no HI-criticality jobs are active at some instant in time. (We will not discuss the process of resetting $\Gamma :=$ LO any further in this document, since this is not relevant to the certification process — LO-criticality certification assumes that the system *never* exhibits any HI-criticality behavior, while HI-criticality certification is not interested in the behavior of the LO-criticality tasks.)

VI. Properties

As stated in Section I, our algorithm is essentially a generalization to multiprocessors of the uniprocessor mixed-criticality scheduling algorithm EDF-VD (for ***E**arliest **D**eadline **F**irst*

Algorithm GLOBAL. Task system $\tau = \{\tau_1, \tau_2, \ldots \tau_n\}$ to be scheduled on m processors.

1) **If** the regular task system
$$\bigcup_i \{(C_i(\chi_i), T_i)\}$$
is deemed schedulable on the m processors by Algorithm fpEDF, **then** declare success and **return**.
2) $x \leftarrow \max\Big(U_{\text{HI}}^{\text{LO}}(\tau)/\big(\frac{m+1}{2} - U_{\text{LO}}^{\text{LO}}(\tau)\big), \max_{\chi_i=\text{HI}}\big\{U_i(\text{LO})\big\}\Big)$
3) **If** the regular task system
$$\bigcup_{\chi_i=\text{HI}} \big\{(C_i(\text{HI}), (1-x)T_i)\big\}$$
is deemed schedulable on the m processors by Algorithm fpEDF, **then**
$\hat{T}_i \leftarrow xT_i$ for each HI-criticality task τ_i
declare success and **return**.

else declare failure and **return**.

Fig. 2. The preprocessing phase.

*with **V**irtual **D**eadlines*) of [6]. We now provide a brief description of this algorithm.

Algorithm EDF-VD, like our algorithm, also computes a modified period $\hat{T}_i = xT_i$ for every HI-criticality task τ, by determining the smallest value for x such that the following two collections of regular (non-MC) implicit-deadline sporadic tasks

1) $\Big(\bigcup_{\chi_i=\text{LO}}\Big\{\big(C_i(\text{LO}), T_i\big)\Big\}\Big) \bigcup \Big(\bigcup_{\chi_i=\text{HI}}\Big\{\big(C_i(\text{LO}), \hat{T}_i\big)\Big\}\Big)$, and
2) $\bigcup_{\chi_i=\text{HI}}\Big\{\big(C_i(\text{HI}), T_i - \hat{T}_i\big)\Big\}$

are each (separately) EDF-schedulable on a unit-speed processor.[3] If such an x cannot be determined, then EDF-VD declares failure; else, it was shown in [6] that τ can be scheduled using the run-time dispatching algorithm that we have described in Section V-B above.

The following theorem concerning EDF-VD is proved in [6].

Theorem 2 (from [6]): Any task system τ satisfying
$$U_{\text{LO}}^{\text{LO}}(\tau) + \min\Big(U_{\text{HI}}^{\text{HI}}(\tau), \frac{U_{\text{HI}}^{\text{LO}}(\tau)}{1 - U_{\text{HI}}^{\text{HI}}(\tau)}\Big) \leq s \tag{4}$$
is successfully scheduled by Algorithm EDF-VD on a preemptive speed-s processor. ■

Using this theorem and Corollary 1, the following ***sufficient schedulability condition*** can be derived for our multiprocessor mixed-criticality scheduling algorithm:

Theorem 3: Any task system τ satisfying
$$U_{\text{LO}}^{\text{LO}}(\tau) + \min\Big(U_{\text{HI}}^{\text{HI}}(\tau), \frac{U_{\text{HI}}^{\text{LO}}(\tau)}{1 - U_{\text{HI}}^{\text{HI}}(\tau)}\Big) \leq \frac{m+1}{2} \tag{5}$$
is successfully scheduled by our algorithm on m preemptive unit-speed processors. ■

In [6], it was also shown that

Theorem 4 (from [6]): Any task system τ satisfying
$$\max\Big(U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{LO}}(\tau), U_{\text{HI}}^{\text{HI}}(\tau)\Big) \leq s \tag{6}$$
is successfully scheduled by Algorithm EDF-VD on a preemptive speed-$\big(\frac{\sqrt{5}+1}{2}s\big)$ processor. ■

Once again, it follows as a consequence of this theorem and Corollary 1 that

Theorem 5: Any task system τ satisfying
$$\max\Big(U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{LO}}(\tau), U_{\text{HI}}^{\text{HI}}(\tau)\Big) \leq m \tag{7}$$
is successfully scheduled by our algorithm on m preemptive speed-$\big(\sqrt{5}+1\big)$ processors. ■

We can use Theorem 5 above to obtain a ***processor speedup bound*** [9] for our multiprocessor scheduling algorithm:

Corollary 2: The processor speedup factor of our algorithm is no larger than $(\sqrt{5}+1)$. That is, any mixed-criticality task system that can be scheduled in a certifiably correct manner on m unit-speed processors by an optimal clairvoyant scheduling algorithm can be scheduled by our algorithm on m speed-$\big(\sqrt{5}+1\big)$ processors.

