
Proceedings Paper:
Baruah, Sanjoy and Burns, Alan orcid.org/0000-0001-5621-8816 (2019) Incorporating Robustness and Resilience into Mixed-Criticality Scheduling Theory. In: 2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC). nternational Symposium on Real-Time Distributed Computing (ISORC). IEEE, pp. 155-162.

https://doi.org/10.1109/ISORC.2019.00038

Incorporating Robustness and Resilience into Mixed-Criticality Scheduling Theory

Sanjoy Baruah
Washington University in Saint Louis
baruah@wustl.edu

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

***Abstract*—Mixed-criticality scheduling theory (MCSh) was developed to allow for more resource-efficient implementation of systems comprising different components that need to have their correctness validated at different levels of assurance. As originally defined, MCSh deals exclusively with pre-runtime verification of such systems; hence many mixed-criticality scheduling algorithms that have been developed tend to exhibit rather poor survivability characteristics during run-time. (E.g., MCSh allows for less-important ("LO-criticality") workloads to be completely discarded in the event that run-time behavior is not compliant with the assumptions under which the correctness of the LO-criticality workload should be verified.) Here we seek to extend MCSh to incorporate survivability considerations, by proposing quantitative metrics for the *robustness* and *resilience* of mixed-criticality scheduling algorithms. Such metrics allow us to make quantitative assertions regarding the survivability characteristics of mixed-criticality scheduling algorithms, and to compare different algorithms from the perspective of their survivability. We propose that MCSh seek to develop scheduling algorithms that possess superior survivability characteristics, thereby obtaining algorithms with better survivability properties than current ones (which, since they have been developed within a survivability-agnostic framework, tend to focus exclusively on pre-runtime verification and ignore survivability issues entirely).**

I. Introduction

Many safety-critical systems are required to have their correctness *validated* (in some cases, *formally verified*) prior to their deployment; in some application domains such as civilian aviation, such a priori validation is mandated by statutory certification requirements. For safety-critical systems in which different functionalities need to have their correctness validated to different degrees of assurance, an approach towards doing such validation that was first advocated by Vestal [1] has garnered a lot of attention in the real-time scheduling theory community.[1] We consider here the preemptive uniprocessor scheduling of systems of independent sporadic tasks that are represented using the Vestal model [1]. Each task τ_i is characterized by the parameters $(T_i, C_i^L, C_i^H, \chi_i)$, where T_1 denotes its period, $\chi_i \in \{\text{LO}, \text{HI}\}$ its criticality with LO denoting lower criticality than HI, and C_i^L and C_i^H its LO and HI criticality worst-case execution time (WCET) estimates (with $C_i^H \geq C_i^L$). The intuition behind this model is that HI-criticality tasks need to have their correctness validated at a higher level of assurance than LO-criticality tasks. The C_i^H and C_i^L parameters represent different estimates, made at levels of assurance consistent with the higher and lower levels of assurance respectively, of the actual (unknown) WCET of task τ_i. Since we only need to validate each task at a level of assurance consistent with its own specified criticality level, the correctness criterion in verification of this system translates to the requirement that

a. if every job of every task τ_i completes execution within C_i^L units of execution then all jobs should meet their deadlines; and
b. if a job of some task τ_i fails to complete execution despite having executed 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. (Since this situation violates the assumptions under which LO-criticality verification is required to be performed, no requirements are placed upon the execution of jobs of LO-criticality tasks).

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 (see, e.g, [3], [4]) that it does not match expectations of systems developers in some important aspects — most of these criticisms are with regards to the behavior of proposed mixed-criticality scheduling algorithms in the event that some jobs execute beyond their LO-criticality WCET estimates. Here we seek to better understand some of these concerns, and construct a formal framework within which to address these concerns.

Organization. The remainder of this document is organized in the following manner. The distinct concepts of *pre-run-time verification* and *run-time survivability* are both foundational to the analysis and evaluation of high-integrity safety-critical systems; we seek to highlight the relationship (in particular, emphasizing the difference) between these two concepts in Section II. In Section III we formally present the Vestal model for mixed-criticality workloads and briefly review some related work that forms the basis of the remainder of this paper. Our major technical contributions are to be found in Section IV, where we explain our proposed approach for extending the Vestal model to incorporate considerations of run-time survivability. We conclude in Section V with a brief summary, and a discussion on future research directions.

[1] Some familiarity is assumed here on the part of the reader with the mixed-criticality scheduling model introduced by Vestal [1] and reviewed in, e.g. [2].

