Scheduling mixed-criticality implicit-deadline sporadic task systems upon a varying-speed processor

Sanjoy Baruah Zhishan Guo
The University of North Carolina at Chapel Hill

***Abstract*—A *mixed criticality (MC)* workload consists of components of varying degrees of importance (or "criticalites"). The problem of executing a MC workload, modeled as a collection of independent implicit-deadline sporadic tasks executing upon a preemptive uniprocessor, is considered. Suitable scheduling strategies are devised for scheduling such systems despite uncertainty and unpredictability in both the amount of execution needed by the tasks, and the effective speed of the processor. These scheduling strategies allow for simultaneously making efficient use of platform resources and ensuring the correctness of the more critical workload components at greater levels of assurance.**

I. Introduction

Special-purpose processors used in implementing safety-critical systems are designed to be highly predictable, in order that it be possible to provide tight bounds on the worst-case run-time system behavior of the system to a very high level of assurance, during system design time itself. Such design-time predictability is essential for safety-critical functionalities, but is difficult to achieve with Commercial Off-The-Shelf (COTS) processors that are typically engineered to provide good average-case performance rather than worst-case guarantees and consequently exhibit considerable (unpredictable) variation between their average-case and their worst-case behaviors. To provide worst-case guarantees upon such platforms requires worst-case resource provisioning, which can lead to significant resource under-utilization during run-time. One approach to overcome such resource under-utilization is to implement the safety-criticality functionalities alongside some less critical functionalities, as part of a *mixed-criticality (MC)* workload, upon a shared platform. Such an approach recognizes that since safety-critical functionalities must have their correctness demonstrated to very high levels of assurance, resources need to be provisioned to the critical functionalities under very conservative assumptions, which will lead to over-provisioning. These over-provisioned resources are unlikely to be actually needed during run-time, but can instead be used to execute the less-critical functionalities.

This general idea has been very widely explored with respect to dealing with variations in execution time of pieces of code; in this paper, we additionally explore this idea with respect to variations in the *execution speed* of processors.

WCET in mixed-criticality systems. The worst-case execution time (WCET) of a given piece of code upon a specified platform denotes an upper bound of the duration of time needed for it to execute. Determining the exact WCET of an arbitrary piece of code is provably an undecidable problem. Even when severe restrictions are placed upon the structure of the code (e.g., loop bounds must be known at compile time), sophisticated features found upon COTS processors (such as multi-level cache, deep pipelining, speculative out-of-order execution, etc.) are hard to analyze and make it extremely difficult to determine WCET precisely. A large body of prior research on mixed criticality scheduling (see [9] for a review of some of this work) has focused upon dealing with the reality that different tools for determining WCET bounds may be more or less conservative than each other: a more conservative tool determines an upper bound on the actual WCET of an input piece of code at a higher level of assurance than a less conservative tool. The upper bound determined by the more conservative tool is larger – sometimes by several orders of magnitude – than the one determined by the less conservative tool. Although it may be necessary (for instance, mandated by a statutory Certification Authority) to use a very conservative tool for validating the correctness of safety-critical functionalities, less conservative tools should suffice for validating the correctness of less critical functionalities. This is modeled by assigning each job two WCET parameters – a larger, more conservative one, and a smaller, less conservative one. Some of the jobs are designated as being safety-critical; the remaining ones are not safety-critical. The objective is to determine a run-time scheduling strategy to ensure that (i) all jobs complete by their deadlines if each job completes execution upon having executed for no more than the smaller of its WCET values; and (ii) the jobs designated as being safety-critical continue to complete by their deadlines (although the non-critical jobs may fail to do so) if some job does not complete execution upon having executed for up to the smaller of its WCET values, although each job does complete upon having executed for the larger of its WCET values.

Varying-speed processors: context and motivation. The take-away message from the discussion above may be summarized as follows. In earlier times, safety-critical systems were highly predictable, and WCETs could therefore be determined very accurately; further, the WCETs were very close to the actual execution times. Over time, increasing

1052-8725/14 $31.00 © 2014 IEEE
DOI 10.1109/RTSS.2014.15

unpredictability/ non-determinism in both code and platforms made it necessary to *estimate* WCETs, rather than determine them precisely. These estimates could be more or less conservative: while we may have a greater level of assurance that a more conservative estimate is correct, it requires that more resources be provisioned.

This narrative is now being repeated with respect to processor speeds. CPU clock rates were derived entirely from crystal-based oscillators and were therefore extremely predictable and deterministic. However, the ongoing quest for ever more powerful and energy-efficient processors is yielding innovations that result in clocks with varying speed during run-time. For example [8] describes a technique for detecting whether signals are late at the circuit level within a CPU micro-architecture, and if so to recover by delaying the next clock tick so that logical faults do not propagate to higher (i.e., the software) levels. Such run-time variation in processor speed is further exacerbated with the increasing trend towards *Globally Asynchronous Locally Synchronous*, or GALS, circuit designs. In a GALS circuit, different parts of the circuit are clocked separately using separate local clocks, and signals propagate between the different synchronous modules in an asynchronous manner. The overall execution rate of the circuit is thus determined by the (highly variable) delay of such asynchronous propagation. Asynchronous processors [12] represent the logical extreme of the GALS trend in that the pace of computation is governed entirely by the time taken for signals to propagate. Such asynchronous processors are known to be extremely energy-efficient and fast, but highly non-deterministic. As GALS and asynchronous processors become the most advanced COTS processors, it is only to be expected that they will increasingly come to be used in safety-critical embedded systems [16]; when this happens, we must be able to deal with run-time variations in the speeds of these processors that are of a magnitude comparable to the variations we see in the execution times of pieces of code on today's advanced COTS processors.

The research reported in this paper seeks to address this uncertainty and run-time variation in processor speeds within a framework similar to the one that was previously established for dealing with uncertainty in WCETs. The general idea is as follows. In order to be able to guarantee that deadlines are met under all run-time conditions, conservative analysis prior to run-time must make the most pessimistic assumptions regarding clock speed: that during run-time *the clock speed takes on the lowest possible value*. If this lowest possible value is highly unlikely to be reached in practice, then a significant under-utilization of the CPU's computing capacity will be observed during run-time. A less conservative analysis, on the other hand, may assume a less pessimistic (i.e., larger) lower bound on the clock speed during run-time. Using such an assumption as the basis for making resource-allocation decisions will likely lead to more efficient usage of the CPU's computing capacity during run-time; however, there is a possibility that the actual processor speed will fall below the lower-bound estimate used in the analysis (thereby invalidating the conclusions drawn during such analysis).

