Mixed-criticality scheduling on multiprocessors*

Sanjoy Baruah Bipasa Chattopadhyay Haohan Li Insik Shin

Abstract

The scheduling of mixed-criticality implicit-deadline sporadic task systems on identical multiprocessor platforms is considered. Two approaches, one for global and another for partitioned scheduling, are described. Theoretical analyses and simulation experiments are used to compare the global and partitioned scheduling approaches.

Keywords. Mixed criticalities; implicit-deadline sporadic tasks; multiprocessors; global scheduling; partitioned scheduling

1 Introduction

In mixed-criticality systems, multiple functionalities that may be of different degrees of importance or criticalities are implemented upon a common platform. The real-time systems research community is showing increasing interest in better understanding the scheduling of such mixed-criticality systems, motivated in large part by the increasing trend in embedded systems towards platform integration 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.

Mixed-criticality systems are increasingly being implemented on multiprocessor platforms. As more functionalities with different degrees of criticality are implemented on a common multiprocessor platform, mixed-criticality systems are becoming more complex, less uniform and predictable, and show greater variation in their performance. Certification of such systems is crucial to their successful deployment. 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.) 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

*Work supported by NSF grants CNS 1218693, CNS 1016954, and CNS 1115284; ARO grant W911NF-09-1-0535; AFOSR grant FA9550-09-1-0549; and AFRL grant FA8750-11-1-0033.

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.

In this paper we describe algorithms for global and partitioned scheduling of mixed-criticality systems that are geared towards achieving goals ***(i)*** and ***(ii)*** mentioned above. We compare and contrast the global and partitioning approaches both theoretically and experimentally.

Organization. The remainder of this paper is organized as follows. We formally define the workload and platform models that we use in Section 2. In Section 3, we describe some of the prior work that has been done on mixed-criticality scheduling and explain how our contributions fit in with this related work. In Section 4, we present a review of some prior results that are used in this paper. In Sections 5 and 6 respectively we describe and prove correct our global and partitioned scheduling algorithms respectively, and derive interesting properties that provide quantitative characterizations of the algorithms' worst-case behaviors. The results of some simulation experiments comparing the two approaches are reported in Section 7. In Section 8 we summarize our observations concerning the relative efficacy of the partitioned and the global approach to multiprocessor mixed-criticality scheduling.

2 System Model and Definitions

As described in Section 1 above, certification authorities(CAs) and system designers typically make different assumptions about the worst-case behavior of a system: the CA's assumptions are usually more conservative than those of the system designer. The system model that we assume is cognizant of the differences in the assumptions of the CA and the system designer and incorporates these differences.

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 CA (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.

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:

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

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.

[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., [8] 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_3	HI	2	10	20

Table 1: An example mixed-criticality implicit-deadline sporadic task system.

Example 1 *Consider the task system depicted in Table 1. 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.

In Sections 5 and 6 we describe our global and partitioning algorithms for scheduling MC task systems for the system model we assume. We also derive *speedup bounds* for these algorithms. Following is a brief discussion on the relevance of *speedup bound* in the context of mixed-criticality scheduling.

Speedup bound. Since the simpler problems of scheduling "regular" (i.e., non-MC) task systems on multiprocessor platforms is already known to be very difficult, it is

evident that MC scheduling is also very difficult. Hence it is very unlikely that we will be able to design efficient algorithms for MC scheduling on multiprocessors that are optimal; instead, we seek efficient algorithms that, while not optimal, exhibit behavior that has bounded deviation from optimality. One metric we use for quantifying such deviation from optimal behavior is the speedup bound of an algorithm:

Definition 1 (Speedup bound) *A scheduling algorithm A is said to have a speedup bound b ($b \geq 1$) if any task system that can be correctly scheduled by an optimal algorithm on m speed-1 processors is correctly scheduled by A on m speed-b processors.* ■

Other things being equal, we seek scheduling algorithms with a smaller speedup bound: the smaller the speedup bound of an algorithm, the closer to optimal its behavior is.

3 Context and Related work

A large body of recent research has addressed mixed-criticality scheduling for certifiability, primarily for systems implemented upon uniprocessor platforms (see, e.g., [32, 14, 24, 23, 16, 28, 18, 21, 8, 9, 18, 31, 20, 21, 5, 25, 33, 29, 30, 17] – this list is by no means exhaustive). However, safety-critical (and other) embedded systems are increasingly coming to be implemented on *multicore* CPU's. There is thus a need for research in the implementation and analysis of mixed-criticality systems on multiprocessor platforms. To our knowledge, there has however not been much work done on studying multiprocessor mixed-criticality scheduling. We briefly list the only prior publications that we are aware of on this topic.

- The paper [29] considers global fixed-priority scheduling of MC task systems; in contrast, our global scheduling algorithm (Section 5) is EDF-based.
- A few papers [27, 20] deal primarily with implementation issues, while considering a very simple workload model (all jobs have the same release time and deadline).
- Another paper [33] addresses some pragmatic issues concerning non-interference in memory access when tasks of different criticality co-exist on a multiprocessor platform.
- Some parts of [6] consider (1) the *global* multiprocessor scheduling of a finite collection of independent *jobs* (Section 4.1); and (2) the partitioned multiprocessor scheduling of implicit-deadline sporadic task systems (Section 4.2).

From among these papers, [6, Section 4.2] contains results most closely related to the research presented in this paper. We will therefore briefly describe these results.

The research in [6, Section 4.2] identifies a relationship between the mixed-criticality partitioning problem and a restriction of the well-known *vector scheduling* problem, and applies a polynomial-time approximation scheme (PTAS) for this vector scheduling problem from [13] to obtain, for any given constant ϵ, a polynomial-time partitioning algorithm that has a speedup bound no larger than $\left(\Phi + \epsilon\right)$, where Φ is the mathematical constant equal to $(\sqrt{5}+1)/2 \approx 1.618$, commonly known as the Golden

Ratio.[2] Although this result is theoretically significant, it is not of much relevance from an implementation perspective since all techniques known for implementing the PTAS from [12, 13] have run-times that are unacceptably high-degree polynomials in the number of tasks and the number of processors – to our knowledge, these PTAS's have never been implemented. In contrast, the partitioning algorithm that we derive in this paper can be implemented very efficiently to have a run-time that is proportional to the product of the number of tasks and the number of processors.

4 Review of related results

In this section we briefly discuss some algorithms and their properties that are relevant to the work presented in this paper. The algorithms we describe are the uniprocessor mixed-criticality scheduling algorithm EDF-VD [5] (Section 4.1), and the EDF-based multiprocessor global scheduling algorithm fpEDF [4], which schedules "regular" (i.e., non mixed-criticality) task systems (Section 4.2).

4.1 Algorithm EDF-VD

Algorithm EDF-VD (for ***E**arliest **D**eadline **F**irst with **V**irtual **D**eadlines*) of [5] is an algorithm for scheduling mixed-criticality implicit-deadline sporadic task systems upon a preemptive uniprocessor. We now provide a high-level description of EDF-VD.

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

1. An initial pre-processing phase 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 *pre-processing*, a schedulability test is performed to determine whether τ can be guaranteed to be successfully scheduled. If it is determined that τ can indeed be guaranteed to be successfully scheduled, then an additional parameter, called a *modified period* denoted $\hat{T}_i$, is computed for each HI-criticality task $\tau_i \in \tau$. It is always the case that $\hat{T}_i \leq T_i$.

