2016 28th Euromicro Conference on Real-Time Systems

Scheduling mixed-criticality systems to guarantee some service under all non-erroneous behaviors

Sanjoy Baruah
The University of North Carolina
Chapel Hill, NC. USA
baruah@cs.unc.edu

Alan Burns
The University of York
York, Yorkshire. UK
alan.burns@york.ac.uk

Zhishan Guo
The University of North Carolina
Chapel Hill, NC. USA
zsguo@cs.unc.edu

***Abstract*—Many reactive systems must be designed and analyzed prior to deployment in the presence of considerable epistemic uncertainty: the precise nature of the external environment the system will encounter, as well as the run-time behavior of the platform upon which it is implemented, cannot be predicted with complete certainty prior to deployment. The widely-studied Vestal model for mixed-criticality workloads addresses uncertainties in estimating the worst-case execution time (WCET) of real-time code. Different estimations, at different levels of assurance, are made about these WCET values; it is required that all functionalities execute correctly if the less conservative assumptions hold, while only the more critical functionalities are required to execute correctly in the (presumably less likely) event that the less conservative assumptions fail to hold but the more conservative assumptions do. A generalization of the Vestal model is considered here, in which a degraded (but non-zero) level of service is required for the less critical functionalities even in the event of only the more conservative assumptions holding. An algorithm is derived for scheduling dual-criticality implicit-deadline sporadic task systems specified in this more general model upon preemptive uniprocessor platforms, and proved to be speedup-optimal.**

I. INTRODUCTION[1]

We consider the preemptive uniprocessor scheduling of systems of dual-criticality implicit-deadline sporadic tasks represented using a generalization of the Vestal model [15]. In the Vestal model each task τ_i is characterized by the parameters $(\chi_i, C_i^L, C_i^H, T_i)$, where $\chi_i \in \{\text{LO}, \text{HI}\}$ denotes its criticality with LO denoting lower criticality than HI, C_i^L and C_i^H its LO and HI criticality worst-case execution times (WCETs) with $C_i^H \geq C_i^L$, and T_i its period. The run-time scheduling objective is to ensure that

a. if every job of every task τ_i completes within C_i^L units of execution then all jobs complete by their deadlines; and
b. if a job of some task τ_i fails to complete despite being allowed to execute for C_i^L time units, then all jobs of each HI-criticality task τ_i should receive up to C_i^H units of execution by their respective deadlines, while jobs of LO-criticality tasks are not required to receive any execution.

This paper has passed an Artifact Evaluation process. For additional details, please refer to http://ecrts.org/artifactevaluation.

Several algorithms (including EDF-VD [1], [2], AMC [3], MC-EDF [14]) have been proposed for scheduling such systems upon preemptive uniprocessor platforms. EDF-VD and MC-EDF are known to be *speedup-optimal* algorithms with speedup bound $\frac{4}{3}$ for this purpose, in the following sense:

- If an optimal clairvoyant algorithm can schedule a given task system correctly upon a unit-speed processor, then these algorithms, too, can schedule the same system correctly upon a processor that is of speed $\frac{4}{3}$; and
- It has been shown [1, Theorem 5] that there exist task systems schedulable by an optimal clairvoyant algorithm upon a unit-speed processor that no non-clairvoyant algorithm can guarantee to schedule correctly upon a processor of speed strictly less than $\frac{4}{3}$.

An extension to the Vestal model [6]. The original Vestal model proved very successful in identifying some of the core challenges that arise in resource-efficient scheduling of mixed-criticality systems, and spawned a large body of research that proposed solutions to some of these challenges. However, this model has met with some criticism from systems engineers that it does not match their expectations in some important aspects. In this paper, we focus upon one such aspect: *in the event of some jobs executing beyond their* LO*-criticality WCET estimates,* LO*-criticality jobs should nevertheless be guaranteed some amount of execution prior to their deadlines.* This desideratum was addressed in [6] by modifying the specification and semantics of the Vestal model in two ways:

§**1.** While each task τ_i continues to be characterized by the two WCET parameters C_i^L and C_i^H, it is required that

1) If $\chi_i = \text{HI}$ then $C_i^H \geq C_i^L$ (this is as in the original Vestal model);
2) If $\chi_i = \text{LO}$, then $C_i^H \leq C_i^L$ (this is different).

§**2.** The run-time scheduling objectives are extended in the following manner to ensure a degraded (but non-zero) level of service for LO-criticality tasks in the event of HI-criticality tasks executing beyond their LO-crtiticality WCETs:

[1] Some familiarity is assumed here on the part of the reader with the mixed-criticality scheduling model introduced by Vestal [15] and reviewed in, e.g. [7].

2159-3833/16 $31.00 © 2016 IEEE
DOI 10.1109/ECRTS.2016.12

a. if each job of each task τ_i completes within C_i^L units of execution then all jobs complete by their deadlines; and
b. if a job of some HI-criticality task τ_i fails to complete despite being allowed to execute for C_i^L time units, then all jobs of all HI-criticality tasks τ_i should be allowed to execute for up to C_i^H units by their deadlines; additionally *all jobs of all* LO*-criticality tasks* τ_i *are guaranteed to receive at least* C_i^H *units of execution by their deadlines*.