Related work. A large body of research on MC scheduling has been conducted over the past 5-7 years — see [9] for an excellent survey. Much of this work has focused on scheduling MC systems in which multiple WCETs are specified for each job or task; some (e.g., [2], [10], [6]) has considered uncertainty in specifying minimum inter-arrival durations for sporadic tasks. A recent paper [7] considered MC systems that are specified as *finite collections of independent jobs*, each characterized by a single WCET, that are implemented upon varying-speed preemptive uniprocessors. We are currently working on extending the results of [7] to model the scheduling of finite collections of independent jobs with each characterized by multiple WCETs.

This research. As discussed above, much prior work on MC scheduling has separately considered uncertainty in (i) estimating upper bounds on WCETs, and (ii) estimating lower bounds on processor speed during run-time. In this paper, we seek to integrate both these dimensions of uncertainty within a single integrated framework, for MC systems comprised of recurrent tasks. (Some related work, [7] in particular, seeks similar unification but for the far simpler workload model of collections of independent jobs. The techniques that need to be developed, and the results obtained, are strikingly different; the unified model thus exhibits the same characteristics as the model of just multiple WCETs, where, for example, the structure and properties of the MCEDF [20] algorithm for scheduling collections of independent jobs proved very different from those of the algorithm in [14] for scheduling sporadic task systems.)

Organization. In Section II we propose a formal model for representing MC systems comprised of recurrent tasks, each of which is (i) designated as being of either high or low criticality[1], and (ii) characterized by two WCET parameters, that are to execute upon a single preemptive processor whose speed may vary during run-time. Different scheduling strategies are possible depending upon whether the varying-speed processor has a means or not of monitoring during run-time what its actual speed is. In Sections III–IV, we consider processors that do not possess such self-monitoring capabilities; in Section V, we turn our attention to processors that do. For all our algorithms we provide sufficient schedulability conditions; for some, we also determine speedup bounds. In addition to such theoretical analyses, we

[1]In common with much earlier work in MC scheduling we will, for the most part, restrict our attention here to such *dual*-criticality systems. In Section VII, we briefly discuss extending our work to systems that have more than two criticality levels defined.

have conducted schedulability experiments upon randomly-generated workloads evaluating our different algorithms — these experiments are reported in Section VI. We conclude in Section VII by discussing extensions that we are currently working upon, and by placing this work within a larger context of mixed-criticality scheduling theory.

II. Model and definitions

A mixed-criticality (MC) implicit-deadline sporadic task system τ is specified as a finite collection of MC implicit-deadline sporadic tasks, each of which may generate an unbounded number of MC jobs.

MC jobs. We will, for the most part, restrict our attention to *dual*-criticality systems: systems with two distinct criticality levels, which we denote as LO and HI. Each such dual-criticality 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, and $d_i \in R^+$ the deadline. We require that $d_i \geq a_i$.
- $\chi_i \in \{\text{LO}, \text{HI}\}$ denotes the criticality of the job. A HI-criticality job is one that is subject to a higher level of validation than a LO-criticality job.
- $c_i(\text{HI})$ and $c_i(\text{LO})$ specify estimates on the WCET of J_i, made at greater and lesser levels of assurance. We assume $c_i(\text{LO}) \leq c_i(\text{HI})$.

System *behavior*. 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 some amount of execution γ_i. 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. These values of γ_i for a given execution of the system defines the kind of *behavior* exhibited by the system during that execution. If each J_i signals completion without exceeding $c_i(\text{LO})$ units of execution, the system exhibits LO-*criticality behavior*; if some job J_i does not signal completion after executing for more than $c_i(\text{LO})$ (but proceeds for no more than $c_i(\text{HI})$) units of execution, the system exhibits HI-*criticality behavior*. If any J_i does not signal completion despite having executed for $c_i(\text{HI})$ units, the system exhibits *erroneous behavior*.

MC implicit-deadline sporadic tasks. Analogously to traditional (non-MC) implicit-deadline sporadic tasks, an MC implicit-deadline sporadic task τ_k is characterized by a four-tuple $(\chi_k, C_k(\text{LO}), C_k(\text{HI}), T_k)$, with the following interpretation. Task τ_k generates an unbounded sequence of jobs, with successive jobs being released $\geq T_k$ time apart. Each job has a deadline T_k time units after its release. The criticality of each 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.

Characterizing a varying-speed processor. A varying-speed processor is characterized by a *normal speed* σ and a *degradation ratio* ρ ($\rho \leq 1$). The processor is said to exhibit *normal behavior* provided its speed never falls below σ; if its speed falls below σ but remains above $\rho \times \sigma$ at all times, it exhibits *degraded behavior*. It is said to exhibit *erroneous behavior* if its speed ever falls below $\rho \times \sigma$.

A mixed-criticality instance $\mathcal{I}$ is specified as a finite collection of MC tasks τ and a varying-speed processor characterized by the two parameters σ and ρ.

Correctness criterion. We define an algorithm for scheduling MC instances to be *correct* if it is able to schedule any system such that
– During all LO-criticality behaviors of the system in which the processor speed remains at or above σ, all jobs receive enough execution between their release time and deadline to signal completion; and
– During all HI-criticality behaviors of the system, all HI-criticality jobs receive enough execution between their release time and deadline to signal completion provided the processor speed remains at or above $\rho \times \sigma$.

That is, if the system exhibits LO-criticality behavior and the processor exhibits normal behavior, then all deadlines should be met; else, all HI-deadlines should be met (provided neither the system nor the processor exhibits erroneous behavior).

Note that if any job executes for more than its LO-criticality WCET or the processor speed falls below σ, we do not require any LO-criticality jobs (including those that may have arrived before this happened) to complete by their deadlines. This is a consequence of the nature of system validation: informally speaking, the system designer fully expects that the system will exhibit LO-criticality behavior and the processor always execute at or above its normal speed, and hence is only concerned that they behave as desired under these circumstances. The validation process for the more critical functionalities, on the other hand, allows for the possibility that some jobs may exhibit HI-criticality behavior and/ or the processor executes at a speed slower than σ (but $\geq \rho\sigma$), and requires that all HI-criticality jobs nevertheless meet their deadlines; however, such validation is not concerned with the fate of the LO-criticality jobs.

Utilization parameters. The *utilization* of a (regular, i.e., non-MC) implicit-deadline sporadic task system denotes the sum of the ratios of the WCETs to periods of all the tasks in the system. We may define analogous concepts for mixed-criticality sporadic task systems. Let τ 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:

$$U_x^y(\tau) = \sum_{\tau_i \in \tau \wedge \chi_i = x} \frac{C_i(y)}{T_i}. \quad (1)$$

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.

A *clairvoyant scheduling algorithm* is one that knows, prior to scheduling an instance, (i) precisely how much execution each job in the instance will require in order to complete, and (ii) the precise manner in which the processor speed will vary during run-time.