Initial run-time scheduling is done according to the earliest-deadline first (EDF) algorithm [26, 15]. Since EDF is defined for regular, rather than mixed-criticality, task

[2] By applying some results from [5], this result can easily be improved to show that the speedup bound is actually no more than $\big((4/3) + \epsilon\big)$.

systems, we map the mixed-criticality tasks in τ to regular tasks as follows: each LO-criticality task $\tau_k = (\chi_k, C_k(\text{LO}), C_k(\text{HI}), T_k)$ is mapped to a regular implicit-deadline task $(C_k(\text{LO}), T_k)$, while each HI-criticality task $\tau_k = (\chi_k, C_k(\text{LO}), C_k(\text{HI}), T_k)$ 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.

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 EDF. This is done by mapping: each HI-criticality MC task $\tau_k = (\chi_k, C_k(\text{LO}), C_k(\text{HI}), T_k)$ in τ to a regular implicit-deadline task $(C_k(\text{HI}), T_k)$.

4.1.1 EDF-VD: Properties

The following properties of Algorithm EDF-VD, derived in [5]; will be used in later sections of this paper.

Theorem 1 (from [5]) *Any mixed-criticality implicit-deadline sporadic task system τ satisfying the property*

$$\max\big(U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{LO}}(\tau), U_{\text{HI}}^{\text{HI}}(\tau)\big) \leq 3/4$$

is successfully scheduled by EDF-VD on a preemptive unit-speed processor. ■

Theorem 1 above asserts that any MC implicit-deadline sporadic task system for which both the LO-criticality utilization $(U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{LO}}(\tau))$ and the HI-criticality utilization $(U_{\text{HI}}^{\text{HI}}(\tau))$ are $\leq 3/4$ is successfully scheduled by EDF-VD on a preemptive unit-speed processor. Since utilization not exceeding processor speed is a necessary condition for schedulability, a speedup bound of $4/3$ for EDF-VD immediately follows:

Corollary 1 *EDF-VD has a speedup bound no greater than* $4/3$. ■

Theorem 1 guarantees schedulability when both LO-criticality and HI-criticality utilizations are bounded by $3/4$; a more general test for when one of the utilizations is greater than $3/4$, and the other less than $3/4$, has also been derived:

Theorem 2 ((from [5])) *Any mixed-criticality implicit-deadline sporadic task system τ satisfying the property*

$$U_{\text{LO}}^{\text{LO}}(\tau) \leq \frac{1 - U_{\text{HI}}^{\text{HI}}(\tau)}{1 - \big(U_{\text{HI}}^{\text{HI}}(\tau) - U_{\text{HI}}^{\text{LO}}(\tau)\big)}$$

is successfully scheduled by EDF-VD on a preemptive unit-speed processor. ■

4.2 Algorithm fpEDF [4]

Algorithm fpEDF [4] is a global EDF-based algorithm for scheduling systems of non mixed-criticality implicit-deadline sporadic tasks upon identical multiprocessor platforms. The global mixed-criticality scheduling algorithm that we present in Section 5 below is based on Algorithm fpEDF; therefore, we describe it here and provide a brief overview of some results from [4].

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

Since preemptive uniprocessor EDF is known [26] to have a schedulable utilization equal to the speed of the processor on which it is implemented, the contrapositive of Theorem 3 yields the following corollary.

Corollary 2 *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* $(m+1)/2$. ■

5 Global Scheduling

Our global mixed-criticality scheduling approach extends the EDF-VD [5] uniprocessor mixed-criticality scheduling algorithm to multiprocessors, by applying the multiprocessor global scheduling algorithm fpEDF, described above, to mixed-criticality systems. This algorithm was first presented in [25]; we provide a detailed description in this section. In Section 5.1 we provide a high-level overview, and attempt to communicate the intuition behind the algorithm design by means of a very simple example. We will fill in the details with a more comprehensive description in Section 5.2, prove some important properties in Section 5.3, and in Section 5.4 we describe some pragmatic improvements that can be made to the algorithm.

5.1 Overview

In this section we provide a high-level overview of the global mixed-criticality scheduling algorithm. 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 a 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 $\hat{T}_i \leq T_i$.

Run-time scheduling is initially done according to Algorithm fpEDF [4] described in 4.2. 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 [7, 3] 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. [3]

Example 2 *Suppose we wish to schedule the mixed-criticality implicit-deadline sporadic task system depicted in Table 1, on a single unit-speed processor. We will see later*

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

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 [26], 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.*

■

5.2 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.

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:

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

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

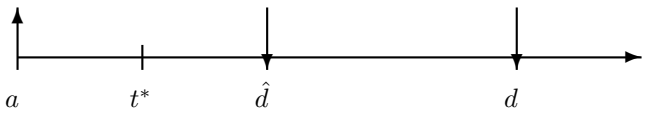


Figure 1: A HI-criticality job arrives at time a, with deadline at d. It is scheduled by Algorithm fpEDF using the modified deadline $\hat{d}$, which is $\leq d$. If no job executes for more than its LO-criticality WCET, then this job can complete by $\hat{d}$. If only the HI-criticality jobs execute 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 $[\hat{d}, d)$.

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

$$\bigcup_{\chi_i = \text{HI}} \left\{ (C_i(\text{HI}), T_i - \hat{T}_i) \right\} \tag{3}$$

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

In Section 5.1, 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 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 [7, 3] 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

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**.

Figure 2: The preprocessing phase.

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 3, 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 equiva-

lent 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.

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 $\chi_i \equiv \text{LO}$, the job is assigned a deadline equal to $t + T_i$.
      - if $\chi_i \equiv \text{HI}$, 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 $\Gamma := \text{HI}$.
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.)

5.3 Properties

Our algorithm is essentially a generalization to multiprocessors of the uniprocessor mixed-criticality scheduling algorithm EDF-VD of [6].

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

are each (separately) EDF-schedulable on a unit-speed processor.[4] 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 5.2.

The following theorem concerning EDF-VD is a generalization to Theorem 1 in [6].

Theorem 4 *Any task system τ, with $U^{\text{HI}}_{\text{HI}}(\tau) \leq s$, satisfying*

$$U^{\text{LO}}_{\text{LO}}(\tau) + \min\Big(U^{\text{HI}}_{\text{HI}}(\tau), \frac{U^{\text{LO}}_{\text{HI}}(\tau)}{1 - U^{\text{HI}}_{\text{HI}}(\tau)/s}\Big) \leq s \tag{4}$$

is successfully scheduled by Algorithm EDF-VD on a preemptive speed-s processor. ■

Proof:

If a task system τ satisfies that

$$\sum_{\forall \tau_i : \chi_i = \text{LO}} \frac{C_i(\text{LO})}{T_i} + \sum_{\forall \tau_i : \chi_i = \text{HI}} \frac{C_i(\text{LO})}{xT_i} \leq s,$$

which is

[4] 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 5.2.

$$U_{\text{LO}}^{\text{LO}}(\tau) + \frac{U_{\text{HI}}^{\text{LO}}(\tau)}{x} \leq s$$

or equivalently

$$x \geq \frac{U_{\text{HI}}^{\text{LO}}(\tau)}{s - U_{\text{LO}}^{\text{LO}}(\tau)}, \tag{5}$$

then the task collection

$$\left(\bigcup_{\chi_i = \text{LO}} \{(C_i(\text{LO}), T_i)\} \right) \bigcup \left(\bigcup_{\chi_i = \text{HI}} \left\{ (C_i(\text{LO}), \hat{T}_i) \right\} \right)$$

is EDF-schedulable on a preemptive speed-s processor;

If the task system τ satisfies that

$$\sum_{\forall \tau_i : \chi_i = \text{HI}} \frac{C_i(\text{HI})}{(1-x)T_i} \leq s,$$

which is

$$\frac{U_{\text{HI}}^{\text{HI}}(\tau)}{1-x} \leq s$$

or equivalently

$$x \leq 1 - \frac{U_{\text{HI}}^{\text{HI}}(\tau)}{s}, \tag{6}$$

then the task collection

$$\bigcup_{\chi_i = \text{HI}} \left\{ (C_i(\text{HI}), T_i - \hat{T}_i) \right\}$$

is EDF-schedulable on a preemptive speed-s processor.