Proof: Suppose that τ can be scheduled in a certifiably correct manner on m unit-speed processors by an optimal clairvoyant scheduling algorithm. It is necessary that its LO-criticality utilization $(U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{LO}}(\tau))$ be $\leq m$, and that its HI-criticality utilization $(U_{\text{HI}}^{\text{HI}}(\tau))$ also be $\leq m$. The speedup result immediately follows, by Theorem 5 above. ■

VII. Pragmatic improvements

The algorithm in Figure 2 describes a technique for computing a scaling factor x; Theorem 3 details sufficient conditions under which this scaling factor is guaranteed to yield a certifiably correct scheduling strategy.

What happens, however, if the task system to be scheduled does *not* satisfy the conditions of Theorem 3? In this case, the algorithm in Figure 2 may fail to find a value for x that results in a certifiably correct scheduling strategy. However, such failure does not necessarily imply that an appropriate

[3] Observe that these conditions are exactly the ones that we have generalized in order to come up with the conditions P1-P2 (Equations 2 and 3) in Section V.

Algorithm GLOBAL-PRAGMATIC. Task system $\tau = \{\tau_1, \tau_2, \ldots \tau_n\}$ to be scheduled on m processors.

1) **If** the regular task system
$$\bigcup_i \{(C_i(\chi_i), T_i)\}$$
is deemed schedulable on the m processors by Algorithm fpEDF, **then** declare success and **return**.
2) **for** each τ_i with $\chi_i = \text{HI}$
   a) $x \leftarrow 2C_i(\text{LO})/T_i$
   b) **If** $x < \min_{\chi_i=\text{HI}}\left(\frac{C_i(\text{LO})}{T_i}\right)$ or $x \geq 1$ or $x \leq 0$ **then continue** // Consider the next task
   c) **If** both the regular task systems
$$\Big(\bigcup_{\chi_i=\text{LO}} \Big\{(C_i(\text{LO}), T_i)\Big\}\Big) \bigcup \Big(\bigcup_{\chi_i=\text{HI}} \Big\{(C_i(\text{LO}), xT_i)\Big\}\Big)$$
   and
$$\bigcup_{\chi_i=\text{HI}} \{(C_i(\chi_i), (1-x)T_i)\}$$
   are deemed schedulable on the m processors by Algorithm fpEDF, **then**
      $\hat{T}_i \leftarrow xT_i$ for each HI-criticality task τ_i
      declare success and **return**
   else continue// Consider the next task
3) **for** each τ_i with $\chi_i = \text{HI}$
   a) $x \leftarrow \big(1 - 2C_i(\text{HI})/T_i\big)$
   b) repeat Step 2b and Step 2c
4) Declare failure and **return**

Fig. 3. The improved preprocessing phase.

value of x does not exist; we may apply heuristic strategies to attempt to determine such an x. One strategy would be to apply exhaustive search: consider all values of x (at an appropriate level of granularity), searching for one that causes the properties P1 and P2 to be satisfied. A somewhat less naive but computationally more efficient heuristic is described in pseudo-code form in Figure 3, as the Algorithm GLOBAL-PRAGMATIC. According to this heuristic, we iterate through only those values for x that, for each τ_i with $\chi_i = \text{HI}$, would make the regular task $(C_i(\text{LO}), xT_i)$ or $(C_i(\text{HI}), (1-x)T_i)$ have utilization $1/2$; for each such value of x, we test whether properties P1 and P2 are satisfied. The algorithm returns success upon finding the first such x for which P1 and P2 are both satisfied; if all these values of x fails either P1 or P2, the algorithm returns failure. The statements in Section V on properties P1 and P2 guarantees the correctness of Algorithm GLOBAL-PRAGMATIC.