*Definition 1 (**optimal scheduling strategy**):* An optimal scheduling strategy for MC instances possesses the property that if it fails to maintain correctness for a given MC instance $\mathcal{I}$, then no non-clairvoyant algorithm can ensure correctness for the instance $\mathcal{I}$.

Run-time support. As stated above, we assume that the processor speed may vary in an *a priori* unknown manner during run-time. A ***self-monitoring*** processor is one that knows its speed at each instant in time. Such a self-monitoring capability is not always available on asynchronous processors. We therefore start out in Sections III-IV assuming that self-monitoring is not available; later in Section V, we consider systems that possess such self-monitoring capability. In both cases, we do however assume the existence of accurate clocks that can measure the *amount* of time that has elapsed. That is, we assume that we can determine how long – i.e., for what duration – a job has executed, even though we may not know how much execution it has accomplished in this duration.

III. VDF-NM: Non self-monitoring processors

In this and the next section, we consider the scheduling of MC implicit-deadline sporadic task systems upon processors that do not possess the capability of self-monitoring. We define an algorithm, VDF-NM (for **V**irtual-**D**eadline **F**irst – **N**on-**M**onitoring) for scheduling such systems. VDF-NM is motivated by, and hence quite similar to, the EDF-VD algorithm that was proposed in [3], [4].

We start out with an overview describing the preprocessing that is done by VDF-NM, and the manner in which it makes run-time scheduling decisions. This is followed by a proof of correctness, and some theorems characterizing its performance. In Section III-A, we derive a quantitative bound on VDF-NM's worst-case performance via the *speedup* metric, which is widely used in characterizing the behavior of mixed-criticality scheduling algorithms.

Overview. Let τ denote the MC implicit-deadline sporadic task system that is to be scheduled on a preemptive processor with normal speed σ that is, without loss of generality, assumed to be equal to one (i.e., $\sigma \leftarrow 1$), and degradation ratio ρ. Prior to run-time, VDF-NM performs a schedulability test to determine whether τ can be successfully scheduled by it 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 $\tau_i \in \tau$. The algorithm for computing these parameters is described

---

$\tau = \cup_{i=1}^{n} \{\tau_i\}$ to be scheduled on a varying-speed processor with normal speed $\sigma = 1$ and degradation ratio ρ

1) Compute x as follows: $x \leftarrow \frac{U_{\text{HI}}^{\text{LO}}(\tau)}{1-U_{\text{LO}}^{\text{LO}}(\tau)}$
2) If $\left(U_{\text{HI}}^{\text{HI}}(\tau)/(1-x) \leq \rho\right)$ **then**
   $\hat{T}_i \leftarrow x\,T_i$ for each HI-criticality task τ_i
   declare success and **return**

else declare failure and **return**

Figure 1. VDF-NM: The preprocessing phase.

in pseudo-code form in Figure 1; this pseudo-code is proved correct in Thms 1–2. Run-time scheduling is done according to the Algorithm EDF, with *virtual deadlines*: deadlines that VDF-NM computes (in a manner to be described below) and assigns to jobs before handing them off to the EDF scheduler. The EDF scheduler will then use these virtual deadlines for the purpose of determining scheduling priority.

Suppose that a job of task τ_i arrives at time-instant t_a. If $\chi_i = \text{LO}$, then this job is assigned a virtual deadline equal to $t_a + T_i$ whereas if $\chi_i = \text{HI}$, it is assigned a virtual deadline equal to $t_a + \hat{T}_i$. If some job executes for a duration exceeding its LO-criticality WCET without signaling that it has completed execution, we know that either the system is no longer exhibiting LO-criticality behavior, or the processor is no longer exhibiting normal behavior (or both). In response, the run-time scheduler immediately discards all LO-criticality jobs[2]; subsequently, only HI-criticality jobs will receive any execution. Subsequent execution of HI-criticality tasks (including the jobs that are currently active) continue to be done according to EDF. But the *actual* job deadlines (arrival time plus period) are used.

Detailed description. As shown in Figure 1, VDF-NM first computes a parameter x (the reason why x is assigned this value is derived below – see Expression 4) and then assigns values to the $\hat{T}_i$ parameters for all HI-criticality tasks as follows:

$$\hat{T}_i \leftarrow x \times T_i. \tag{2}$$

Theorem 1: The following condition is sufficient for ensuring that VDF-NM successfully schedules all LO-criticality behaviors of τ:

$$x \geq \frac{U_{\text{HI}}^{\text{LO}}(\tau)}{1 - U_{\text{LO}}^{\text{LO}}(\tau)}. \tag{3}$$

Proof: If EDF is able to schedule, upon a unit-speed (since $\sigma \equiv 1$) processor, all LO-criticality behaviors of the task system obtained from τ by replacing each HI-criticality task τ_i by one with a reduced period, then it follows from the *sustainability* [5] of uniprocessor EDF that EDF is able to schedule all LO-criticality behaviors of τ upon a unit-speed

[2]An efficient implementation of such a run-time dispatcher may be obtained using the technique described in [3, Sec. V-A], to have run-time that is logarithmic in the number of tasks.

processor as well. Note that scaling down the period of each HI-criticality task by a factor x is equivalent to inflating its utilization by a factor $1/x$. Since the utilization bound of EDF for implicit-deadline tasks is known to be equal to the processor capacity [17], we therefore conclude that

$$\Big(U_{\text{LO}}^{\text{LO}}(\tau) + \frac{U_{\text{HI}}^{\text{LO}}(\tau)}{x} \leq 1\Big) \Leftrightarrow \Big(x \geq \frac{U_{\text{HI}}^{\text{LO}}(\tau)}{1 - U_{\text{LO}}^{\text{LO}}(\tau)}\Big)$$

is sufficient for ensuring that VDF-NM successfully schedules all LO-criticality behaviors of τ. ■

Algorithm VDF-NM thus chooses for x the smallest value such that Theorem 1 is satisfied:

$$x \leftarrow \frac{U_{\text{HI}}^{\text{LO}}(\tau)}{1 - U_{\text{LO}}^{\text{LO}}(\tau)}. \tag{4}$$

With this value of x, we now derive a sufficient condition for ensuring that VDF-NM meets all HI-criticality deadlines during all HI-criticality behaviors of τ:

Theorem 2: The following condition is sufficient for ensuring that VDF-NM successfully schedules all HI-criticality behaviors of τ:

$$U_{\text{HI}}^{\text{HI}}(\tau)/(1-x) \quad \leq \quad \rho. \tag{5}$$

Proof: Suppose that at some instant t^* during run-time, the scheduler detects that some job has executed for a duration exceeding its LO-criticality WCET without signaling completion. It immediately discards all LO-criticality jobs, re-assigns each active HI-criticality job a deadline equal to its release time plus the original period of the task that generated it, and assigns each future-arriving HI-criticality job a deadline equal to its release time plus the period of the task that generates it.