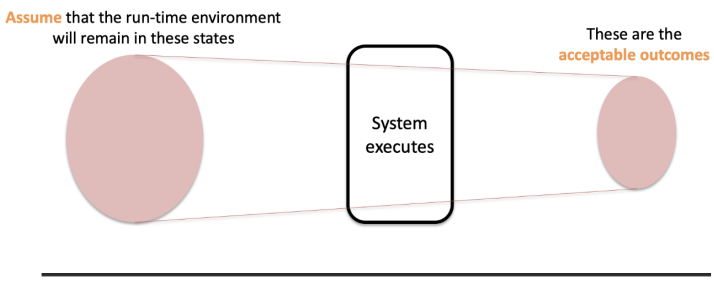

Fig. 1. Assume-Guarantee Specifications

II. Verification versus Survivability

The issue of ensuring correctness in high-integrity safety-critical systems may be considered from two rather distinct perspectives: *(i)* (pre-run-time) verification, and *(ii)* survivability. Pre-run-time ***verification*** of a safety-critical system is the process of ensuring, prior to deployment, that the run-time behavior of the system will be consistent with expectations. In one commonly used formal framework for pre-run-time verification (see Figure 1), *assumptions* are made regarding the kinds of circumstances that will be encountered by the system during run-time, and *guarantees* specified that the run-time behavior of the system is required to satisfy (provided that the assumptions hold). In contrast, ***survivability*** addresses expectations of system behavior in the event that the assumptions fail to fully hold (in which case a "*fault*" may be stated to have occurred during run-time). Survivability may further be considered to comprise two notions — robustness and resilience. Informally, the *robustness* of a system is a measure of the degree of fault it can tolerate without compromising on the quality of service it offers; *resilience*, by contrast, refers to the degree of fault for which it can provide degraded yet acceptable quality of service.

The model proposed by Vestal [1], and much of the subsequent scheduling theory that has been developed based upon this model, are designed to deal with verification, not survivability. Hence many mixed-criticality scheduling algorithms that have been proposed (including the ones in [5], [6], [7], [8], [9], [10], [11], [12]) *drop all* LO*-criticality tasks* upon the event of any job executing beyond its LO-criticality WCET estimate.

We would like to emphasize that from the verification perspective such job-dropping has no adverse implications for the LO-criticality workload: in verifying the correctness of the LO-criticality workloads, we assume that the LO-criticality WCET estimates are never exceeded (and hence such dropping will never occur). Therefore, in considering verification of mixed-criticality systems there is no particular benefit towards incorporating survivability properties into the run-time scheduling algorithms, whereas doing so tends to result in algorithms with more complex descriptions and run-time semantics. Consequently mixed-criticality scheduling algorithms have tended to be designed without incorporating survivability, and are emphatically *not* survivable from the perspective of the LO-criticality workloads: even the smallest violation of the assumption that LO-criticality WCET estimates will not be exceeded results in the entire LO-criticality workload being dropped. Much of the criticism that has been directed at mixed-criticality scheduling theory that is based on the Vestal model can be traced to these poor survivability properties of the proposed algorithms. Some efforts have recently been made at designing mixed-criticality scheduling algorithms that exhibit some forms of survivability (see, for example [13], [14], [15], [16]); however, these are all ad hoc approaches towards incorporating some properties that are desirable from the perspective of survivability and do not attempt to formally define and quantify survivability — to our knowledge, Burns et al. [17] represents a first effort at doing so. The focus in [17] is on defining task models that allow for the representation of robustness properties (in particular, HI-criticality tasks are defined as being robust to the dropping of individual jobs in the sense that the functionality of the task is not compromised upon such dropping, and algorithms are derived in [17] for scheduling systems comprising such tasks that judiciously select jobs to be dropped in order to not need to discard any LO-criticality tasks). We, in contrast, do not address the modeling aspect at all but instead explore scheduling-centric approaches for achieving survivability, by proposing a framework for the development scheduling algorithms and analyses that accommodate the concepts of robustness and resilience in a model-agnostic manner. Specifically, we seek to define quantitative metrics of robustness and resilience that are applicable to the Vestal model [1] (which, as previously stated, does not incorporate survivability in its current form), and to correlate these metrics to the resulting run-time survivability guarantees of the system.

III. System model

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 $C_i^L \leq C_i^H$. 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.

System *behaviors*. Since the period parameter of a sporadic task denotes the minimum (rather than exact) separation 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. The *criticality level* of a behavior is determined by how much execution is needed by the jobs in order to complete execution in that behavior:

- If every job completes upon executing for no more than the LO-criticality WCET estimate of the task that generated it, then the behavior is defined to be a LO-*criticality behavior*.

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 τ_i 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. 2. The run-time scheduling strategy used by Algorithm MC-Fluid

- Every behavior that is not a LO-criticality behavior in which every job completes upon executing for no more than the HI-criticality WCET estimate 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 job receives enough execution between its release time and deadline to complete execution, and
- during all HI-criticality behaviors of the system, all HI-criticality jobs receive enough execution between their release time and deadline to complete execution.