According to Property P1 and P2, if these both task collections are EDF-schedulable, the original task system τ is schedulable by Algorithm EDF-VD on a preemptive speed-s processor.

Noticing inequality (5) and (6), it's clear that if

$$\frac{U_{\text{HI}}^{\text{LO}}(\tau)}{s - U_{\text{LO}}^{\text{LO}}(\tau)} \leq 1 - \frac{U_{\text{HI}}^{\text{HI}}(\tau)}{s}, \tag{7}$$

then the existence of x is secured. Therefore, by transforming (7) to

$$U_{\text{LO}}^{\text{LO}}(\tau) + \frac{U_{\text{HI}}^{\text{LO}}(\tau)}{1 - U_{\text{HI}}^{\text{HI}}(\tau)/s} \leq s, \tag{8}$$

we've shown that the inequality (8) is a sufficient condition to schedule the task system τ by Algorithm EDF-VD on a preemptive speed-s processor.

It is trivial to show that a task system τ satisfying

$$U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{HI}}(\tau) \leq s \tag{9}$$

is schedulable on a preemptive speed-s processor. The combination of inequality (8) and (9) completes the proof. ■

Using this theorem and Corollary 2, the following ***sufficient schedulability condition*** can be derived for our multiprocessor mixed-criticality scheduling algorithm:

Theorem 5 *Any task system τ, with $U_{\text{HI}}^{\text{HI}}(\tau) \leq \frac{m+1}{2}$, satisfying*

$$U_{\text{LO}}^{\text{LO}}(\tau) + \min\Big(U_{\text{HI}}^{\text{HI}}(\tau), \frac{U_{\text{HI}}^{\text{LO}}(\tau)}{1 - U_{\text{HI}}^{\text{HI}}(\tau) \cdot 2/(m+1)}\Big) \leq \frac{m+1}{2} \tag{10}$$

is successfully scheduled by our algorithm on m preemptive unit-speed processors. ■

The following theorem is a generalization to Theorem 2 in [6].

Theorem 6 *Any task system τ satisfying*

$$\max\Big(U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{LO}}(\tau), U_{\text{HI}}^{\text{HI}}(\tau)\Big) \leq s \tag{11}$$

is successfully scheduled by Algorithm EDF-VD on a preemptive speed-$\big(\frac{\sqrt{5}+1}{2}s\big)$ processor. ■

Proof: Denoting $\phi = (\sqrt{5}+1)/2$, if $U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{HI}}(\tau) \leq \phi s$, then it is trivial to show that the task system τ is successfully scheduled by Algorithm EDF-VD on a preemptive speed-ϕs processor. Otherwise, we need to show that

$$U_{\text{LO}}^{\text{LO}}(\tau) + \frac{U_{\text{HI}}^{\text{LO}}(\tau)}{1 - U_{\text{HI}}^{\text{HI}}(\tau)/\phi s} \leq \phi s,$$

or equivalently

$$U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{HI}}(\tau) - \frac{U_{\text{LO}}^{\text{LO}}(\tau)U_{\text{HI}}^{\text{HI}}(\tau)}{\phi s} \leq \phi s.$$

By respectively applying $\max\left(U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{LO}}(\tau), U_{\text{HI}}^{\text{HI}}(\tau)\right) \leq s$, $U_{\text{LO}}^{\text{LO}}(\tau) \leq s < \phi s$ and $U_{\text{LO}}^{\text{LO}}(\tau) > \phi s - U_{\text{HI}}^{\text{HI}}(\tau) \geq (\phi - 1)s$, we have

$$\begin{aligned}
&U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{HI}}(\tau) - \frac{U_{\text{LO}}^{\text{LO}}(\tau)U_{\text{HI}}^{\text{HI}}(\tau)}{\phi s} \\
\leq & s + \left(1 - \frac{U_{\text{LO}}^{\text{LO}}(\tau)}{\phi s}\right) U_{\text{HI}}^{\text{HI}}(\tau) \\
\leq & s + \left(1 - \frac{U_{\text{LO}}^{\text{LO}}(\tau)}{\phi s}\right) s \\
< & s + \left(1 - \frac{\phi - 1}{\phi}\right) s = \phi s.
\end{aligned}$$

By Theorem 4, the task system τ is successfully scheduled by Algorithm EDF-VD on a preemptive speed-ϕs processor. ■

Once again, it follows as a consequence of this theorem and Corollary 2 that

Theorem 7 *Any task system τ satisfying*

$$\max\Big(U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{LO}}(\tau), U_{\text{HI}}^{\text{HI}}(\tau)\Big) \leq m \tag{12}$$

is successfully scheduled by our algorithm on m preemptive speed-$\big(\sqrt{5}+1\big)$ processors. ■

We can use Theorem 7 above to obtain a ***processor speedup bound*** [22] for our multiprocessor scheduling algorithm:

Corollary 3 *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 7 above. ■

5.4 Pragmatic improvements

The algorithm in Figure 2 describes a technique for computing a scaling factor x; Theorem 5 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 5? 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 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 5.2 on properties P1 and P2 guarantees the correctness of Algorithm GLOBAL-PRAGMATIC.

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**

Figure 3: The improved preprocessing phase.

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 the simulation evaluations described in [25] indicate that it seems to perform quite well.

6 Partitioned Scheduling

We start with an **overview** of our partitioning algorithm. It proceeds in two phases:

1. During the first phase each HI-criticality task is assigned to some processor while ensuring that the cumulative HI-criticality utilization assigned to each processor does not exceed $3/4$.

2. During the second phase each LO-criticality task is assigned to some processor while ensuring that the cumulative LO-criticality utilization assigned to each processor also does not exceed $3/4$.

Observe that by Theorem 1, such an assignment procedure ensures that each processor remains schedulable by EDF-VD. The algorithm reports failure if it fails to successfully assign every task.

We now fill in the **details**. Let τ denote the implicit-deadline sporadic task system that is to be partitioned amongst m processors. Let us assume that there are n tasks in τ, of which n_1 are HI-criticality tasks. Without loss of generality, assume that $\tau_1, \tau_2, \ldots, \tau_{n_1}$ are the HI-criticality tasks, and $\tau_{n_1+1}, \ldots, \tau_n$ the LO-criticality ones. Let $\pi_1, \pi_2, \ldots, \pi_m$ denote the m processors. Figure 4 gives a pseudo-code representation of our algorithm, MC-PARTITION.