Since the modified relative deadline of a job of HI-criticality task τ_i is equal to xT_i (see Fig. 2), if this job is active at time-instant t^* its actual deadline must be at least $(T_i - xT_i)$ or $(1-x)T_i$ time units in the future. The utilization of task τ_i beyond time-instant t^* is therefore no greater than that of an implicit-deadline sporadic task with execution requirement $c_i(\text{HI})$ and period $(1-x)T_i$. Summing over all HI-criticality tasks and using once again the fact that EDF has a utilization bound equal to the processor capacity, we conclude that

$$\sum_{\chi_i=\text{HI}} \frac{c_i(\text{HI})}{(1-x)T_i} \;\leq\; \rho \;\Leftrightarrow\; \frac{1}{1-x}\sum_{\chi_i=\text{HI}} \frac{c_i(\text{HI})}{T_i} \;\leq\; \rho$$
$$\Leftrightarrow \qquad U_{\text{HI}}^{\text{HI}}(\tau)/(1-x) \;\leq\; \rho$$

is a sufficient condition for VDF-NM to meet all HI-criticality job deadlines upon the degraded processor of speed $\geq \rho$. ■

A. A speedup bound

Speedup bounds have been widely used for characterizing the performance of mixed-criticality scheduling algorithms. We now derive such a bound for VDF-NM.

Definition 2 (Speedup bound): An algorithm A is defined to have a speedup bound $\mathbf{b}$, where $\mathbf{b}$ is a positive real number ≥ 1, if any task system τ that can be correctly scheduled by any hypothetical clairvoyant scheduling algorithm upon a processor with normal speed σ and degradation ratio ρ, is correctly scheduled by Algorithm A upon a processor with normal speed $\mathbf{b} \times \sigma$ and (the same) degradation ratio ρ.

It is evident that a smaller speedup bound is better – a speedup bound of 1 means that the algorithm is optimal. We will prove that VDF-NM has a speedup bound no larger than ϕ, where ϕ is the famous mathematical constant $(\sqrt{5}+1)/2 \approx 1.618$ (known as the *Golden Ratio*).

We start out proving a sufficient schedulability condition:

Theorem 3: If τ satisfies

$$U_{\text{LO}}^{\text{LO}}(\tau) + \min\Big(\frac{U_{\text{HI}}^{\text{HI}}(\tau)}{\rho}, \frac{U_{\text{HI}}^{\text{LO}}(\tau)}{1 - \frac{U_{\text{HI}}^{\text{HI}}(\tau)}{\rho}}\Big) \leq 1, \tag{6}$$

then it is successfully scheduled by VDF-NM.

Proof: We consider two cases:

Case A: $U_{\text{LO}}^{\text{LO}}(\tau) + \frac{U_{\text{HI}}^{\text{HI}}(\tau)}{\rho} \leq 1$. In this case, consider the regular (i.e., not mixed-criticality) task system obtained by including all the LO-criticality tasks and, for each HI-criticality task τ_i, a task with WCET equal to $c_i(\text{HI})/\rho$ and period equal to T_i. The utilization of this system is equal to

$$U_{\text{LO}}^{\text{LO}}(\tau) + \sum_{\chi_i=\text{HI}} \frac{c_i(\text{HI})/\rho}{T_i} = U_{\text{LO}}^{\text{LO}}(\tau) + \frac{U_{\text{HI}}^{\text{HI}}(\tau)}{\rho}, \tag{7}$$

which is assumed ≤ 1. It therefore follows that we can EDF-schedule the original MC system with each HI-criticality task's HI-criticality WCET *inflated* by a factor $1/\rho$ to meet all deadlines. Hence, all HI-criticality jobs will continue to meet their deadlines even if the processor speed falls by as much as a factor ρ.

Case B: $U_{\text{LO}}^{\text{LO}}(\tau) + \frac{U_{\text{HI}}^{\text{HI}}(\tau)}{\rho} > 1$. For Inequality (7) to hold, it must then be the case that

$$\begin{aligned}
&U_{\text{LO}}^{\text{LO}}(\tau) + \frac{U_{\text{HI}}^{\text{LO}}(\tau)}{1 - \frac{U_{\text{HI}}^{\text{HI}}(\tau)}{\rho}} \leq 1 \\
&\Leftrightarrow \quad \frac{U_{\text{HI}}^{\text{LO}}(\tau)}{1 - \frac{U_{\text{HI}}^{\text{HI}}(\tau)}{\rho}} \leq 1 - U_{\text{LO}}^{\text{LO}}(\tau) \\
&\Leftrightarrow \quad \frac{U_{\text{HI}}^{\text{LO}}(\tau)}{1 - U_{\text{LO}}^{\text{LO}}(\tau)} \leq 1 - \frac{U_{\text{HI}}^{\text{HI}}(\tau)}{\rho} \\
&\Leftrightarrow \quad x \leq 1 - \frac{U_{\text{HI}}^{\text{HI}}(\tau)}{\rho} \text{ (By Inequality (4))} \\
&\Leftrightarrow \quad U_{\text{HI}}^{\text{HI}}(\tau)/(1-x) < \rho
\end{aligned}$$

and τ is successfully scheduled by Algorithm VDF-NM according to Theorem 2. ■

In Theorem 4 below, we will use Theorem 3 to prove that VDF-NM has a speedup bound no greater than ϕ. But first, we briefly enumerate some properties of ϕ that will be used in this proof. Let Φ denote the multiplicative inverse of ϕ

(i.e., $\Phi \equiv 1/\phi$).The following identities are easily validated algebraically:

1) $\Phi^2 + \Phi = 1$;
2) $\Phi/(1+\Phi) = \Phi^2$.

Theorem 4: Algorithm VDF-NM has a speedup bound that is no larger than ϕ.

Proof: We will show below that any MC task system τ that can be correctly scheduled by an optimal algorithm on a processor with normal speed Φ and degradation ratio ρ, is correctly scheduled by VDF-NM on a processor with normal speed 1 and the same degradation ratio. This establishes the theorem since it shows that a processor that are faster by a factor of $1/\Phi$ (which is ϕ) is sufficient for VDF-NM to correctly schedule τ.