An interpretation of the extended model. Vestal [15] had suggested that the different WCET parameters of each task be thought of as estimates, at different levels of assurance, of the true WCET parameter of the task; intuitively speaking, although one does not know for certain precisely what the maximum duration a job of the task τ_i may take to complete its execution, one has a greater degree of confidence that this maximum duration is bounded from above by the larger value of C_i^H than that it is bounded by the smaller value of C_i^L.

In the extension [6] of the standard Vestal model for dual-criticality systems, it is perhaps helpful to interpret the WCET parameters of HI-criticality and LO-criticality tasks differently. During run-time jobs of the HI-criticality tasks are required to execute to completion, but the run-time environment monitors and *budgets* the execution of jobs generated by LO-criticality tasks — any such job will be suspended (or perhaps terminated) once it consumes its budgeted amount of execution, regardless of whether it has completed execution or not. That is, the WCET parameters of HI-criticality tasks are assumptions or *rely conditions* [12], and the WCET parameters of LO-criticality tasks are corresponding *guarantees*, in the following sense: if each HI-criticality job completes upon executing for no more than the LO-criticality (HI-criticality, respectively) WCET of the task that generated it, then each LO-criticality job is guaranteed an execution of at least the LO-criticality (HI-criticality, resp.) WCET of the task that generated it. In other words, by *assuming* that each job of each HI-criticality task completes upon executing for no more than its LO-criticality WCETs, we are able to *guarantee* each LO-criticality job an amount of execution up to its LO-criticality WCET. If instead we make the *more conservative assumption* that each job of each HI-criticality task may need to execute for up to its HI-criticality WCET to complete, we are only able to make the *weaker guarantee* to the LO-criticality tasks that each LO-criticality job will get to execute for a smaller amount as specified by its HI-criticality WCET parameter.

Observe that with regards to modeling capabilities, this extended model is a strict generalization of the original model of Vestal. This follows from the observation that the modeling intent of the original Vestal model –that no execution guarantees are required for LO-criticality tasks in the event that any HI-criticality job executes beyond its LO-criticality WCET– may be represented in this more general model and extended semantics by simply setting the HI-criticality WCET C_i^H of each LO-criticality task equal to zero.

This research. In this paper, we obtain an algorithm for the preemptive uniprocessor scheduling of dual-criticality task systems represented in this more general model, and prove that our algorithm has a speedup factor equal to $\frac{4}{3}$. Since this model is a generalization of the one for which the lower bound of $\frac{4}{3}$ on speedup was proved in [1, Theorem 5], it follows that no algorithm for scheduling the more general model may have a speedup bound smaller than $\frac{4}{3}$ and our algorithm is thus speedup-optimal. Our algorithm is obtained using techniques that are inspired by, and based upon, some recently-introduced [13], [4] techniques in which tasks are scheduled assuming a *fluid* model. In the fluid model, several tasks may execute simultaneously upon a single processor with each assigned a fraction of the processor's computing capacity, subject to the constraint that the sum of the fractions assigned to all the tasks at each instant in time not exceed the capacity of the processor.

Organization. The remainder of this paper is organized as follows. In Section II we formally describe the task model we use, and briefly review some needed prior recent research concerning the fluid scheduling of dual-criticality task systems. We derive, and prove the correctness of, our proposed algorithm for scheduling dual-criticality implicit-deadline sporadic task systems represented using the more general model in Section III. Our algorithm can also be used to schedule task systems represented using the original Vestal model [15]; in Section IV we compare, both formally and via simulation experiments upon randomly-generated task sets, the performance of our algorithm and Algorithm EDF-VD [1], [2] for scheduling task systems represented using the original Vestal model.

II. SYSTEM MODEL

In this paper, we consider the scheduling of systems of independent dual-criticality implicit-deadline sporadic tasks upon a shared preemptive processor. We assume that a dual-criticality implicit-deadline sporadic task τ_i is characterized by the parameters $(T_i, C_i^L, C_i^H, \chi_i)$, where $\chi_i \in \{\text{LO}, \text{HI}\}$ denotes its criticality, C_i^L and C_i^H its LO and HI criticality WCETs, and T_i its period. We require that if $\chi_i = \text{LO}$ than $C_i^L \geq C_i^H$, while if $\chi_i = \text{HI}$ then $C_i^L \leq C_i^H$. Some additional notation: we let $u_i^L \stackrel{\text{def}}{=} (C_i^L/T_i)$ and $u_i^H \stackrel{\text{def}}{=} (C_i^H/T_i)$ denote the LO-criticality and HI-criticality *utilizations* of task τ_i.

Example 1: An example task system comprising three tasks is depicted in Table I. Observe that as mandated by the model, the LO-criticality tasks τ_1 and τ_2 have $C_i^L \geq C_i^H$, while the HI-criticality task τ_3 has $C_i^L \leq C_i^H$.