Let us suppose that tasks $\tau_1, \tau_2, \ldots, \tau_{i-1}$ have been successfully assigned. We now explain how the task τ_i is assigned to a processor.

For any processor π_k, let $\tau(\pi_k)$ denote the tasks from amongst $\tau_1, \tau_2, \ldots, \tau_{i-1}$ that have already been assigned to it. Algorithm MC-PARTITION assigns the task τ_i to any processor π_k satisfying the following condition. If $i \leq n_1$ (i.e., if τ_i is a HI-criticality task) then

$$\left(U_i(\text{HI}) + \sum_{\tau_j \in \tau(\pi_k)} U_j(\text{HI}) \right) \leq \frac{3}{4} \tag{13}$$

```
MC-PARTITION(τ, m)
    ▷ τ = {τ_1, ..., τ_n} is to be partitioned on m identical, unit-capacity processors
      denoted π_1, ..., π_m. Tasks τ_1, ..., τ_{n_1} are HI-criticality tasks; tasks
      τ_{n_1+1}, ..., τ_n are LO-criticality tasks. The set of tasks assigned to processor
      π_k is denoted as τ(π_k); initially, τ(π_k) ← ∅ for all k.
 1  for i ← 1 to n_1 ▷ Phase 1: HI-criticality tasks
 2          for k ← 1 to m
 3                  if τ_i satisfies Condition 13 on processor π_k
                      then ▷ assign τ_i to π_k;
 4                           τ(π_k) ← τ(π_k) ⋃ {τ_i}
 5                           break;
 6          end (of inner for loop)
 7          if (k > m) return PARTITIONING FAILED
 8  end (of outer for loop)
 9  for i ← (n_1 + 1) to n ▷ Phase 2: LO-criticality tasks
10          for k ← 1 to m
11                  if τ_i satisfies Condition 14 on processor π_k
                      then ▷ assign τ_i to π_k;
12                           τ(π_k) ← τ(π_k) ⋃ {τ_i}
13                           break;
14          end (of inner for loop)
15          if (k > m) return PARTITIONING FAILED
16  end (of outer for loop)
17  return PARTITIONING SUCCEEDED
```

Figure 4: Pseudo-code for partitioning algorithm

else (i.e., $i > n_1$ and τ_i is hence a LO-criticality task)

$$\left(U_i(\text{LO}) + \sum_{\tau_j \in \tau(\pi_k)} U_j(\text{LO}) \right) \leq \frac{3}{4} \tag{14}$$

If no such π_k exists, then Algorithm MC-PARTITION declares failure: it is unable to partition τ upon the m-processor platform.

The following lemma asserts that, in assigning a task τ_i to a processor π_k, Algorithm MC-PARTITION does not adversely affect the schedulability of the tasks previously assigned to the processors.

Lemma 1 *If the tasks previously assigned to each processor were schedulable on that processor by EDF-VD and Algorithm* MC-PARTITION *assigns task τ_i to processor π_k, then the tasks assigned to each processor (including processor π_k) remain schedulable on that processor by EDF-VD.*

Proof: Observe that the schedulability of the processors other than processor π_k is not affected by the assignment of task τ_i to processor π_k. It remains to demonstrate that, if the tasks assigned to each processor were schedulable by EDF-VD on π_k prior to the assignment of τ_i and Algorithm MC-PARTITION assigns τ_i to π_k, then the tasks on π_k remain schedulable by EDF-VD after adding τ_i. To see that this is true, we consider two cases.

- If $i \leq n_1$, Condition 13 must hold for Algorithm MC-PARTITION to assign τ_i to π_k. This condition ensures that the sum of the HI-criticality utilizations of all HI-criticality tasks assigned to processor π_k remains $\leq 3/4$; since each task's LO-criticality utilization is no greater than its HI-criticality utilization, this also means that the sum of the LO-criticality utilizations of all tasks assigned to processor π_k also remains $\leq 3/4$.
- If $i > n_1$, Condition 14 must hold for Algorithm MC-PARTITION to assign τ_i to π_k. And, this condition ensures that the sum of the LO-criticality utilizations of all tasks assigned to processor π_k remains $\leq 3/4$ while the sum of the HI-criticality utilizations of HI-criticality tasks does not change.

It is thus the case that both the sum of the HI-criticality utilizations and the sum of the LO-criticality utilizations upon each processor remains $\leq 3/4$; the correctness of the lemma then follows from Theorem 1. ■

The correctness of Algorithm MC-PARTITION can now be established by repeated applications of Lemma 1.

Theorem 8 *If Algorithm* MC-PARTITION *returns* PARTITIONING SUCCEEDED *on task system τ, then the resulting partitioning is schedulable by EDF-VD.*

Proof: Observe that Algorithm MC-PARTITION returns PARTITIONING SUCCEEDED if and only if it has successfully assigned each task in τ to some processor.

Prior to the assignment of task τ_1, each processor has been assigned no tasks, and is therefore trivially schedulable by EDF-VD. It follows from Lemma 1 that all processors remain schedulable by EDF-VD after each task assignment as well. Hence, all processors are schedulable by EDF-VD after all tasks in τ have been successfully assigned. ■

Run-time complexity. Algorithm MC-PARTITION can be implemented to maintain, for each processor, the cumulative HI-criticality and LO-criticality utilizations of all the tasks that have been assigned to that processor thus far. For each task τ_i and each processor π_k, Condition 13 or Condition 14 can then be evaluated in constant time. Therefore the i'th task can be assigned in $\mathcal{O}(m)$ time, for each i; this yields an overall run-time of $\mathcal{O}(n \times m)$.

6.1 A speedup bound for Algorithm MC-PARTITION

We now derive a *sufficient* schedulability condition for Algorithm MC-PARTITION in Lemma 2 below, and use this schedulability condition to derive a speedup bound for Algorithm MC-PARTITION in Theorem 9.

We would like to stress that Lemma 2 is not intended to be used as a schedulability test to determine whether Algorithm MC-PARTITION would successfully schedule a given sporadic task system – since the algorithm itself runs efficiently in polynomial time, the "best" (i.e., most accurate) polynomial-time schedulability test for determining whether a particular task system is successfully scheduled by it is to actually run Algorithm MC-PARTITION.

Lemma 2 *Suppose that Algorithm* MC-PARTITION *fails to assign some task τ_i. One or both of the following conditions must hold:*

$$U_{\text{HI}}^{\text{HI}}(\tau) > \left(\frac{3}{4}m - (m-1)\,U_i(\text{HI})\right) \tag{15}$$

$$\left(U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{LO}}(\tau)\right) > \left(\frac{3}{4}m - (m-1)\,U_i(\text{LO})\right) \tag{16}$$

Proof: Let us first consider the case when $i \leq n_1$. Since τ_i cannot be accommodated on any processor, Condition 13 must be violated for task τ_i on each of the m processors. Summing the negation of Condition 13 across all m processors, we have

$$\begin{aligned}
&\left(\frac{3}{4} - U_i(\text{HI})\right) \times m < \sum_{j=i}^{i-1} U_j(\text{HI}) \\
\Leftrightarrow \quad &\frac{3}{4}m - mU_i(\text{HI}) + U_i(\text{HI}) < \sum_{j=i}^{i-1} U_j(\text{HI}) + U_i(\text{HI}) \\
\Leftrightarrow \quad &\frac{3}{4}m - (m-1)U_i(\text{HI}) < \sum_{j=i}^{i} U_j(\text{HI}) \\
\Rightarrow \quad &\frac{3}{4}m - (m-1)U_i(\text{HI}) < U_{\text{HI}}^{\text{HI}}(\tau)
\end{aligned}$$

which is as claimed.