Observe that any τ that is correctly scheduled by a clairvoyant scheduler upon a processor with normal speed Φ and degradation ratio ρ must necessarily satisfy

$$\max\Big(U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{LO}}(\tau), \frac{U_{\text{HI}}^{\text{HI}}(\tau)}{\rho}\Big) \leq \Phi, \tag{8}$$

since its LO-criticality utilization $(U_{\text{LO}}^{\text{LO}}(\tau)+U_{\text{HI}}^{\text{LO}}(\tau))$ must be $\leq \Phi$ and its HI-criticality utilization $U_{\text{HI}}^{\text{HI}}(\tau)$ must be $\leq \Phi\rho$. We will now show that any such task system is correctly scheduled by VDF-NM upon a varying speed processor with normal speed 1 and degradation ratio ρ.

Case A: $U_{\text{HI}}^{\text{LO}}(\tau) \geq \Phi U_{\text{LO}}^{\text{LO}}(\tau)$. From Inequality (8), we have

$$\begin{aligned} & U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{LO}}(\tau) \leq \Phi \\ \Rightarrow\ & U_{\text{LO}}^{\text{LO}}(\tau)(1+\Phi) \leq \Phi \\ \Leftrightarrow\ & U_{\text{LO}}^{\text{LO}}(\tau) \leq \frac{\Phi}{1+\Phi} \\ \Leftrightarrow\ & U_{\text{LO}}^{\text{LO}}(\tau) \leq \Phi^2. \text{ (See listed properties of } \phi \text{ above)} \end{aligned}$$

Consequently,

$$U_{\text{LO}}^{\text{LO}}(\tau) + \frac{U_{\text{HI}}^{\text{HI}}(\tau)}{\rho} \leq \Big(\Phi^2 + \Phi\Big) = 1.$$

And it follows from Theorem 3 that τ is scheduled correctly by VDF-NM.

Case B: $U_{\text{HI}}^{\text{LO}}(\tau) \leq \Phi U_{\text{LO}}^{\text{LO}}(\tau)$. Again Inequality (8) yields

$$\begin{aligned} & U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{LO}}(\tau) \leq \Phi \\ \Rightarrow\ & \frac{1}{\Phi}U_{\text{HI}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{LO}}(\tau) \leq \Phi \\ \Leftrightarrow\ & \frac{\Phi+1}{\Phi}U_{\text{HI}}^{\text{LO}}(\tau) \leq \Phi \\ \Leftrightarrow\ & U_{\text{HI}}^{\text{LO}}(\tau) \leq \Phi\frac{\Phi}{\Phi+1} \ = \ \Phi^3. \end{aligned}$$

The last equality holds since $\Phi/(\Phi+1) = \Phi^2$ – again see listed properties of ϕ.

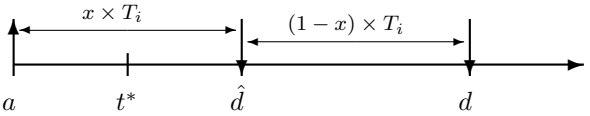

Figure 2. A job of HI-criticality task τ_i that arrives at time a is assigned a virtual deadline of $\hat{d} = a + \hat{T}_i$; its real deadline is at $d = a + T_i$. If a criticality level change is observed at $t^* \leq \hat{d}$, there is at least $(1-x)T_i$ duration until the *real* deadline.

Using the above relationship, we have,

$$\begin{aligned} & U_{\text{LO}}^{\text{LO}}(\tau) + \frac{U_{\text{HI}}^{\text{LO}}(\tau)}{1 - \frac{U_{\text{HI}}^{\text{HI}}(\tau)}{\rho}} \\ = \ & \text{(Using the identity } \tfrac{a}{1-b} \equiv a + a\tfrac{b}{1-b} \text{ on the 2}^{\text{nd}} \text{ term)} \\ & U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{LO}}(\tau)\Big(\frac{U_{\text{HI}}^{\text{HI}}(\tau)/\rho}{1 - \frac{U_{\text{HI}}^{\text{HI}}(\tau)}{\rho}}\Big) \\ \leq \ & \text{(By Inequality (8), } \tfrac{U_{\text{HI}}^{\text{HI}}(\tau)}{\rho} \leq \Phi\text{)} \\ & U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{LO}}(\tau)\Big(\frac{\Phi}{1-\Phi}\Big) \\ \leq \ & \Big(U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{LO}}(\tau)\Big) + \Phi^3\Big(\frac{\Phi}{1-\Phi}\Big) \\ \leq \ & \Phi + \Phi^3\Big(\frac{\Phi}{1-\Phi}\Big) \ = \ 1. \end{aligned}$$

And it follows from Theorem 3 that τ is successfully scheduled by VDF-NM. ■

IV. VDF-NM+: A PRAGMATIC IMPROVEMENT

The top-level idea behind Algorithm VDF-NM is essentially this: determine the smallest scaling factor $x < 1$ such that the system with HI-criticality deadlines scaled by a factor x remains EDF-schedulable in LO-criticality behaviors, and then determine whether shrinking HI-criticality deadlines in this manner will allow all HI-criticality deadlines to be guaranteed meet in the event of a HI-criticality behavior being identified (see Figure 2 above). Both the LO-criticality and the HI-criticality schedulability testing is done via the utilization-based EDF schedulability test. For the LO-criticality schedulability testing, each HI-criticality task τ_i is modeled as a task with WCET $c_i(\text{LO})$ and period xT_i; for HI-criticality schedulability testing, it is modeled as a task with WCET $c_i(\text{HI})$ and period $(1-x)T_i$.

Although this approach is correct (as was shown in Section III above) it can be *pessimistic*. This fact was earlier observed, for the case of multiple WCETs but a single processor speed, in [14], which proposed an approach that required the determining of a separate scaling factor for each HI-criticality task. Since this can prove very computation-intensive, it was recently suggested [19] that the HI-criticality tasks be partitioned into two categories and a different scaling factor be applied to each.

The algorithm we advocate in this section, Algorithm VDF-NM+, takes a different approach to reducing pessimism: for the purposes of doing the schedulability anal-

yses, model the HI-criticality tasks as *constrained-deadline* (rather than implicit-deadline) tasks [18]. More specifically, for each HI-criticality task τ_i,

- For LO-criticality schedulability analysis, model it as a constrained-deadline task with WCET $c_i(\text{LO})$, relative deadline xT_i, and period T_i;
- For HI-criticality schedulability analysis, model it as a constrained-deadline task with the parameters[3] WCET $c_i(\text{LO})$, relative deadline $(1-x)T_i$, and period T_i.