We point out that since the sum of the utilizations of the tasks at their own criticality levels $= (u_1^L + u_2^L + u_3^H)$ $= (0.2 + 0.4 + 0.6) > 1$, the system cannot be scheduled by the *Worst-Case Reservations (WCR)* approach [1], [2] of simply reserving for each task enough of the processor to independently ensure its correctness under all legal circumstances. ■

System *behaviors*. Since the period parameter of a sporadic task denotes the minimum (rather than exact) separation

	T_i	C_i^L	C_i^H	χ_i	u_i^L	u_i^H
τ_1	10	2	1	LO	0.2	0.1
τ_2	20	8	2	LO	0.4	0.1
τ_3	30	6	18	HI	0.2	0.6

TABLE I
EXAMPLE TASK SYSTEM

between successive jobs generated by the task, and WCET's merely denote estimated upper bounds on the actual execution time needed to complete executing a job of the task, a single sporadic task system may exhibit different *behaviors* during different executions. As stated above, we assume that the run-time environment budgets the execution of jobs generated by LO-criticality tasks — any such job will be terminated once it consumes its budgeted amount of execution, regardless of whether it has completed execution or not. The *criticality level* of a behavior is determined by how much execution is needed by the HI-criticality jobs in order to complete execution in that behavior:

- If every HI-criticality job completes upon executing for no more than the LO-criticality WCET of the task that generated it, then the behavior is defined to be a LO-*criticality behavior*.
- every behavior that is not a LO-criticality behavior in which every HI-criticality job completes upon executing for no more than the HI-criticality WCET of the task that generated it is defined to be a HI-*criticality behavior*.
- All other behaviors are *erroneous*.

Correctness criterion. We define an algorithm for scheduling MC task systems to be *correct* if it is able to schedule any system in such a manner that both the following properties are satisfied.

- During all LO-criticality behaviors of the system, each HI-criticality job receives enough execution between its release time and deadline to complete, and each LO-criticality job either completes or receives at least its LO-criticality WCET, between its release time and deadline.
- During all HI-criticality behaviors of the system, all HI-criticality jobs receive enough execution between their release time and deadline to complete, and each LO-criticality job either completes or receives at least its HI-criticality WCET (which, recall, is $\leq$ its LO-criticality WCET), between its release time and deadline.

Some additional notation. We now describe some notation that we will be using later in this document. We will let τ denote a collection of n dual-criticality implicit-deadline sporadic tasks that are to be scheduled upon a preemptive unit-speed processor. As a general rule, τ with a subscript (as in τ_i) denotes an individual task in τ; however, $\tau_H \subseteq \tau$ ($\tau_L \subseteq \tau$, respectively) denotes the collection of all the HI-criticality tasks (all the LO-criticality tasks, resp.) in τ.

---

1) Each τ_i initially executes at a constant rate θ_i^L. That is, at each time-instant it is executing upon θ_i^L fraction of a processor.
2) If a job of any task $\tau_i \in \tau_H$ does not complete despite having received C_i^L units of execution (equivalently, having executed for a duration (C_i^L/θ_i^L)), then
   - All LO-criticality tasks are immediately discarded, and
   - Each HI-criticality task henceforth executes at a constant rate θ_i^H.

Fig. 1. The run-time scheduling strategy used by Algorithm MC-Fluid

Various system utilization parameters are defined for τ as follows:

$$
\begin{aligned}
U_L^L &\stackrel{\text{def}}{=} \sum_{\tau_i \in \tau_L} u_i^L \\
U_H^L &\stackrel{\text{def}}{=} \sum_{\tau_i \in \tau_H} u_i^L \\
U_L^H &\stackrel{\text{def}}{=} \sum_{\tau_i \in \tau_L} u_i^H \\
U_H^H &\stackrel{\text{def}}{=} \sum_{\tau_i \in \tau_H} u_i^H
\end{aligned}
$$

A. Fluid scheduling of dual-criticality systems

The MC-Fluid scheduling algorithm [13] was designed for scheduling dual-criticality implicit-deadline sporadic task systems upon identical multiprocessor platforms under the *fluid scheduling* model, which allows for schedules in which individual tasks may be assigned a fraction ≤ 1 of a processor (rather than an entire processor, or none) at each instant in time. (Although MC-Fluid was designed as a *multi*processor scheduling algorithm, we will be applying it to scheduling upon uniprocessor platforms; hence our use of the results in [13], [4] initialize the number of processors to 1: $m \leftarrow 1$.)

MC-Fluid operates in the following manner. Prior to run-time, it computes LO-criticality and HI-criticality *execution rates* θ_i^L and θ_i^H for each task $\tau_i \in \tau$ such that the run-time scheduling algorithm depicted in Figure 1 constitutes a correct scheduling strategy for τ. An algorithm for computing suitable values for the θ_i^L and θ_i^H parameters is presented in [13]. It is shown in [13] that this approach has a *speedup factor* no worse than $(1+\sqrt{5})/2 \approx 1.62$: if a given task system τ can be scheduled correctly by an optimal clairvoyant scheduler upon an m-processor platform, then the run-time algorithm of Figure 1, with values for the θ_i^L and θ_i^H parameters computed in the manner derived in [13], will successfully schedule τ upon an m-processor platform in which each processor is faster by a factor of 1.62. A superior speedup bound was subsequently proved in [4]: it was shown that if a task system can be scheduled correctly by an optimal clairvoyant scheduler upon an m-processor platform then the run-time algorithm of

1) Each τ_i initially executes at a constant rate θ_i^L.
2) If a job of any task $\tau_i \in \tau_H$ does not complete despite having received C_i^L units of execution (equivalently, having executed for a duration (C_i^L/θ_i^L)), then each task τ_i immediately changes its execution rate and henceforth executes at a constant rate θ_i^H.

Fig. 2. Modified run-time scheduling strategy

Figure 1, with values for the θ_i^L and θ_i^H parameters computed as in [13], will in fact successfully schedule τ upon an m-processor platform in which each processor is faster by a factor of $\frac{4}{3}$.