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.

Various system utilization parameters are defined for τ as follows:

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

A. Fluid scheduling of dual-criticality systems

The MC-Fluid scheduling algorithm [10], [12] 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 [10], [12] 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 2 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 [10]. It is shown in [10] 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 2, with values for the θ_i^L and θ_i^H parameters computed in the manner defined in [10], 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 [12]: 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 Figure 2, with values for the θ_i^L and θ_i^H parameters computed as in [10], will in fact successfully schedule τ upon an m-processor platform in which each processor is faster by a factor of $\frac{4}{3}$ (since $\frac{4}{3} < (1+\sqrt{5})/2$, this is a superior speedup bound). A somewhat simpler (and more efficient, in terms of run-time computational complexity) algorithm than the one in [10] for computing θ_i^L and the θ_i^H parameters was presented in [12] – this simpler algorithm, called Algorithm MCF, is presented in Figure 3 (specialized for uniprocessors – i.e., for $m \leftarrow 1$). It was also shown in [12] that the $\frac{4}{3}$'rds speedup bound holds even when the θ_i^L and the θ_i^H parameters are computed using Algorithm MCF.

IV. Incorporating Survivability into MCF

In this section we describe how quantitative notions of survivability — robustness and resilience — may be incorporated into the Vestal model, and how Algorithm MCF may be modified in order to provided specified degrees of robustness and resilience.[2]

From the perspective of survivability, it is perhaps helpful to interpret the WCET parameters of HI-criticality and LO-criticality tasks differently. We can look upon the WCET parameters of HI-criticality tasks as *assumptions*, and the

[2]We would like to point out that we are using Algorithm MCF as an examplar of our approach towards quantifying survivability primarily because these concepts appear easier to highlight for fluid scheduling models. Although these concepts can also be introduced in the context of other mixed-criticality scheduling algorithms that have been proposed such as AMC [6], EDF-VD [11], etc., that are not based on fluid-scheduling, discussing them with respect to these other algorithms requires that many orthogonal concepts also be dealt with; this obfuscates some of the survivability issues that we seek to highlight in this document.

1) Define ρ as follows:

$$\rho \leftarrow \max\Big\{U_L^L + U_H^L, U_H^H\Big\} \tag{1}$$

2) **If** $\rho > 1$ **then** declare failure; **else** assign values to the execution-rate variables as follows:

$$\theta_i^H \leftarrow u_i^H/\rho \text{ for all } \tau_i \in \tau_H \tag{2}$$

$$\theta_i^L \leftarrow \begin{cases} \frac{u_i^L \theta_i^H}{\theta_i^H - \left(u_i^H - u_i^L\right)}, & \text{if } \tau_i \in \tau_H \\ u_i^L, & \text{else (i.e., if } \tau_i \in \tau_L) \end{cases} \tag{3}$$

3) **If**

$$\sum_{\tau_i \in \tau} \theta_i^L \leq 1 \tag{4}$$

then declare success **else** declare failure

Fig. 3. Algorithm MCF

WCET parameters of LO-criticality tasks as the corresponding *guarantees*: if each HI-criticality job completes upon executing for no more than the LO-criticality WCET estimate of the task that generated it, then each LO-criticality job is guaranteed an execution of at least the LO-criticality WCET estimate 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 WCET estimate, we are able to *guarantee* each LO-criticality job an amount of execution up to its LO-criticality WCET estimate.

For incorporating survivability into the Vestal model it is equivalent, and perhaps more convenient, to think of a *server* of bandwidth/ capacity $u_i^L \stackrel{\text{def}}{=} (C_i^L/T_i)$ as being associated with each LO-criticality task τ_i: all the jobs that are generated by τ_i are executed by this server. (Hence if some such job has a WCET $> C_i^L$, the impact of this only falls upon future jobs that are generated by this task τ_i.) This is the approach we will take in this paper, associating a server with each LO-criticality task. Under conventional mixed-criticality scheduling (e.g., the run-time algorithm of Figure 2), these servers that are associated with the LO-criticality tasks are terminated if any job of a HI-criticality task executes for more than its LO-criticality WCET estimate. Survivability can now be defined in terms of the capacities assigned to these servers in the event of HI-criticality jobs executing beyond their LO-criticality WCET estimates:

- A *robust* scheduler would not change the capacities of these LO-criticality servers.
- A *resilient* scheduler would reduce the capacity of these LO-criticality servers rather than terminating them entirely; the quantitative metrics of resilience that we are proposing seek to quantify the degree of such reduction.