Although EDF-schedulability analysis of constrained-deadline sporadic task systems is NP-hard [13], polynomial-time approximation schemes (PTAS's) are known (see, e.g., [1]) that can solve this problem in efficient polynomial time to any desired degree of accuracy. We have therefore implemented the following method for computing the scaling factor x that is used by VDF-NM:

1) Use binary search over the range $(0, 1)$ to determine, to any desired degree of accuracy, the smallest value of x for which the constrained-deadline task system
$$\cup_{\chi_i=\text{LO}}\{(c_i(\text{LO}), T_i, T_i)\} \bigcup \cup_{\chi_i=\text{HI}}\{c_i(\text{LO}), xT_i, T_i\}$$
is EDF-schedulable.
2) For the value of x determined above, check whether the constrained-deadline task system
$$\cup_{\chi_i=\text{HI}}\{c_i(\text{HI}), (1-x)T_i, T_i\}$$
is EDF-schedulable. If so, use this value of x as the scaling factor in Step 2 of Fig. 1; else, declare failure.

This is clearly a strict improvement over the method for computing the scaling factor used in Step 1 of Fig. 1, in the sense that the value of x computed can only be smaller, and hence failure will be declared for fewer systems. (Since the binary search procedure may terminate without checking the value of x computed in Equation 4, we should additionally test the value of x computed in Equation 4 to ensure strict dominance over the approach of Figure 1.) The speedup bound of ϕ therefore continues to hold for the improved algorithm as well. We have conducted extensive schedulability experiments on randomly-generated workloads to explore the amount of improvement achieved by this pragmatic improvement; the outcomes of these experiments is reported in Section VI. Across all the experiments that we conducted, it appears that this simple pragmatic improvement to VDF-NM's schedulability testing provides between one-half to two-thirds the benefits, in terms of enhanced schedulability, that is obtained by implementing the MC system upon more powerful self-monitoring processors (as discussed in Section V below).

[3]Although it may not be immediately obvious that this is an accurate modeling of the worst-case workload of the HI-criticality task upon the identification of HI-criticality behavior, tabulating the *processor demand* [11] of the HI-criticality task, and of the task modeled using these parameters, over different interval-lengths should demonstrate their equivalence.

V. VDF-WM: Self-monitoring processors

If the processor upon which an MC implicit-deadline sporadic task system is being implemented knows the rate at which it is executing during each instant of run-time, it is possible to design a scheduling algorithm to exploit such knowledge. We now define such an algorithm, VDF-WM (for **V**irtual-**D**eadline **F**irst – **W**ith **M**onitoring). In terms of run-time behavior, VDF-WM differs from the algorithm VDF-NM described in Section III above only in that while the trigger for VDF-NM to drop LO-criticality jobs and revert to original deadlines was that some job executes for a duration exceeding its LO-criticality WCET without signaling that it has completed execution, the trigger for VDF-WM is that some job executes for a duration exceeding its LO-criticality WCET without signaling that it has completed execution (as with VDF-NM) *or the processor speed is observed to fall below its normal value of* σ.

The pre runtime processing phase for VDF-WM is very similar to VDF-NM (Figure 1). In particular, Step 1 is identical — the same scaling factor $x = U_{\text{HI}}^{\text{LO}}(\tau)/(1 - U_{\text{LO}}^{\text{LO}}(\tau))$ is computed. However, the acceptance test – Step 2 of the pseudo-code – is different: VDF-WM checks to determine whether the value of x computed in Step 1 satisfies

$$x\, U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{HI}}(\tau) \leq \rho. \tag{9}$$

Since the scaling factor x used by VDF-WM is the same as the one used by VDF-NM, Theorem 1 continues to hold and VDF-WM is therefore seen to schedule all LO-criticality behaviors correctly. In Theorem 5 below, we prove that all HI-criticality behaviors are also scheduled correctly:

Theorem 5: The following condition is sufficient for ensuring that VDF-WM successfully schedules all HI-criticality behaviors of τ:

$$xU_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{HI}}(\tau) \leq \rho. \tag{10}$$

Proof: Suppose that VDF-WM cannot meet all deadlines in all HI-criticality behaviors of τ. Let I denote a minimal instance of jobs released by τ, on which a deadline is missed. Without loss of generality, assume that the earliest job-release in I occurs at time zero, and let t_f denote the instant of the (first) deadline miss — since (as argued above) Theorem 1 holds for VDF-WM this must be the deadline of a HI-criticality job, in a HI-criticality behavior. Let t^* denote the time-instant at which HI-criticality behavior is first flagged (i.e., the first instant at which some job executes for more than its LO-criticality worst-case execution time without signaling that it has completed execution).

We now introduce some notation for the remainder of this proof:

1) For each i, $1 \leq i \leq n$, let η_i denote the amount of execution over the interval $[0, t_f]$ that is needed by jobs in I that are generated by task τ_i.

2) For each i, $1 \leq i \leq n$, let $u_i(\chi)$ denote the quantity $C_i(\chi)/T_i$. (That is, $u_i(\text{LO})$ denotes τ_i's LO-criticality utilization, and $u_i(\text{HI})$ denotes its HI-criticality utilization).
3) Let J_1 denote the job with the earliest release time amongst all those that execute in $[t^*, t_f)$. Let a_1 denote its release time, and d_1 its deadline. (Note that $a_1 \leq t^*$.)

Lemma 1: All jobs that execute in $[t^*, t_f)$ have deadline $\leq t_f$.

Proof: Suppose not. Consider the latest instant t' in $[t^*, t_f)$ when a job with deadline $> t_f$ executes. Only those jobs in I that have release time $\geq t'$ and deadline $\leq t_f$ are sufficient to cause a deadline miss; this contradicts the assumed minimality of I. ■

It immediately follows that d_1, the deadline of the job J_1, is $\leq t_f$.

Lemma 2: Any LO-criticality task τ_i has

$$\eta_i \leq u_i(\text{LO})\big(a_1 + x(t_f - a_1)\big). \tag{11}$$

Proof: No LO-criticality job will execute after t^*. For it to execute after a_1, it must have a deadline no larger than J_1's virtual deadline, which is $(a_1 + x(d_1 - a_1))$. Therefore, no LO-criticality job with deadline $> (a_1 + x(t_f - a_1))$ will execute after a_1.

Suppose that some LO-criticality job with deadline $> (a_1 + x(t_f - a_1))$ were to execute, at some time $< a_1$. Let t' denote the latest instant at which any such job executes. This means that at this instant, there were no jobs with effective deadline $\leq (a_1 + x(t_f - a_1))$ awaiting execution. Hence by considering only those jobs in I that have release times $\geq t'$, the instance (with this LO-criticality task removed) also misses a deadline; this contradicts the assumed minimality of I. ■

Lemma 3: Any HI-criticality task τ_i has

$$\eta_i \leq \frac{u_i(\text{LO})}{x} a_1 + (t_f - a_1) u_i(\text{HI}). \tag{12}$$

Proof: We consider separately the cases when τ_i does not have a job with release time $\geq a_1$, and when it does.