The idea of considering only those specific candidate values for x can be briefly motivated as follows: In Algorithm fpEDF, if the tasks whose jobs get highest priority are fixed, the schedulability of the task system depends entirely on the values of U_{sum} and U_{max}. It can be shown that if the tasks whose jobs are assigned highest priority are fixed, then these two values change monotonically with x. Therefore we only check the values on the boundary of monotonic intervals, which are specifically, the values that change the assignment of the tasks whose jobs are assigned highest priority. It is evident that those are the values that make the regular task $(C_i(\text{LO}), xT_i)$ or $(C_i(\text{HI}), (1-x)T_i)$ have utilization $1/2$, as we described above. Thus in Algorithm GLOBAL-PRAGMATIC, we check these values to see if it is a valid assignment of x; although this is not necessarily guaranteed to always find an x that renders the system schedulable even if one exists, it is computationally more efficient than exhaustive search and our simulation evaluations (described in the next section) indicate that it seems to perform quite well.

VIII. Evaluation via simulation

We have conducted a series of simulation experiments to evaluate the effectiveness of Algorithm GLOBAL and Algorithm GLOBAL-PRAGMATIC in finding certifiably correct scheduling strategies. Our experiments were conducted upon randomly-generated task systems that were generated according to (a slight modification of) the workload-generation algorithm introduced by Guan et al. [8]. The input parameters for our workload generation algorithm are as follows:

- U_{bound}: The desired value of the larger of LO-criticality and HI-criticality utilization of the task system:
$$\max\Big(U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{LO}}(\tau), U_{\text{HI}}^{\text{HI}}(\tau)\Big) = U_{\text{bound}} \quad (8)$$
- $[U_L, U_U]$: Utilizations are uniformly drawn from this range; $0 \leq U_L \leq U_U \leq 1$.

TASKGEN($P, U_L, U_U, U_{\text{bound}}, Z_L, Z_U$)

▷ Let $\mathcal{U}(a, b)$ return a number uniformly drawn from $[a, b]$
1 $i \leftarrow 0$
2 $S_L \leftarrow 0$ ▷ Current value of $(U_{\text{LO}}^{\text{LO}} + U_{\text{HI}}^{\text{LO}})$
3 $S_H \leftarrow 0$ ▷ Current value of $U_{\text{HI}}^{\text{HI}}$
4 **repeat**
5 $i \leftarrow i + 1$
6 $u_H \leftarrow \mathcal{U}(U_L, U_U)$
7 $u_L \leftarrow u_H / \mathcal{U}(Z_L, Z_U)$
8 $\chi_i \leftarrow$ HI with prob. P; LO with prob. $(1 - P)$
9 **if** $(\chi_i \equiv \text{LO})$
10 **then**
$U_i(\text{LO}) \leftarrow \min(u_L, U_{\text{bound}} - S_L)$
11 $S_L \leftarrow S_L + U_i(\text{LO})$
12 **else**
$U_i(\text{HI}) \leftarrow \min(u_H, U_{\text{bound}} - S_H)$
13 $U_i(\text{LO}) \leftarrow \min(U_i(\text{HI}), u_L, U_{\text{bound}} - S_L)$
14 $S_H \leftarrow S_H + U_i(\text{HI})$
15 $S_L \leftarrow S_L + U_i(\text{LO})$
16 **until** $(\max(S_L, S_H) \geq U_{\text{bound}})$

Fig. 4. Pseudo-code for task generation procedure

- $[Z_L, Z_U]$: The ratio of the HI-criticality utilization of a task to its LO-criticality utilization is uniformly drawn from this range; $1 \leq Z_L \leq Z_U$.
- P: The probability that a task in a HI-criticality task; $0 \leq P \leq 1$.

Gives these parameters, the task-generation algorithm initializes the task system τ to be empty and repeatedly adds tasks τ_i, $i = 1, 2, \ldots$, until the utilization bound is met. The procedure is described in pseudo-code form in Figure 4.

We used our simulation platform to investigate several interesting questions concerning the behavior of our algorithm, including the following:

1) Does the behavior depend significantly on whether the LO-criticality system utilization ($U_{\text{LO}}^{\text{LO}} + U_{\text{HI}}^{\text{LO}}$) and the HI-criticality system utilization ($U_{\text{HI}}^{\text{HI}}$) take on similar or very different values? This question is particularly interesting since our speedup factor result (Corollary 2) depends only upon the larger of the two utilizations, and is independent of whether these utilization values are close to each other or not.
2) How does *increasing* the ratio of HI-criticality to LO-criticality utilization effect our algorithm? This is investigated by fixing all other parameters and varying the upper bound Z_U on the ratios of the utilizations. Larger values of the ratio imply that obtaining WCET bounds at greater levels of assurance results in increased pessimism; this could reflect, for instance, the presence of more advanced (and non-deterministic) features such as high-speed cache, speculative out-of-order execution, etc., in the CPUs.