III. A SCHEDULING ALGORITHM

In this section we describe how to extend and adapt the results described in Section II-A above to construct correct pre-emptive uniprocessor scheduling strategies for dual-criticality implicit-deadline sporadic task systems that are characterized using the extended model, in which each LO-criticality task expects some level of service even in HI-criticality behaviors.

As a first modification, LO-criticality tasks cannot be dropped entirely even in the event of some HI-criticality job executing beyond its LO-criticality WCET (as is done in step 2 of the run-time strategy that is used by MC-Fluid and depicted in Figure 1). The run-time scheduling strategy is therefore modified to the form shown in Figure 2. Of course, the θ_i^L and θ_i^H values must be computed differently now — amongst other factors, the values of θ_i^H for LO-criticality tasks were never used in the runtime strategy depicted in Figure 1 (and therefore did not need to be computed), but they are needed in the runtime strategy of Figure 2.

Computing the θ_i^L's and θ_i^H's. Given a dual-criticality task system τ, our algorithm for computing the execution rates proceeds in the following four steps.

Step 1. We first reserve, for each LO-criticality task τ_i, a fraction u_i^H of the processor for τ_i's exclusive use in all behaviors. This uses up a fraction $\left(\sum_{\tau_i \in \tau_L} u_i^H\right)$ or U_L^H of the computing capacity of the processor.

Step 2. We next obtain a task system $\tilde{\tau}$ from the original task system τ by including in $\tilde{\tau}$

- for each HI-criticality task $\tau_i \in \tau_H$, a HI-criticality task $\tilde{\tau}_i$ with $\tilde{u}_i^L \leftarrow u_i^L$ and $\tilde{u}_i^H \leftarrow u_i^H$; and
- for each LO-criticality task $\tau_i \in \tau_L$, a LO-criticality task $\tilde{\tau}_i$ with utilization parameters $\tilde{u}_i^L \leftarrow (u_i^L - u_i^H)$ and $\tilde{u}_i^H \leftarrow 0$.

Observe that the cumulative HI-criticality utilization of all the tasks in task system $\tilde{\tau}$ is equal to

$$\begin{aligned} & \sum_{\tau_i \in \tau_H} \tilde{u}_i^H + \sum_{\tau_i \in \tau_L} \tilde{u}_i^H \\ = & \sum_{\tau_i \in \tau_H} u_i^H + \sum_{\tau_i \in \tau_L} 0 \\ = & \; U_H^H \end{aligned} \tag{1}$$

We can derive an analogous expression for the cumulative LO-criticality utilization of all the tasks in $\tilde{\tau}$'s:

$$\begin{aligned} & \sum_{\tau_i \in \tau_H} \tilde{u}_i^L + \sum_{\tau_i \in \tau_L} \tilde{u}_i^L \\ = & \sum_{\tau_i \in \tau_H} u_i^L + \sum_{\tau_i \in \tau_L} (u_i^L - u_i^H) \\ = & \sum_{\tau_i \in \tau_H} u_i^L + \sum_{\tau_i \in \tau_L} u_i^L - \sum_{\tau_i \in \tau_L} u_i^H \\ = & \; U_H^L + (U_L^L - U_L^H) \end{aligned} \tag{2}$$

Step 3. Observe that the task system $\tilde{\tau}$ obtained in Step 2 above is one that fits the "traditional" Vestal model, in that each LO-criticality task requires no service at all in HI-criticality behaviors ($\tilde{u}_i^H \equiv 0$ for all tasks with $\chi_i = \text{LO}$). Task system $\tilde{\tau}$ can therefore by correctly scheduled using algorithms developed for scheduling such traditional Vestal systems. In Step 1 above, we had pre-assigned a fraction U_L^H of the processor capacity to the LO-criticality tasks in τ; we will now compute execution rates for $\tilde{\tau}$ upon the remaining capacity of the processor – an amount $\left(1-U_L^H\right)$. We will use the technique of [13] to compute these execution rates; as stated in Section II, this technique is proved [4] speedup-optimal. Let $\tilde{\theta}_i^L$ and $\tilde{\theta}_i^H$ denote the execution rates so computed for task $\tilde{\tau}_i$, $1 \leq i \leq n$.

Step 4. Finally, we compute the execution rates θ_i^L and θ_i^H for all tasks in τ from the values computed for the corresponding tasks in $\tilde{\tau}$ by the technique of [13], by adding back the reserved capacities of Step 1 as follows:

- For each HI-criticality task,

$$\begin{aligned} \theta_i^L & \leftarrow \tilde{\theta}_i^L \\ \theta_i^H & \leftarrow \tilde{\theta}_i^H \end{aligned}$$

- For each LO-criticality task,

$$\begin{aligned} \theta_i^L & \leftarrow \tilde{\theta}_i^L + u_i^H \\ \theta_i^H & \leftarrow \tilde{\theta}_i^H + u_i^H \\ & = u_i^H \end{aligned}$$

(the last step following from the observation that $\tilde{\theta}_i^H$ is set equal to zero, since the technique of [13] assigns zero execution rates to all LO-criticality tasks in HI-criticality behaviors).

	χ_i	$\tilde{u}_i^L$	$\tilde{u}_i^H$
$\tilde{\tau}_1$	LO	0.1	0.0
$\tilde{\tau}_2$	LO	0.3	0.0
$\tilde{\tau}_3$	HI	0.2	0.6

TABLE II
EXAMPLE TASK SYSTEM - TRANSFORMED

A. An example

In this section, we illustrate the operation of the algorithm described above by applying it to the task system of Example 1, the parameters of which are enumerated in Table I.