Case A: If τ_i does not release a job at or after a_1. We claim that each job of τ_i has a virtual deadline $\leq (a_1 + x(t_f - a_1))$. To see why this is so, consider some job with a virtual deadline $> (a_1 + x(t_f - a_1))$, and let t' denote the latest instant at which this job executes. All jobs in I that have release times $\geq t'$ also miss a deadline; this contradicts the assumed minimality of I.

Since each job has a virtual deadline $\leq (a_1 + x(t_f - a_1))$, their actual deadlines are all $\leq \frac{a_1}{x} + (t_f - a_1)$. Therefore, their cumulative execution requirement is at most

$$\begin{aligned} & \frac{a_1}{x} u_i(\text{LO}) + (t_f - a_1) u_i(\text{LO}) \\ \leq \quad & \frac{a_1}{x} u_i(\text{LO}) + (t_f - a_1) u_i(\text{HI}). \end{aligned}$$

Case B: If τ_i releases a job at or after a_1. Let a_i denote the first release $\geq a_1$. The cumulative execution requirement of all jobs of τ_i is at most

$$\begin{aligned} & a_i u_i(\text{LO}) + (t_f - a_i) u_i(\text{HI}) \\ \leq \quad & \text{(Since } a_1 \leq a_i \text{ and } u_i(\text{LO}) \leq u_i(\text{HI})) \\ & a_1 u_i(\text{LO}) + (t_f - a_1) u_i(\text{HI}) \\ \leq \quad & \text{(Since } x \leq 1) \\ & \frac{a_1}{x} u_i(\text{LO}) + (t_f - a_1) u_i(\text{HI}). \end{aligned}$$

■

Let us sum the cumulative demand of all the tasks over $[0, t_f)$:

$$\begin{aligned} & \sum_{\chi_i = \text{LO}} \eta_i + \sum_{\chi_i = \text{HI}} \eta_i \\ \leq \quad & \sum_{\chi_i = \text{LO}} u_i(\text{LO})\big(a_1 + x(t_f - a_1)\big) \\ & + \sum_{\chi_i = \text{HI}} \frac{a_1}{x} u_i(\text{LO}) + (t_f - a_1) u_i(\text{HI}) \\ = \quad & a_1\big(U^{\text{LO}}_{\text{LO}}(\tau) + \frac{U^{\text{LO}}_{\text{HI}}(\tau)}{x}\big) \\ & + (t_f - a_1)(x U^{\text{LO}}_{\text{LO}}(\tau) + U^{\text{HI}}_{\text{HI}}(\tau)) \\ \leq \quad & \text{(By choice of } x \text{ [Eqn. 3], } (U^{\text{LO}}_{\text{LO}}(\tau) + \tfrac{U^{\text{LO}}_{\text{HI}}(\tau)}{x}) \leq 1) \\ & a_1 + (t_f - a_1)(x U^{\text{LO}}_{\text{LO}}(\tau) + U^{\text{HI}}_{\text{HI}}(\tau)). \end{aligned}$$

Since the amount of computation available on the processor is $t^* + \rho(t_f - t^*)$ and $a_1 \leq t^*$, it follows from the infeasibility of this instance that

$$\begin{aligned} & a_1 + (t_f - a_1)(x U^{\text{LO}}_{\text{LO}}(\tau) + U^{\text{HI}}_{\text{HI}}(\tau)) > a_1 + \rho(t_f - a_1) \\ \Leftrightarrow \quad & (t_f - a_1)(x U^{\text{LO}}_{\text{LO}}(\tau) + U^{\text{HI}}_{\text{HI}}(\tau)) > \rho(t_f - a_1) \\ \Leftrightarrow \quad & x U^{\text{LO}}_{\text{LO}}(\tau) + U^{\text{HI}}_{\text{HI}}(\tau) > \rho. \end{aligned}$$

Taking the contrapositive, it follows that $(x U^{\text{LO}}_{\text{LO}}(\tau) + U^{\text{HI}}_{\text{HI}}(\tau) \leq \rho)$ is sufficient to ensure HI-criticality schedulability by VDF-NM, as is claimed in this theorem. ■

We have thus established the correctness of Algorithm VDF-WM: by Theorem 1 the value assigned to x ensures the correctness of all LO-criticality behaviors whereas Theorem 5 guarantees the correct scheduling of all HI-criticality behaviors.

VI. Experimental evaluation

We have conducted a series of schedulability experiments to evaluate the relative effectiveness of the three scheduling strategies VDF-NM, VDF-NM with the pragmatic improvement (henceforth referred to as VDF-NM+), and VDF-WM in guaranteeing to correctly schedule MC implicit-deadline sporadic task systems. Our experiments were conducted upon randomly-generated task systems that were generated according to a minor modification of the workload-generation algorithm introduced by Guan et al. [15]. The

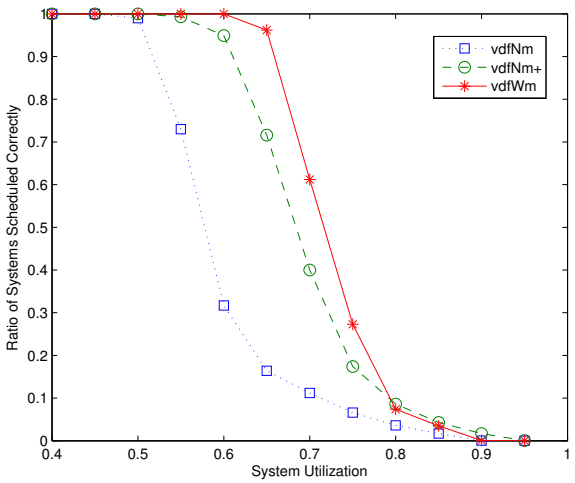

Figure 3. Example outcome of schedulability experiments, for parameters $[U_L, U_U] = [0.02, 0.2]; [T_L, T_U] = [5, 50]; [Z_L, Z_U] = [1, 4]; P = 0.5, \rho = 0.8$. The lowest line represents VDF-NM, the middle line represents VDF-NM+, and the top line represents VDF-WM.

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: $\max\left(U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{LO}}(\tau), U_{\text{HI}}^{\text{HI}}(\tau)\right)$.
- $[U_L, U_U]$: Utilizations are uniformly drawn from this range; $0 \leq U_L \leq U_U \leq 1$.
- $[T_L, T_U]$: Task periods are uniformly drawn from this range; $0 < T_L \leq T_U$.
- $[Z_L, Z_U]$: 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; $0 \leq P \leq 1$.

To generate a task system for a given combination of parameter values, the task-generation algorithm repeatedly adds tasks to an initially empty system until the utilization bound is met (see [15]). We considered a fairly large number (several hundreds) of different combinations of parameter values. A sample outcome is depicted graphically in Figure 3 — the fraction of systems that were determined to be schedulable is depicted on the y-axis as a percentage, and the system utilization U_{bound} on the x-axis. Each data-point was obtained by randomly generating 1000 task systems, testing each for schedulability according to all three algorithms, and calculating the percentage of systems deemed schedulable by each algorithm.