In our experiments, we determined the fraction of randomly-generated task systems that are deemed to be schedulable by the algorithm under consideration, as a function of the ratio (U_{bound}/m), where m denotes the number of unit-speed processors in the platform (i.e., this ratio is the system utilization normalized by the number of processors). Some of our results are depicted graphically in Figures 5-11. In each graph, the fraction of systems that were determined to be schedulable is depicted on the y-axis, and the normalized utilization on the x-axis. Each data-point was obtained by randomly generating 1000 task systems, testing each for schedulability according to the appropriate algorithm, and calculating the fraction of systems deemed schedulable. The parameters used in generating these task systems (other than the normalized system utilization, which is depicted on the x-axis) are provided in the caption of the graph; e.g., the task systems for Figure 5 were generated using the parameters $U_L = 0.05$, $U_U = 0.75$, $Z_L = 1$, $Z_U = 8$, and $P = 0.5$. For each set of parameters, we conducted simulations on 2-processor, 4-processor, 8-processor, and 16-processor platforms. Three algorithms were compared:

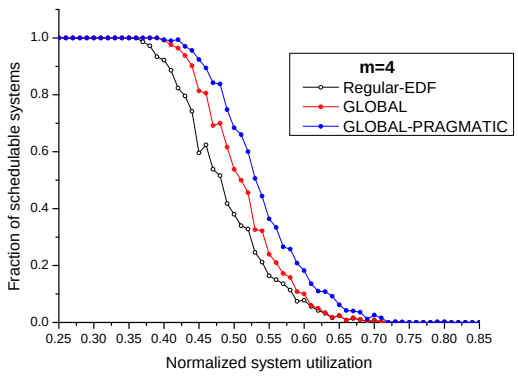
1) Regular EDF (i.e., EDF on the task system $\bigcup_i \{(C_i(\chi_i), T_i)\}$) – this corresponds to "reserving" capacity for a HI-criticality tasks according to its HI-criticality WCET;
2) Algorithm GLOBAL (as depicted in Figure 2) ; and
3) Algorithm GLOBAL-PRAGMATIC (as depicted in Figure 3).

Since a graph containing all the data for a given combination of parameters (such as Figure 5) is too dense to be visually parsed comfortably, we have split the information in this graph into two graphs (for Figure 5, these are the two graphs in Figure 6). The first graph in Figure 6 plots the acceptance fractions for the three algorithms being compared, for $m = 4$ processors; it is evident from this graph that Algorithm GLOBAL-PRAGMATIC performs better than Algorithm GLOBAL, which in turn performs significantly better than regular EDF. (In this particular graph, for example, for a normalized utilization equal to 0.5, the fractions of schedulable systems were 0.684, 0.538, and 0.380 for the three algorithms.) This same phenomenon was observed for all the parameter-settings we considered, although the exact amount of the improvement of one algorithm over the others may have differed; consequently, for subsequent parameter settings we only depict the results for the best of the algorithms (GLOBAL-PRAGMATIC), for the different numbers of processors considered.

A. Observations

Several trends are revealed by our simulation experiments. It is evident that both our algorithms consistently exhibit noticeable improvements over a simple EDF scheduler. We do not seek to make quantitative claims about the degree of such improvement based on simulation data, but we do notice some trends.

1) When the average utilization percentage is smaller than 0.3, the task system is always schedulable. This observation matches our speed-up factor computation, since $1/(\sqrt{5}+1) \approx 0.308$.