In **step 1**, we reserve fractions $u_1^H = 0.1$ and $u_2^H = 0.1$ of the processor for the LO-criticality tasks τ_1 and τ_2.

Next, in **step 2** we define the task system $\tilde{\tau}$ in the following manner.

- Task τ_1 is a LO-criticality task; the task $\tilde{\tau}_1$ therefore has LO-criticality utilization equal to $(u_1^L - u_1^h) = (0.2 - 0.1)$ or 0.1, and HI-criticality utilization equal to zero.
- Task τ_2 is also a LO-criticality task; the task $\tilde{\tau}_2$ therefore has LO-criticality utilization equal to $(u_2^L - u_2^h) = (0.4 - 0.1)$ or 0.3, and HI-criticality utilization equal to zero.
- Task τ_3 is a HI-criticality task; the task $\tilde{\tau}_3$ therefore has LO-criticality utilization equal to $u_2^L = 0.2$, and HI-criticality utilization equal to $u_2^H = 0.6$.

This task system $\tilde{\tau}$ is depicted in tabular form in Table II.

In **step 3**, this task system is to be scheduled upon a processor of speed

$$\big(1 - (u_1^H + u_2^H)\big) = \big(1 - (0.1 + 0.1)\big) = 0.8$$

Observe that the LO-criticality utilizations of all the tasks in $\tilde{\tau}$ sum to 0.6 (i.e., $\sum_{i=1}^{3} \tilde{u}_i^L = 0.6$); the HI-criticality utilizations of all the tasks in $\tilde{\tau}$ also sum to 0.6 (i.e., $\sum_{i=1}^{3} \tilde{u}_i^H = 0.6$ as well). Hence any preemptive uniprocessor upon which $\tilde{\tau}$ is scheduled correctly by an optimal clairvoyant scheduling algorithm must be of speed ≥ 0.6. As stated in Section II, computing the execution rates according to the technique of [13], [4] yields a speedup bound of $\frac{4}{3}$'rds. Since $\big(0.6 \times \frac{4}{3}\big) = 0.8$, we would expect that $\tilde{\tau}$ is scheduled correctly upon a speed-0.8 processor using the execution rates computed according to the technique of [13], [4]. This is indeed the case, and applying the technique of [4] to $\tilde{\tau}$ yields the following execution rates:

$$\begin{array}{lccc} & \tilde{\theta}_1^L = 0.1; & \tilde{\theta}_2^L = 0.3; & \tilde{\theta}_3^L = 0.4 \\ \text{and} & \tilde{\theta}_1^H = 0.0; & \tilde{\theta}_2^H = 0.0; & \tilde{\theta}_3^H = 0.8 \end{array}$$

Finally applying **step 4** of the algorithm, the rates that are computed for the task system τ are then as follows:

- Task τ_1:
  $$\theta_1^L = \tilde{\theta}_1^L + u_1^H = 0.1 + 0.1 = 0.2$$
  $$\theta_1^H = u_1^H = 0.1$$
- Task τ_2:
  $$\theta_2^L = \tilde{\theta}_2^L + u_2^H = 0.3 + 0.1 = 0.4$$
  $$\theta_2^H = u_2^H = 0.1$$
- Task τ_3:
  $$\theta_3^L = \tilde{\theta}_3^L = 0.4$$
  $$\theta_3^H = \tilde{\theta}_3^H = 0.8$$

Hence the tasks τ_1, τ_2 and τ_3 are initially assigned execution rates $0.2, 0.4$, and 0.4 respectively; if HI-criticality behavior is detected, then the rates immediately change to $0.1, 0.1$, and 0.8 respectively. ■

B. A proof of correctness

The correctness of our algorithm follows directly from the correctness of the procedure for computing the execution rates derived in [4], which was proved correct there. Below, we briefly outline the main arguments to establish that our algorithm is indeed correct.

Correctness in LO-criticality behaviors. For this, it suffices to prove that $\theta_i^L \geq u_i^L$ for all $\tau_i \in \tau$. To do so, we will use a result concerning the $\tilde{\theta}_i^L$ parameter values that were computed during Step 3 of our algorithm by using the algorithm of [4]. The following statement was proved in [4] (re-stated here in the context of the task system $\tilde{\tau}$ that was defined in Step 2 of our algorithm):

From [4, Lemma 3]: For each $\tilde{\tau}_i \in \tilde{\tau}$, $\tilde{\theta}_i^L \geq \tilde{u}_i^L$.

For LO-criticality tasks, observe that Step 4 of our algorithm assigns each such task an execution rate θ_i^L that is equal to $\tilde{\theta}_i^L + u_i^H$:

$$\begin{array}{rcll} \theta_i^L & = & \tilde{\theta}_i^L + u_i^H & \\ & \geq & \tilde{u}^L + u_i^H & \text{(By [4, Lemma 3])} \\ & = & \big(u_i^L - u_i^H\big) + u_i^H & \text{(As set in Step 2)} \\ & = & u_i^L & \end{array}$$

Hence, we have $\theta_i^L \geq u_i^L$ for each LO-criticality task.

For HI-criticality tasks, Step 4 of our algorithm assigns each such task an execution rate θ_i^L that is equal to $\tilde{\theta}_i^L$. By [4, Lemma 3], this is $\geq \tilde{u}_i^L$; Step 2 of the algorithm assigns $\tilde{u}_i^L$ the value u_i^L. Hence, we have $\theta_i^L \geq u_i^L$ for each such HI-criticality task as well.