Now let us consider when $i > n_1$. Since τ_i cannot be accommodated on any processor, Condition 14 must be violated for task τ_i on each of the m processors. Summing the negation of Condition 14 across all m processors, we have

$$\begin{aligned}
&\left(\frac{3}{4} - U_i(\text{LO})\right) \times m < \sum_{j=i}^{i-1} U_j(\text{LO}) \\
\Leftrightarrow \quad &\frac{3}{4}m - mU_i(\text{LO}) + U_i(\text{LO}) < \sum_{j=i}^{i-1} U_j(\text{LO}) + U_i(\text{LO}) \\
\Leftrightarrow \quad &\frac{3}{4}m - (m-1)U_i(\text{LO}) < \sum_{j=i}^{i} U_j(\text{LO}) \\
\Rightarrow \quad &\frac{3}{4}m - (m-1)U_i(\text{LO}) < U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{LO}}(\tau)
\end{aligned}$$

which, too, is as claimed in the lemma. ■

Using Lemma 2 above, we now derive a speedup bound for our partitioning algorithm.

Theorem 9 *The speedup bound of Algorithm* MC-PARTITION *on an m-processor platform is $(\frac{8m-4}{3m})$.*

Proof: To prove this, we must show that any MC implicit-deadline sporadic task system that can be partitioned upon an m-processor platform by an optimal algorithm can be partitioned by Algorithm MC-PARTITION upon an m-processor platform in which each processor is $(\frac{8m-4}{3m})$ times as fast.

Let us assume that $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ can be scheduled by an optimal scheduling algorithm on m processors each of computing capacity equal to ξ. It must therefore be the case that

$$\begin{aligned} U_i(\text{LO}) &\leq \xi \quad \text{for each } i, 1 \leq i \leq n \\ U_i(\text{HI}) &\leq \xi \quad \text{for each } i, 1 \leq i \leq n_1 \\ U^{\text{LO}}_{\text{LO}}(\tau) + U^{\text{LO}}_{\text{HI}}(\tau) &\leq m\xi \\ U^{\text{HI}}_{\text{HI}}(\tau) &\leq m\xi \end{aligned}$$

Suppose that Algorithm MC-PARTITION fails to partition τ on m unit-capacity processors. By Lemma 2 above, it must be the case that at least one of Conditions 15 or 16 holds. If Condition 15 holds, it must be the case that

$$\begin{aligned} & U^{\text{HI}}_{\text{HI}}(\tau) > \left(\frac{3}{4}m - (m-1)\, U_i(\text{HI})\right) \\ \Leftrightarrow \quad & m\xi > \frac{3}{4}m - (m-1)\xi \\ \Leftrightarrow \quad & (2m-1)\xi > \frac{3}{4}m \\ \Leftrightarrow \quad & \xi > \frac{3m}{4(2m-1)} \end{aligned}$$

Similarly if Condition 16 holds, it must be the case that

$$\begin{aligned} & U^{\text{LO}}_{\text{LO}}(\tau) + U^{\text{LO}}_{\text{HI}}(\tau) > \left(\frac{3}{4}m - (m-1)\, U_i(\text{LO})\right) \\ \Leftrightarrow \quad & m\xi > \frac{3}{4}m - (m-1)\xi \\ \Leftrightarrow \quad & (2m-1)\xi > \frac{3}{4}m \\ \Leftrightarrow \quad & \xi > \frac{3m}{4(2m-1)} \end{aligned}$$

We have shown that for either of Conditions 15 or 16 to hold, ξ must exceed $\frac{3m}{4(2m-1)}$. Hence if $\xi \leq \frac{3m}{4(2m-1)}$ then τ is successfully scheduled by Algorithm MC-PARTITION on m unit-speed processors; equivalently, if $\xi \leq 1$ then τ is successfully scheduled by Algorithm MC-PARTITION on a speed-$\frac{4(2m-1)}{3m}$ processors, as claimed by the theorem. ■

We note that $\frac{4(2m-1)}{3m} < 8/3$ for all $m \geq 1$, asymptotically approaching $8/3$ as $m \to \infty$. Hence, $8/3 \approx 2.67$ is an upper bound on the speedup of Algorithm MC-PARTITION, for all values of m.

For the global scheduling algorithm of [25], a speedup bound of $(\sqrt{5}+1) \approx 3.24$ was derived [25, Corollary 2]. Since smaller speedup bounds correspond to better

(closer to optimal) behavior, from the perspective of the speedup bounds computed thus far, Algorithm MC-PARTITION has better behavior than the global scheduling algorithm of [25]. However, the speedup bound of the global scheduling algorithm may be improved and we leave that as future work.

6.2 Pragmatic improvements

We now describe a series of modifications to Algorithm MC-PARTITION; although these modifications do not change the speedup bound, they do represent improvements in the sense that the modified algorithm is able to successfully schedule some task systems that the original algorithm could not.

6.2.1 Algorithm MC-PARTITION-UT-0.75

This version incorporates two modifications:

§**1: Preprocessing tasks with** $U_i(\text{HI}) > 3/4$**.** Algorithm MC-PARTITION is modified to incorporate the partitioning of tasks with $3/4 < U_i(\text{HI}) \leq 1$. The modification assigns one such high criticality task per processor prior to partitioning other tasks (we call this the *pre-processing phase*.) Each processor upon which a task was assigned during this pre-processing phase will only be assigned HI-criticality tasks; hence during Phase 1 of Algorithm MC-PARTITION each such processor π_k is assigned tasks so long as the resulting HI-criticality utilization on π_k does not exceed 1:

$$\left(U_i(\text{HI}) + \sum_{\tau_j \in \tau(\pi_k)} U_j(\text{HI}) \right) \leq 1 \tag{17}$$

(For the remaining processors, Condition 13 remains the requirement for assigning tasks during phase 1.)

§**2: Improved utilization during phase 2.** Condition 14 is based upon Theorem 1; as stated in Section 4.1 Theorem 2 is a superior sufficient schedulability test to the one in Theorem 1. We therefore replace Condition 14 with the following condition:

$$\left(U_i(\text{LO}) + \sum_{\tau_j \in \tau(\pi_k) \wedge \chi_i = \text{LO}} U_j(\text{LO}) \right) \leq \frac{1 - U_{\text{HI}}^{\text{HI}}(\tau(\pi_k))}{1 - \left(U_{\text{HI}}^{\text{HI}}(\tau(\pi_k)) - U_{\text{HI}}^{\text{LO}}(\tau(\pi_k)) \right)} \tag{18}$$

It follows from Theorem 2 that satisfying Condition 18 will ensure that the system is schedulable. Furthermore, it is possible that tasks with LO-criticality utilization $> 3/4$ will be accommodated upon some processor (on which the cumulative HI-criticality utilization assigned during phase 1 is strictly less than $3/4$).

6.2.2 Algorithm MC-PARTITION-UT-1

This is obtained by replacing Condition 13 by the following:

$$\left(U_i(\text{HI}) + \sum_{\tau_j \in \tau(\pi_k)} U_j(\text{HI}) \right) \leq 1$$

Also, Condition 14 for low criticality tasks is replaced by Condition 18. Note, that with this modification we do not need the pre-processing phase for high criticality tasks.