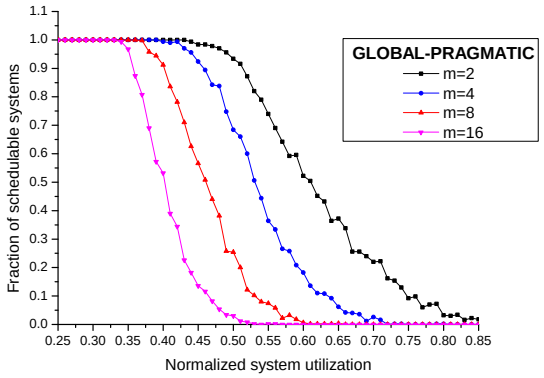


Fig. 6. $U_L = 0.05, U_U = 0.75, Z_L = 1, Z_U = 8, P = 0.3$

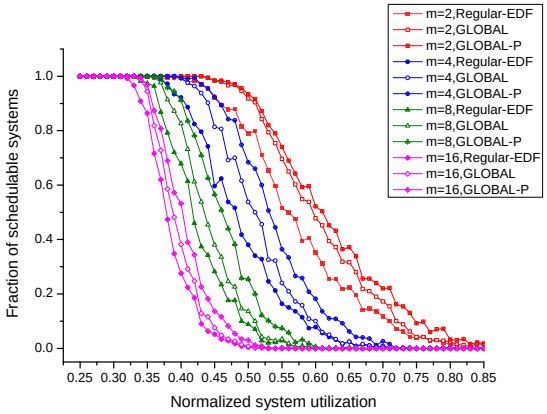


Fig. 5. $U_L = 0.05, U_U = 0.75, Z_L = 1, Z_U = 8, P = 0.3$

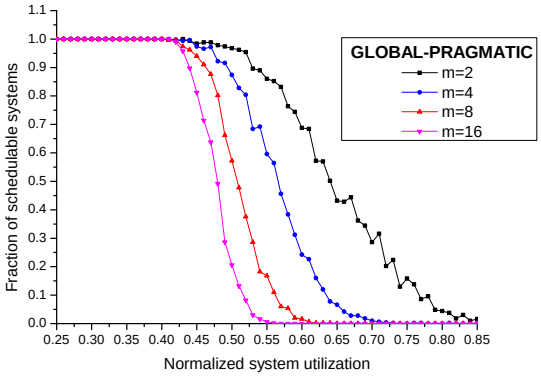


Fig. 7. $U_L = 0.05, U_U = 0.75, Z_L = 1, Z_U = 2, P = 0.5$

2) These improvements become more significant with increasing value of Z_U (i.e., as the ratio of the HI-criticality WCET to LO-criticality WCET increases). The intuitive explanation for this is that our algorithms take more advantage from processing LO-criticality and HI-criticality behaviors separately when the LO-criticality and HI-criticality behaviors overlap less.
3) When $C_i(\text{HI})$ is close to $C_i(\text{LO})$, our algorithms are not significantly superior to regular EDF. This is not unexpected: for such systems, there is not much difference between the LO-criticality and HI-criticality behavior of tasks, and hence reserving for the HI-criticality behavior (which, in effect, is what regular EDF does) is not inordinately expensive.

IX. Conclusions

Most prior scheduling-theoretic work on mixed-criticality systems has focused on uniprocessor scheduling. However, embedded real-time systems are increasingly coming to be implemented upon multicore and multiprocessor platforms; this motivates a need for the study of multiprocessor algorithms for scheduling mixed-criticality systems.