(Asymptotic) correctness in LO-criticality behaviors. We now prove that if the system exhibits HI-criticality behavior, the assigned execution rates are asymptotically (i.e., in steady state) adequate: $\theta_i^H \geq u_i^H$ for all $\tau_i \in \tau$.

For LO-criticality tasks, Step 4 assigns each task τ_i an execution rate equal to u_i^H; hence, asymptotic correctness for LO-criticality tasks follows immediately.

To show this for HI-criticality tasks, we use the following result from [4] concerning the $\tilde{\theta}_i^L$ parameter values that were computed during Step 3 of our algorithm by using the algorithm of [4]

From [4, Eqn (8)]: For each $\tilde{\tau}_i \in \tilde{\tau}_H$, $\tilde{\theta}_i^H \geq \tilde{u}_i^H$.

Step 4 of our algorithm assigns each HI-criticality task an execution rate θ_i^H that is equal to $\tilde{\theta}_i^H$. By [4, Eqn (8)], this is $\geq \tilde{u}_i^H$; Step 2 of the algorithm assigns $\tilde{u}_i^H$ the value u_i^H. Hence, we have $\theta_i^H \geq u_i^H$ for each such HI-criticality task, and asymptotic correctness for HI-criticality tasks is thereby established.

Correctness upon transition to HI-criticality behavior. For this, we will use the following result from [4]:

From [4, Lemma 4]: *Let t_o denote the first time-instant at which some job does not signal completion despite having executed for its LO-criticality WCET. Any HI-criticality job that is active (i.e., that has been released but has not completed execution) at time-instant t_o receives an amount of execution no smaller than its HI-criticality WCET prior to its deadline.*

Since for each HI-criticality task Step 2 of our algorithm assigns $\tilde{u}_i^H$ the value as u_i^H, this lemma can be used to show that during transition to HI-criticality behavior, jobs of HI-criticality tasks τ_i receive an amount of execution no smaller than their HI-criticality WCETs prior to their deadlines. The correctness of LO-criticality jobs during such transitions is trivial since each LO-criticality task τ_i always receives a share that is $\geq u_i^H$.

C. Speedup bound

We now prove that our algorithm has a speedup bound of $\frac{4}{3}$'rds. That is, suppose that some task system τ described in the extended Vestal model is scheduled correctly upon a speed-s processor by some optimal clairvoyant scheduling algorithm. We will prove (in Theorem 1 below) that if $s \leq \frac{3}{4}$ then τ is scheduled correctly upon a unit-speed processor by our scheduling algorithm.

Consider a behavior in which each task has jobs arriving at the maximum rate permitted, each LO-criticality job consuming all the budget allocated to it by the run-time mechanism, and each HI-criticality task's jobs executing for exactly their HI-criticality WCETs; the effective utilization of the resulting implicit-deadline task system is $\big(U_H^H + U_L^H\big)$. Since this behavior is assumed to be correctly scheduled, it must be the case that $\big(U_H^H + U_L^H\big)$ is no larger than the processor speed s:

$$U_H^H + U_L^H \leq s \tag{3}$$

Analogously to the argument above, a behavior in which each task has jobs arriving at the maximum rate permitted, each LO-criticality job consuming all the budget allocated to it by the run-time mechanism, and each HI-criticality task's jobs executing for exactly their LO-criticality WCETs has effective utilization $\big(U_H^L + U_L^L\big)$; this, too may be no larger than s:

$$U_H^L + U_L^L \leq s \tag{4}$$

Simplifying Inequality 3 using algebra, we have

$$\begin{aligned}
&U_H^H + U_L^H \leq s \\
&\quad\Leftrightarrow\quad U_H^H \leq s - U_L^H \\
&\quad\Rightarrow\quad U_H^H \leq s - s \times U_L^H \quad \text{(Since } s < 1\text{)} \\
&\quad\Leftrightarrow\quad U_H^H \leq s\big(1 - U_L^H\big)
\end{aligned} \tag{5}$$

Similarly simplifying Inequality 4, we obtain

$$\begin{aligned}
&U_H^L + U_L^L \leq s \\
&\quad\Leftrightarrow\quad U_H^L + U_L^L - U_L^H \leq s - U_L^H \\
&\quad\Rightarrow\quad U_H^L + U_L^L - U_L^H \leq s - s \times U_L^H \\
&\quad\Leftrightarrow\quad U_H^L + \big(U_L^L - U_L^H\big) \leq s\big(1 - U_L^H\big)
\end{aligned} \tag{6}$$

From Equation 1 and Equation 2, we observe that the expressions on the LHS of Inequalities 5 and 6 represent respectively the HI- and LO-criticality utilizations of the task system $\tilde{\tau}$. Since they are both $\leq s\big(1 - U_L^H\big)$, we conclude, from the $\frac{4}{3}$'rds speedup bound of Algorithm MC-Fluid [13], [4], that Step 3 of our algorithm is successful upon a processor of speed $\leq \frac{4}{3}s$. Hence

Theorem 1: Our scheduling algorithm has a speedup factor no worse than $\frac{4}{3}$: any instance that is scheduled correctly by an optimal clairvoyant algorithm upon a speed-s processor is scheduled correctly by our algorithm upon a unit-speed processor, for all values of $s \leq \frac{3}{4}$. ■