6.2.3 Algorithm MC-PARTITION-UT-INC

This is obtained by replacing Condition 13 by:

$$\left(U_i(\text{HI}) + \sum_{\tau_j \in \tau(\pi_k)} U_j(\text{HI}) \right) \leq val \tag{19}$$

where val is variable that iteratively takes on values in the range $[0.5, 1]$ (in pre-determined steps). The intuition behind this modification is that depending upon the value of val the high criticality tasks are assigned to processors differently, which in turn affects the partitioning of the low criticality tasks. As a result, different values of val might result in a success or failure in partitioning different task systems. In the pre-processing phase of Algorithm MC-PARTITION-UT-INC one high criticality task with utilization greater than val is assigned per processor. If m' is the number of processors to which tasks were assigned in the pre-processing phase then the remaining high criticality are assigned to the processors while ensuring that Condition 17 is satisfied on the m' processors and Condition 19 is satisfied on the processors excluding the m' processors. Also, Condition 14 for low criticality tasks is replaced by Condition 18. Algorithm MC-PARTITION-UT-INC returns PARTITIONING FAILED only if partitioning fails for all the values of val that are considered.

It is evident that Algorithm MC-PARTITION-UT-INC dominates both Algorithms MC-PARTITION-UT-0.75 and MC-PARTITION-UT-1, since MC-PARTITION-UT-INC checks with different values of val, including 0.75 and 1, and returns PARTITIONING FAILED only if the partitioning failed for all the values that were considered.

7 Evaluation via Simulation

In this section we experimentally compare the global and partitioning approaches presented in Sections 5 and 6 respectively. Our experiments were conducted upon randomly-generated task systems that were generated using the task generation algorithm described in [25] which is a slight modification of the workload-generation algorithm introduced by Guan et al. [19]. 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:

$$\max\Big(U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{LO}}(\tau), U_{\text{HI}}^{\text{HI}}(\tau)\Big) = U_{\text{bound}} \tag{20}$$

- $[U_L, U_U]$: The high criticality utilizations of tasks are uniformly drawn from this range; $0 \leq U_L \leq U_U \leq 1$.

- $[Z_L, Z_U]$: 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 is a HI-criticality task; $0 \leq P \leq 1$. If a task becomes a low criticality task then the high criticality utilization of the task is set to 0.

Given 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.

After generating the task sets, 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 5-9.

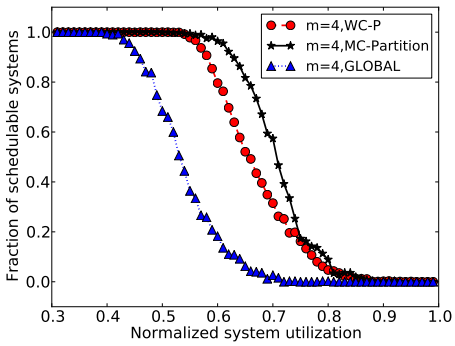
In each graph, the fraction of systems that were determined to be schedulable is depicted on the y-axis, and the normalized utilization on the x-axis. Each data-point was obtained by randomly generating 1000 task systems, testing each for schedulability according to the appropriate algorithm, and calculating 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 $P = 0.3$. For each set of parameters, we conducted simulations on 2-processor, 4-processor, 8-processor, and 16-processor platforms. Three algorithms were compared:

1. WORST-CASE-PARTITION, (i.e., partition the task system $\bigcup_i \{(C_i(\chi_i), T_i)\}$) – this corresponds to partitioning the high criticality tasks as per their high criticality utilization and the low criticality tasks as per their low criticality utilization such that for each processor π_k

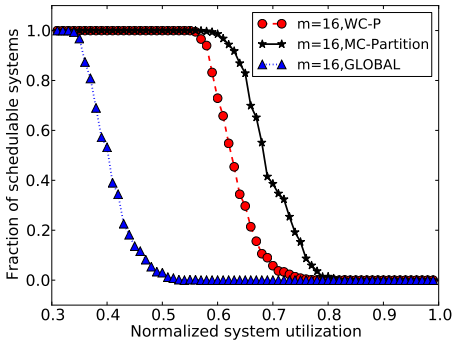
$$\sum_{\tau_j \in \tau(\pi_k)} C_j(\chi_j)/T_j \ \leq 1.$$

2. Algorithm MC-PARTITION (as depicted in Figure 4)

3. Algorithm GLOBAL from [25][5].



(a)



(b)

Figure 5: $U_L = 0.05, U_U = 0.75, Z_L = 1, Z_U = 8, P = 0.3$

Partitioning vs global. The graphs in Figure 5 plot the acceptance fractions for the three algorithms being compared, for $m = 4$ and $m = 16$ processors; it is evident from these graphs that Algorithm MC-PARTITION performs better than WORST-CASE-PARTITION and both the partitioning algorithms perform significantly better than Algorithm GLOBAL.

This same phenomenon was observed for all the parameter-settings we considered. We stress that these results are comparing the performance of MC-PARTITION *without* any pragmatic improvements incorporated, with the performance of a version of GLOBAL that incorporates all pragmatic improvements proposed for it in [25].

Effectiveness of the pragmatic improvements. We separately compared Algorithm WORST-CASE-PARTITION and MC-PARTITION with the pragmatic improvements described in Section 6.2, namely:

1. Algorithm MC-PARTITION-UT-0.75
2. Algorithm MC-PARTITION-UT-1
3. Algorithm MC-PARTITION-UT-INC, in our experiments we iterate the values for val from 0.5 to 1.0 in increments of 0.01.

The graphs in Figure 6 plot the acceptance fractions for $m = 4$ and $m = 16$ processors for the algorithms being compared. As expected, we observe that Algorithm MC-PARTITION-UT-INC does better than Algorithm WORST-CASE-PARTITION, Algorithm MC-PARTITION, and the other pragmatic improvements described in Section 6.2.

The graphs in Figures 7-9 show the effect on Algorithm MC-PARTITION-UT-INC, of varying different parameter settings in the workload-generation procedure.

We considered different values of Z_U (Z_L was set equal to one in all our experiments), thus considering different ratios between HI-criticality WCET and LO-criticality WCET for HI-criticality tasks. We noticed that the effect of different values of Z_U is most prominent when the task system is balanced (in a *balanced* task system the total HI-criticality utilization of the task system is equal to the total LO-criticality utilization of the task system, i.e. $U_{HI} = U_{LO}$). For generating balanced task systems we omitted the parameter P from the task generation algorithm described above. Instead we generated all the high criticality tasks until U_{HI} was equal to U_{bound}, defined in Equation 20, and then generated all the low criticality tasks until U_{LO} was equal to U_{bound}.

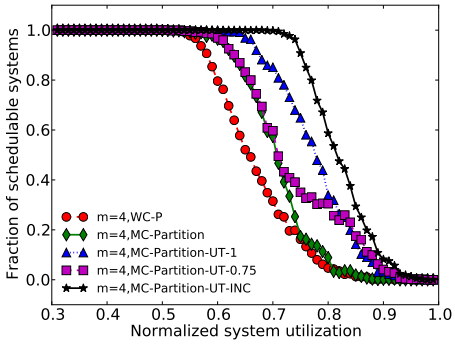
To explore the effect of having different total HI-criticality and LO-criticality utilizations for the system (i.e. an *imbalanced* task system), we set P such that the expectation of LO-criticality utilization and HI-criticality utilization are in the desired ratio.