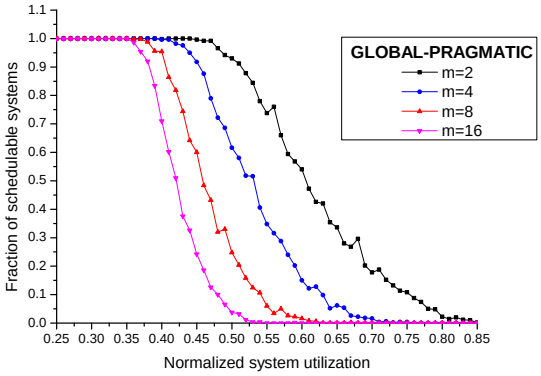


Fig. 8. $U_L = 0.05, U_U = 0.75, Z_L = 1, Z_U = 4, P = 0.5$

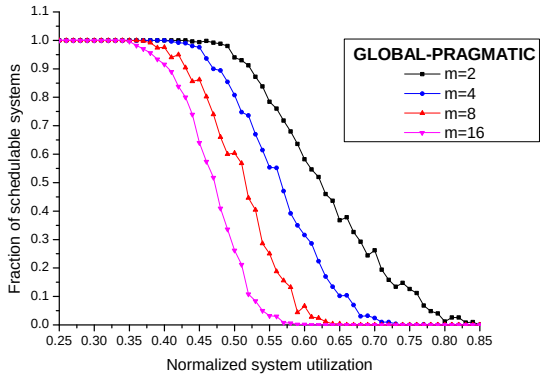


Fig. 9. $U_L = 0.05, U_U = 0.75, Z_L = 1, Z_U = 8, P = 0.5$

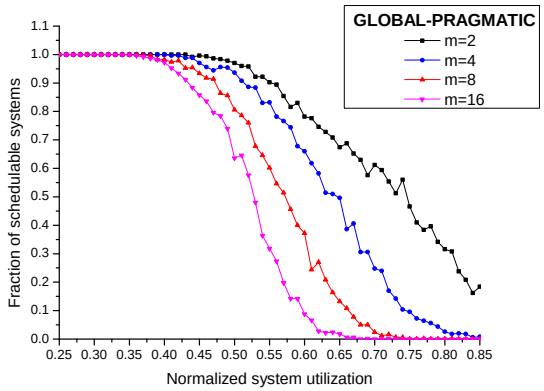


Fig. 10. $U_L = 0.05, U_U = 0.75, Z_L = 1, Z_U = 16, P = 0.1$

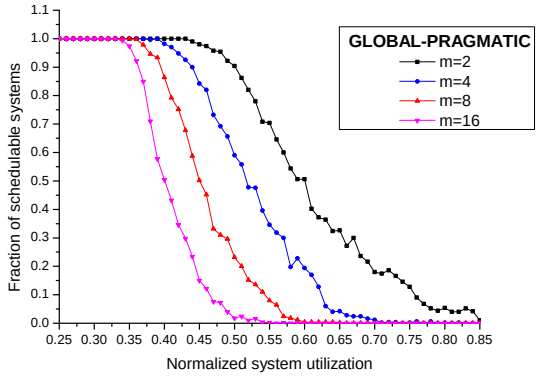


Fig. 11. $U_L = 0.3, U_U = 0.8, Z_L = 1, Z_U = 8, P = 0.3$

In this paper, we have taken a step towards meeting this need: we have studied the global scheduling of implicit-deadline mixed-criticality systems on identical multiprocessor platforms. We have derived, and proved the correctness of, an algorithm for scheduling such systems. We have derived a sufficient schedulability condition, and a processor speedup factor, for our algorithm. We have also conducted extensive simulation experiments in an attempt to better understand the behavior of our algorithm.

ACKNOWLEDGEMENT

Supported in part by NSF grants CNS 1016954 and CNS 1115284; ARO grant W911NF-09-1-0535; AFOSR grant FA9550-09-1-0549; and AFRL grant FA8750-11-1-0033.

REFERENCES

[1] T. Baker and S. Baruah. Sustainable multiprocessor scheduling of sporadic task systems. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, Dublin, July 2008. IEEE Computer Society Press.

[2] S. Baruah. Optimal utilization bounds for the fixed-priority scheduling of periodic task systems on identical multiprocessors. *IEEE Transactions on Computers*, 53(6), 2004.

[3] S. Baruah and A. Burns. Sustainable scheduling analysis. In *Proceedings of the IEEE Real-time Systems Symposium*, pages 159–168, Rio de Janeiro, December 2006. IEEE Computer Society Press.

[4] S. Baruah, A. Burns, and R. Davis. Response-time analysis for mixed criticality systems. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, Vienna, Austria, 2011. IEEE Computer Society Press.

[5] S. Baruah and G. Fohler. Certification-cognizant time-triggered scheduling of mixed-criticality systems. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, Vienna, Austria, 2011. IEEE Computer Society Press.

[6] S. K. Baruah, V. Bonifaci, G. D'Angelo, A. Marchetti-Spaccamela, S. van der Ster, and L. Stougie. Mixed-criticality scheduling of sporadic task systems. In *Proceedings of the 19th Annual European Symposium on Algorithms*, pages 555–566, Saarbrucken, Germany, September 2011. Springer-Verlag.

[7] N. Guan, P. Ekberg, M. Stigge, and W. Yi. Effective and efficient scheduling for certifiable mixed criticality sporadic task systems. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, Vienna, Austria, 2011. IEEE Computer Society Press.

[8] N. Guan and W. Yi. Improving the scheduling of certifiable mixed criticality sporadic task systems. Technical report, Uppsala University, 2012.

[9] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 37(4):617–643, 2000.

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

[11] M. Mollison, J. Erickson, J. Anderson, S. Baruah, and J. Scoredos. Mixed-criticality real-time scheduling for multicore systems. In *Proceedings of the IEEE International Conference on Embedded Systems and Software*, Bradford, UK, 2010. IEEE Computer Society Press.

[12] S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Proceedings of the Real-Time Systems Symposium*, pages 239–243, Tucson, AZ, December 2007. IEEE Computer Society Press.