IV. Comparison with EDF-VD [1], [2]

As we had stated in Section I, the task model we consider in this paper is a generalization of the original Vestal model; hence, our algorithm can also be applied to task systems represented using the original Vestal model. In this section, we compare, both formally and via the use of simulation experiments, our algorithm and EDF-VD. These algorithms are both speedup optimal — they share the (optimal) speedup factor of $4/3$ — yet their performance in terms of schedulability varies. (Another algorithm that uses different scaling factors to compute virtual deadlines for different tasks is presented in [9], [8]; although that algorithm, which is based on iteratively adjusting the virtual deadlines of individual tasks while preserving schedulability, is shown to be very effective in practice, it is not speedup optimal – its speedup bound is instead given by the golden ratio, ≈ 1.618. In this paper we are restricting our attention to speedup-optimal algorithms, and so do not include a comparison with the algorithm in [9], [8].)

A. A theoretical comparison[2]

Below (Theorem 2) we show that the algorithm we have derived in this paper strictly dominates EDF-VD.

Lemma 1: There exist dual-criticality task systems schedulable by our algorithm that EDF-VD fails to schedule correctly.
Proof: The following is an example of such a task system:

Task ID	u_i^L	u_i^H	χ_i
τ_1	0.10	0.20	HI
τ_2	0.10	0.61	HI
τ_3	0.50	0	LO

The schedulability test of EDF-VD [1, Figure 1] computes a scaling factor x as follows:

$$x \leftarrow U_H^L/(1 - U_L^L)$$

[2]The proofs presented in this section pre-suppose familiarity with the results and proofs in [1], [13], [4].

and declares failure if

$$xU_L^L + U_H^H > 1$$

For our example task system above, it may be verified that $x \leftarrow 0.4$; hence $xU_L^L + U_H^H = 0.4 \times 0.1 + 0.81 > 1$ and EDF-VD consequently declares failure.

However, our scheduling algorithm (which merely computes rates according to the algorithms in [13], [4] for systems that are represented in the original Vestal model) does indeed declare the system schedulable; for instance, Algorithm MCF [4, Figure 2] deems the system schedulable and computes the values $\rho \leftarrow 0.81$ and $\theta^L \leftarrow [0.1681, 0.3198, 0.1]$.

■

Lemma 2: Any dual-criticality instance that is schedulable by EDF-VD is also schedulable by our algorithm.

Proof: An algorithm for computing execution rates (the θ_i^L and θ_i^H values) is defined to be *optimal* in [13, Definition 5], if it can find an assignment of values to these rates that renders a system feasible, whenever such values exist. It is then shown [13, Theorem 3] that the algorithm for computing the rates that was derived in [13] is in fact an optimal one.

Now, the scaling factor x computed by EDF-VD [1, Figure 1] can be interpreted in terms of execution rates, in the following manner. Assigning a value x_o to the scaling factor is equivalent to assigning each LO-criticality task τ_i execution rates $\theta_i^L \leftarrow u_i^L$ and $\theta_i^H \leftarrow 0$, and assigning each HI-criticality task τ_i a LO-criticality execution rate $\theta_i^L \leftarrow (u_i^L/x_o)$; the corresponding θ_i^H values for HI-criticality tasks can be obtained by reverse engineering of the relationship between the θ_i^L and θ_i^H values that is established in [13, Lemma 6]. Since the rate-assignment algorithm of [13] is optimal, it is therefore guaranteed to find these rates for any instance that is deemed schedulable by EDF-VD.

■

As a direct consequence of Lemmas 1 and 2 above, we conclude

Theorem 2: Our algorithm strictly dominates EDF-VD for the preemptive uniprocessor scheduling of implicit-deadline sporadic task systems represented using the original Vestal model [15]: all task systems that are correctly scheduled by EDF-VD are also correctly scheduled by our algorithm, and there are task systems correctly scheduled by our algorithm that EDF-VD fails to schedule correctly.

B. Schedulability experiments

The schedulability experiments reported in this section further explore the relationship between our new algorithm and Algorithm EDF-VD.

Workload Generation. Our experiments are conducted upon randomly generated mixed-criticality workloads that are generated using the workload-generators used in [10] [11] with some minor modification. The parameters of our workload generation algorithm are as follows:

- n: Number of tasks in the system, uniformly drawn from from the range $[5, 20]$;
- $P_H = 0.5$: The probability of a task being HI- criticality;
- U_L: Total LO-criticality utilization (varied from 0 to 1, with step-size 0.05);
- $[\mathcal{L}_H, \mathcal{U}_H] = [1, 2]$: The ratio of the HI-criticality utilization of a HI-criticality task to its LO-criticality utilization is uniformly drawn from this range;
- $[\mathcal{L}_L, \mathcal{U}_L] = [1/4, 1/2]$: The ratio of the HI-criticality utilization of a LO-criticality task to its LO-criticality utilization is uniformly drawn from this range.

For each task-set, we first use the UUniFast algorithm [5] to determine LO-criticality utilizations for all the tasks. Then for HI-criticality utilizations, after inflating the utilizations of HI-criticality tasks (which is similar to the steps in [10] [11]) for U_H^H's, we also shrink the utilizations of LO-criticality tasks for U_L^H. The detailed inflating and shrinking ratios uniformly distribute over the above-mentioned ranges $[\mathcal{L}_H, \mathcal{U}_H], [\mathcal{L}_L, \mathcal{U}_L]$. In case the set is obviously not feasible ($U_H^H + U_L^H > 1$), we discard and regenerate until $U_H^H + U_L^H \leq 1$.