We also varied the values of U_U and U_L to observe the affect of different utilization ranges of tasks.

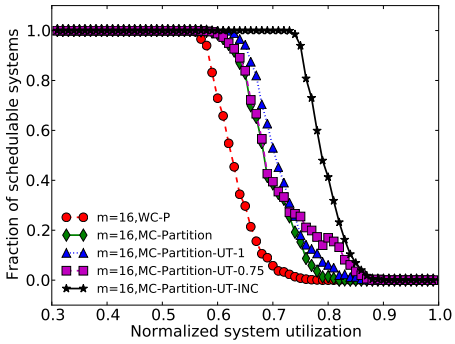
Several trends are revealed by our simulation experiments.

1. For balanced task systems, as Z_U increases, i.e. as the ratio between the HI-criticality and LO-criticality WCET for HI-criticality tasks increases, the fraction

[5]There are two global scheduling algorithms described in [25]. One of the algorithms is experimentally shown to be better than the other. In this experiment we use the better of the two algorithms.

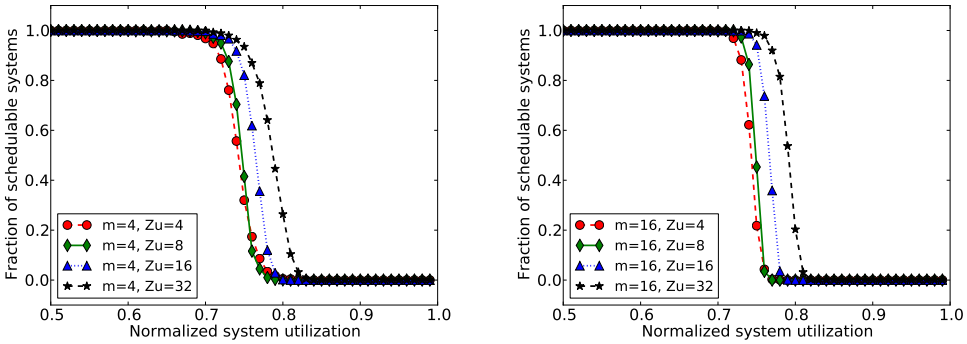


(a)



(b)

Figure 6: $U_L = 0.05, U_U = 0.75, Z_L = 1, Z_U = 8, P = 0.3$



(a) (b)

Figure 7: MC-PARTITION-UT-INC, $U_L = 0.05, U_U = 0.75, Z_L = 1$

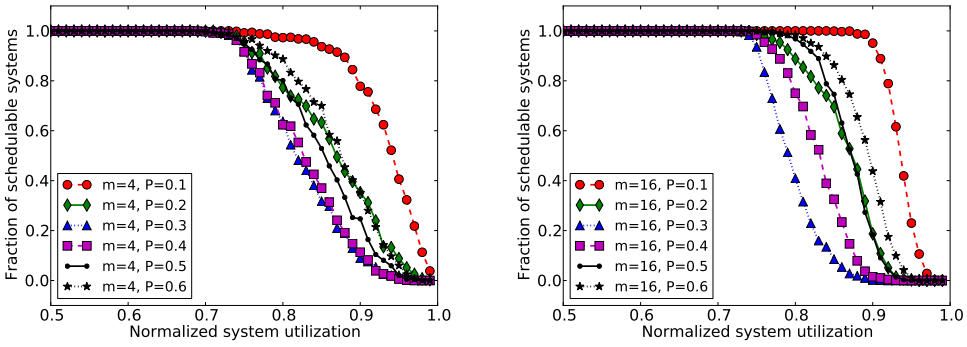
of schedulable task systems increases. This can be observed in the graphs in Figure 7. Theorem 2 can be used to explain this observation. In Theorem 2 we notice that as $U_{HI}^{LO}(\tau)$ decreases, the equation is satisfied for larger values of $U_{LO}^{LO}(\tau)$; i.e., as the sum of the LO-criticality utilizations of the HI-criticality tasks assigned to a processor decreases, there is more capacity available for the LO-criticality tasks. Larger values of Z_U lead to smaller $U_{HI}^{LO}(\tau(\pi_k))$ on each processor. From Theorem 2 we know that in this case there is more capacity available for the LO-criticality tasks. Thus in balanced task systems (where U_{HI} and U_{LO} are equal), if Z_U is larger, it is more likely that we can accommodate LO-criticality tasks on the processors, which leads to a successful partition.

2. Partitioning is more successful when the ratio P favors the task system to have more of the same criticality of tasks, i.e. when HI-criticality utilization U_{HI}, is much greater than LO-criticality utilization U_{LO}, and vice-versa. For example, in the graphs in Figure 8, $P = 0.1$ and $P = 0.6$ results in a larger fraction of task systems being scheduled. On any given processor, more capacity is available to accommodate HI-criticality tasks if the LO-criticality utilization assigned to the processor is small. Thus, if U_{HI} is much greater than U_{LO}, the processors could predominantly have higher HI-criticality utilization and the partitioning is more likely to be successful for smaller LO-criticality utilization. We can present a similar argument if U_{LO} is much greater than U_{HI}.

3. We did not observe any significant difference in schedulability by varying the values of the utilization ranges U_U and U_L used to generate the tasks.

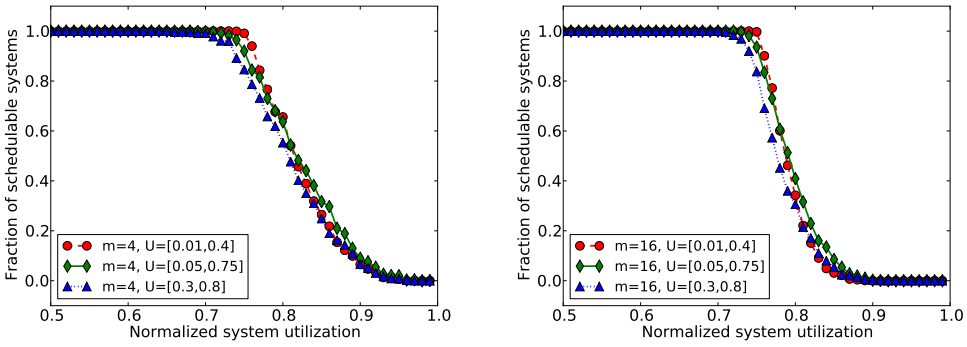
8 Conclusion

We have described and evaluated algorithms for partitioned and global scheduling of mixed-criticality implicit-deadline sporadic task systems upon identical multiproces-



(a) (b)

Figure 8: MC-PARTITION-UT-INC, $U_L = 0.05, U_U = 0.75, Z_L = 1, Z_U = 8$



(a) (b)

Figure 9: MC-PARTITION-UT-INC, $Z_L = 1, Z_U = 8, P = 0.3$

sor systems. Given the rapidly increasing interest in mixed-criticality systems in both industry and academia, and the fact that mixed-criticality (and other) embedded systems are increasingly coming to be implemented upon multiprocessor platforms, this research seems particularly timely and relevant.