Although we do not claim that our experiments are comprehensive enough in coverage to enable us to draw authoritative conclusions, they do point to some pretty convincing trends. It was very evident in all our experiments that VDF-NM+ consistently exhibits noticeably superior performance over VDF-NM; i.e., the pragmatic improvement to the EDF-schedulability test of VDF-NM that was described in Section IV seems to provide significant benefit. Also, VDF-WM consistently exhibits noticeable improvement over VDF-NM+, indicating that self-monitoring in processors, if available, can be exploited to ensure considerable enhancement of schedulability. We do not feel comfortable making quantitative claims about the degree of such improvement based on our experiments since this is necessarily influenced by the nature of our random workload generator, but instead simply report our observations.

– All three algorithms were optimal for system utilization values that were no larger than about three-fifths times ρ. (This is as expected, given the speedup bound of $\phi \approx 0.61$ derived in Section III-A which holds for all three algorithms.) The percentage of schedulable systems falls off sooner, and more rapidly, for VDF-NM than for VDF-NM+, which in turn falls off more rapidly than for VDF-WM.

– Across all the simulation experiments that we conducted across a wide range of parameters, it appears that the simple pragmatic improvement to VDF-NM's schedulability testing that was implemented in VDF-NM+ provides between *one-half* to *two-thirds* the improvement that the more powerful platform capabilities of self-monitoring exploited in VDF-WM provides, with larger improvement ratios occurring at smaller system utilizations.

VII. Summary and Conclusions

We have considered here the scheduling of mixed-criticality implicit-deadline sporadic task systems in the face of uncertainty in both *(i)* the amount of execution required by the tasks, and *(ii)* the rate at which the processor upon which the tasks are executing is able to complete work. A good deal of research has recently been done addressing uncertainty in execution amounts; however, dealing with uncertainty in processor speeds is relatively less understood. Given the emergent trend towards incorporating asynchronous components in advanced modern processors, we believe this is a shortcoming of current MC scheduling research that this work seeks to address. We have accordingly

- proposed a formal model for representing mixed-criticality systems that are comprised of a finite collection of independent implicit-deadline tasks executing upon a varying-speed preemptive processor;
- separately considered the situations where the processor is *self-monitoring* – knows the rate at which it is executing work – or not;
- designed scheduling algorithms, and associated schedulability tests, for MC scheduling upon both kinds of processors; and
- evaluated these algorithms both theoretically and via schedulability experiments on randomly-generated workloads.

The fundamental difference of the varying-speed mixed-criticality model and classic multi-WCET one is that if processors can monitor their speed, one can notice speed-dropping much earlier than the resulting potential low-criticality WCET violation. This is one of the main motivations for improvement over current state of art on mixed-criticality scheduling upon constant-speed processors.

Primarily for ease of exposition, we have restricted the discussion in this paper to *dual*-criticality systems. All of our techniques extend in a straightforward manner to systems with more than two criticality levels defined, although it is not yet clear whether our theoretical analyses are also easily extendible – we plan to explore this as future work. Also as future work, we will seek to extend our results to more general recurrent real-time workload models, as well as to multiprocessor platforms.

ACKNOWLEDGMENT

This research was supported in part by the National Science Foundation via grants CNS 1016954, CNS 1115284, and CNS 1218693; and the Army Research Office via grant W911NF-09-1-0535.

REFERENCES

[1] K. Albers and F. Slomka. An event stream driven approximation for the analysis of real-time systems. In *Proceedings of the EuroMicro Conference on Real-Time Systems (ECRTS)*, pages 187–195. IEEE Computer Society Press, 2004.

[2] S. Baruah. Certification-cognizant scheduling of tasks with pessimistic frequency specification. In *Proceedings of the IEEE Symposium on Industrial Embedded Systems (SIES)*. IEEE Press, 2012.

[3] 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)*. IEEE Computer Society, 2012.

[4] 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.

[5] S. Baruah and A. Burns. Sustainable scheduling analysis. In *Proceedings of the IEEE Real-time Systems Symposium (RTSS)*, pages 159–168. IEEE Computer Society Press, 2006.

[6] S. Baruah and B. Chattopadhyay. Response-time analysis of mixed criticality systems with pessimistic frequency specification. In *Proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2013.

[7] S. Baruah and Z. Guo. Mixed-criticality scheduling upon varying-speed processors. In *Proceedings of the 34th IEEE Real-Time Systems Symposium (RTSS)*. IEEE Computer Society Press, 2013.

[8] D. Bull, S. Das, K. Shivshankarand, G. Dasika, K. Flautner, and D. Blaauw. A power-efficient 32b ARM ISA processor using timing-error detection and correction for transient- error tolerance and adaptation to PVT variation. In *Proceedings of the IEEE International Solid-State Circuits Conference*, pages 284–285, 2010.

[9] A. Burns and R. Davis. Mixed-criticality systems: A review. http://www-users.cs.york.ac.uk/ burns/review.pdf.

[10] A. Burns and R. Davis. Mixed criticality on controller area network. In *Proceedings of the 25th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 125–134, 2013.

[11] G. C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Second edition, 2005.

[12] T. Deb. Asynchronous microprocessors: The clockless future. *Electronics For You*, pages 64–68, 2010.

[13] F. Eisenbrand and T. Rothvoß. EDF-schedulability of synchronous periodic task systems is coNP-hard. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, 2010.

[14] 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)*. IEEE Computer Society Press, 2011.

[15] N. Guan, P. Ekberg, M. Stigge, and W. Yi. Improving the scheduling of certifiable mixed-criticality sporadic task systems. Technical Report 2013-008, Department of Information Technology, Uppsala University, 2013.

[16] M. Laurence. Keynote address: Low-power high performance asynchronous processors. In *ESWEEK: Embedded Systems Week*, Sept. 2013.

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

[18] A. Mok. *Fundamental Design Problems of Distributed Systems for The Hard-Real-Time Environment*. PhD thesis, Laboratory for Computer Science, Massachusetts Institute of Technology, 1983. Available as Technical Report No. MIT/LCS/TR-297.

[19] D. Müller and A. Masrur. The schedulability region of two-level mixed-criticality systems based on EDF-VD. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, 2014.

[20] D. Socci, P. Poplavko, S. Bensalem, and M. Bozga. Mixed critical earliest deadline first. In *Proceedings of the 2013 25th Euromicro Conference on Real-Time Systems (ECRTS)*. IEEE Computer Society Press, 2013.