Observations. In our experiment, $10,000$ task sets are generated for each given U_L, with U_L's varying from 0.4 to 0.95 (and are 0.05 apart). We focus on the metric of *acceptance ratio*, which denotes the fraction of the generated task sets that are deemed to be schedulable by the specified algorithm under specified conditions. In Figure 3 we report the average acceptance ratios as a function of the *normalized utilization bound* [4] —- the larger of the individual LO- and HI-criticality utilizations. It is clear that all the task sets are schedulable when their normalized utilization falls below 0.75 (the point in the graph corresponding to the x-axis value of 0.725 reflects the average acceptance ratio of all task sets with utilization bound between 0.7 and 0.75); this is to be expected since both algorithms have a speedup bound of $4/3$'rds. Although we do not claim that our experiments are comprehensive enough to enable us draw authoritative conclusions, it is evident that at least in our experiments our algorithm outperforms EDF-VD quite significantly for normalized utilizations > 0.75,

V. Conclusions

The Vestal model for mixed-criticality workloads proved very useful in identifying some of the core challenges that arise in resource-efficient scheduling of mixed-criticality systems, and succeeded in motivating the real-time community to devote considerable effort to understanding the scheduling behavior of mixed-criticality systems. However, this model does suffer from some shortcomings; one shortcoming of particular concern to systems engineers is that the model assumption that *in the event of some jobs executing beyond their* LO-*criticality WCET estimates,* LO-*criticality tasks may be abandoned entirely* is not in keeping with currently acceptable industrial practice. This shortcoming was addressed in [6] by proposing an extension to the specification and semantics of the Vestal model. In this paper, we have studied the preemptive uniprocessor scheduling of dual-criticality systems of implicit-deadline sporadic tasks represented in this extended model.

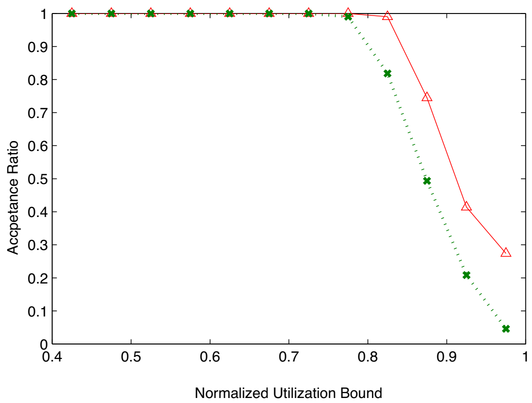

Fig. 3. Comparison of normalized utilization bounds. The upper line denotes the normalized utilization bounds of the new algorithm proposed in this paper; the lower line, those of EDF-VD.

We have shown that, at least from the perspective of speedup factor in a fluid scheduling model, this extension is available "for free."

ACKNOWLEDGEMENTS

We are grateful to Rob Davis for pointing out some shortcomings in an earlier version of this manuscript.

This research has been supported in part by EPSRC grant MCC (K011626/1), NSF grants CNS 1115284, CNS 1218693, CNS 1409175, and CPS 1446631, AFOSR grant FA9550-14-1-0161, ARO grant W911NF-14-1-0499, and a grant from General Motors Corp.

REFERENCES

[1] 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.

[2] S. Baruah, V. Bonifaci, G. D'angelo, H. Li, A. Marchetti-Spaccamela, S. Van Der Ster, and L. Stougie. Preemptive uniprocessor scheduling of mixed-criticality sporadic task systems. *Journal of the ACM*, 62(2):14:1–14:33, May 2015.

[3] 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.

[4] S. Baruah, A. Easwaran, and Z. Guo. MC-Fluid: simplified and optimally quantified. In *Real-Time Systems Symposium (RTSS), 2015 IEEE*, Dec 2015.

[5] E. Bini and G. Buttazzo. Biasing effects in schedulability measures. In *Proceedings of the 16th EuroMicro Conference on Real-Time Systems (ECRTS)*, 2004.

[6] A. Burns and S. Baruah. Towards a more practical model for mixed criticality systems. In *Proceedings of the International Workshop on Mixed Criticality Systems (WMC)*, December 2014.

[7] A. Burns and R. Davis. Mixed-criticality systems: A review (6th edition). http://www-users.cs.york.ac.uk/~burns/review.pdf (Accessed on Jan 6th, 2016), 2015.

[8] 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 Press.

[9] 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.

[10] 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.

[11] Z. Guo and S. Baruah. The concurrent consideration of uncertainty in wcets and processor speeds in mixed criticality systems. In *Proceedings of the International Conference on Real-Time and Network Systems*, RTNS '15, New York, NY, USA, 2015. ACM.

[12] C. Jones. *Development Methods for Computer Programs including a Notion of Interference*. PhD thesis, Oxford University, 1981. Printed as Programming Research Group, Technical Monograph 25.

[13] J. Lee, K.-M. Phan, X. Gu, J. Lee, A. Easwaran, I. Shin, and I. Lee. MC-Fluid: Fluid model-based mixed-criticality scheduling on multiprocessors. In *Real-Time Systems Symposium (RTSS), 2014 IEEE*, pages 41–52, Dec 2014.

[14] 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 '13, Paris (France), 2013. IEEE Computer Society Press.

[15] 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.