Upon multiprocessor platforms, current engineering practice favors the use of partitioned rather than global scheduling. Several researchers have recently argued in favor of global scheduling (or *clustered* scheduling, in which the processors are partitioned into clusters each consisting of a few processors, tasks are assigned to clusters, and scheduled globally within the clusters). We see much merit to these arguments in general; however with regards to mixed-criticality scheduling (at least within the framework we have considered here) we believe that the evidence argues in favor of the partitioned approach. We now summarize the main points that favor this perspective.

As stated earlier in Section 6.1, the partitioning algorithm is better, from the perspective of ***speedup bounds***, when compared to the global algorithm. However, we also noted that the ***speedup bound*** of the global scheduling algorithm may be improved and we leave that as future work.

The ***simulation-based evidence*** is more conclusively in favor of partitioned scheduling: as depicted in Figure 5, even the most naïve partitioning algorithm (WORST-CASE PARTITION, labeled as WC-P in the graphs) significantly out-performs the best global scheduling algorithm (labeled GLOBAL).

We point out that these simulation-based conclusions are consistent with prior findings [1, 2, 10, 11] comparing the partitioned and global scheduling of regular (non-MC) task systems. One of the reasons for the better performance of partitioned algorithms is that properties, often pessimistic, that merely characterize the behavior of a partitioning algorithm may constitute the actual schedulability tests for global scheduling. That is, whereas we can actually run a partitioning algorithm to determine whether a system is successfully partitioned or not, the test for global schedulability is often a pessimistic utilization (or similar) bound: only task systems with utilization not exceeding this bound can be guaranteed schedulable. Hence a simulation-based comparative evaluation would necessarily find all systems with utilization above the bound unschedulable using global scheduling, whereas some of these systems may be successfully scheduled by the partitioned algorithm.

There is also the issue of the comparative ***implementation overhead*** of global versus partitioned scheduling. From this perspective, too, partitioned scheduling appears to win out; as Baker [2] has observed, there is a school of thought that strongly believes that *the overhead of synchronizing schedulers between processors, and lost performance due to translation look-aside buffer and memory cache misses following the migration of a task between processors, will inevitably negate any possible improvement in scheduling efficiency.*

In conclusion, we believe that from the perspective of speedup bound it is not conclusive whether the partitioned mixed-criticality scheduling approach is better than the global one because further work may reveal a better speedup bound for the global mixed-criticality scheduling approach. However, from the simulation-based results we obtained, and from the perspective of implementation overhead it is clear that the partitioned scheduling approach outperforms the global one.

References

[1] T. Baker. Comparison of empirical success rates of global vs. partitioned fixed-priority and EDF scheduling for hard real time. Technical Report TR-050601, Department of Computer Science, Florida State University, 2005.

[2] T. Baker. A comparison of global and partitioned EDF schedulability tests for multiprocessors. In *Proceeding of the International Conference on Real-Time and Network Systems*, Poitiers, France, 2006.

[3] 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.

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

[5] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, S. van der Ster, and L. Stougie. The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems. In *Proceedings of the 2012 24th Euromicro Conference on Real-Time Systems*, ECRTS '12, Pisa (Italy), 2012. IEEE Computer Society.

[6] S. 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] 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.

[8] 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.

[9] 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.

[10] M. Bertogna. *Real-Time Scheduling Analysis for Multiprocessor Platforms*. PhD thesis, Scuola Superiore Santa Anna, Pisa, Italy, 2008.

[11] M. Bertogna. Evaluation of existing schedulability tests for global EDF. In *ICPPW '09: Proceedings of the 2009 International Conference on Parallel Processing Workshops*, pages 11–18, Washington, DC, USA, 2009. IEEE Computer Society.

[12] C. Chekuri and S. Khanna. On multi-dimensional packing problems. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 185–194, January 1999.

[13] C. Chekuri and S. Khanna. On multidimensional packing problems. *SIAM J. Comput.*, 33(4):837–851, 2004.

[14] D. de Niz, K. Lakshmanan, and R. R. Rajkumar. On the scheduling of mixed-criticality real-time task sets. In *Proceedings of the Real-Time Systems Symposium*, pages 291–300, Washington, DC, 2009. IEEE Computer Society Press.

[15] M. Dertouzos. Control robotics : the procedural control of physical processors. In *Proceedings of the IFIP Congress*, pages 807–813, 1974.

[16] F. Dorin, P. Richard, M. Richard, and J. Goossens. Schedulability and sensitivity analysis of multiple criticality tasks with fixed-priorities. *Real-Time Systems*, 2010.

[17] P. Ekberg and W. Yi. Bounding and shaping the demand of mixed-criticality sporadic tasks. In *Proceedings of the 2012 24th Euromicro Conference on Real-Time Systems*, ECRTS '12, Pisa (Italy), 2012. IEEE Computer Society.

[18] 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.

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

[20] J. Herman, C. Kenna, M. Mollison, J. Anderson, and D. Johnson. Rtos support for multicore mixed-criticality systems. In *Proceedings of the 2012 IEEE Real-Time and Embedded Technology and Applications Symposium*, RTAS '12, Beijing (China), 2012. IEEE Computer Society.

[21] H.-M. Huang, C. Gill, and C. Lu. Implementation and evaluation of mixed-criticality scheduling algorithms for periodic tasks. In *Proceedings of the 2012 IEEE Real-Time and Embedded Technology and Applications Symposium*, RTAS '12, Beijing (China), 2012. IEEE Computer Society.

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

[23] K. Lakshmanan, D. de Niz, and R. R. Rajkumar. Mixed-criticality task synchronization in zero-slack scheduling. In *Proceedings of the Real-Time and Embedded Technology and Applications Symposium*, Chicago, 2011. IEEE Computer Society Press.

[24] K. Lakshmanan, D. de Niz, R. R. Rajkumar, and G. Moreno. Resource allocation in distributed mixed-criticality cyber-physical systems. In *Proceedings of the 30th International Conference of Distributed Computing Systems*. IEEE Computer Society Press, 2010.

[25] H. Li and S. Baruah. Global mixed-criticality scheduling on multiprocessors. In *Proceedings of the 2012 24th Euromicro Conference on Real-Time Systems*, ECRTS '12, Pisa (Italy), 2012. IEEE Computer Society.

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

[27] 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.

[28] T. Park and S. Kim. Dynamic scheduling algorithm and its schedulability analysis for certifiable dual-criticality systems. In *Proceedings of the 11th International Conference on Embedded Software*, EMSOFT-2011, pages 253–262, 2011.

[29] R. Pathan. Schedulability analysis of mixed-criticality systems on multiprocessors. In *Proceedings of the 2012 24th Euromicro Conference on Real-Time Systems*, ECRTS '12, Pisa (Italy), 2012. IEEE Computer Society.

[30] F. Santy, L. George, P. Thierry, and J. Goossens. Relaxing mixed-criticality scheduling strictness for task sets scheduled with FP. In *Proceedings of the 2012 24th Euromicro Conference on Real-Time Systems*, ECRTS '12, Pisa (Italy), 2012. IEEE Computer Society.

[31] D. Tamas-Selicean and P. Pop. Design optimization of mixed-criticality real-time applications on cost-constrained partitioned architectures. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, Vienna, Austria, 2011. IEEE Computer Society Press.

[32] 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.

[33] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha. Memory access control in multiprocessor for real-time systems with mixed criticality. In *Proceedings of the 2012 24th Euromicro Conference on Real-Time Systems*, ECRTS '12, Pisa (Italy), 2012. IEEE Computer Society.