A running example. For ease of explanation we will introduce our ideas via illustration upon a simple task system that is depicted in Table I. As can be seen from Table I, this example task system comprises three tasks, of which two, τ_1 and τ_2, are of LO criticality and the third, τ_3, is a HI-criticality one. (The example has only one HI-criticality task by design, to better explain our proposals for quantitative measures of robustness and resilience; in Section IV-A, we will explain how to apply our concepts to task systems with > 1 HI-criticality task.)

	T_i	C_i^L	C_i^H	χ_i	u_i^L	u_i^H
τ_1	10	2	–	LO	0.2	–
τ_2	20	6	–	LO	0.3	–
τ_3	30	3	18	HI	0.1	0.6

$$\begin{aligned} U_L^L &= 0.2 + 0.3 = \mathbf{0.5} \\ U_H^L &= 0.1 \\ U_H^H &= 0.6 \end{aligned}$$

TABLE I
EXAMPLE TASK SYSTEM

The three system-utilization parameters U_L^L, U_L^H, and U_H^H for this example are also computed and presented in Table I. Recall that we are associating a server with each LO-criticality task: for this example, the servers associated with the LO-criticality tasks τ_1 and τ_2 would have bandwidths of $u_1^L \stackrel{\text{def}}{=} C_1^L/T_1$ and $u_2^L \stackrel{\text{def}}{=} C_2^L/T_2$ respectively; as can be seen from the table, these are equal to 0.2 and 0.3 respectively.

Algorithm MCF on the example. Applying Algorithm MCF (Figure 3) to the task system of Table I, we get

$$\rho \leftarrow \max\{0.5 + 0.1, 0.6\} = \mathbf{0.6}$$

Consequently, $\theta_3^H = 0.6/0.6$ or 1.0, and

$$\begin{aligned} \theta_1^L &= u_1^L = 0.2 \\ \theta_2^L &= u_2^L = 0.3 \\ \theta_3^L &= \frac{u_3^L \theta_3^H}{\theta_3^H - \left(u_3^H - u_3^L\right)} = \frac{0.1 \times 1}{1 - (0.6 - 0.1)} = \frac{0.1}{0.5} = 0.2 \end{aligned}$$

Since

$$\theta_1^L + \theta_2^L + \theta_3^L = 0.2 + 0.3 + 0.2$$

which is clearly ≤ 1, Algorithm MCF declares success: the run-time algorithm of Figure 2 is able to successfully schedule this system with these computed values for the θ_i^L and θ_i^H parameters.

Incorporating Robustness. Although our example is correctly scheduled by Algorithm MCF (as we have seen above), the system is not at all robust during run-time: if any job of task τ_3 executes beyond C_3^L (i.e., 3) time units, the run-time algorithm of Figure 2 immediately discards tasks τ_1 and τ_2. To incorporate robustness, we ask the question: "*What is the largest value to which we can increase C_3^L such that Algorithm MCF continues to declare success?*" We determine

this value below; once this is computed, we can enhance the robustness of the system by not discarding τ_1 and τ_2's jobs so long as jobs of τ_3 do not execute beyond this computed value (instead of doing so upon any of τ_3's jobs executing beyond 3 time units).

To compute the desired value, we first observe that Algorithm MCF assigns θ_3^H the value

$$u_3^H / \max(U_L^L + U_H^L, U_H^H) = 0.6/0.6 = \mathbf{1}$$

Next, we note that Algorithm MCF declares success as long as

$$\begin{aligned} & \theta_1^L + \theta_2^L + \theta_3^L \leq 1 \\ \Leftrightarrow \quad & u_1^L + u_2^L + \theta_3^L \leq 1 \\ \Leftrightarrow \quad & 0.2 + 0.3 + \theta_3^L \leq 1 \\ \Leftrightarrow \quad & \theta_3^L \leq 0.5 \end{aligned}$$

Hence it is not necessary to have $\theta_3^L = 0.2$ (as had been done by Algorithm MCF); we can in fact assign θ_3^L any value not exceeding 0.5. Let us therefore choose $\theta_3^L \leftarrow 0.5$; by Equation 3 in Figure 3, we have

$$\begin{aligned} & \theta_3^L = \frac{u_3^L \times \theta_3^H}{\theta_3^H - (u_3^H - u_3^L)} \leq 0.5 \\ \Leftrightarrow \quad & \theta_3^L = \frac{u_3^L \times 1.0}{1.0 - (0.6 - u_3^L)} \leq 0.5 \\ \Leftrightarrow \quad & u_3^L \leq 0.5 \times (0.4 + u_3^L) \\ \Leftrightarrow \quad & u_3^L \leq 0.4 \end{aligned}$$

The system would therefore remain correctly schedulable by Algorithm MCF as long as $u_3^L \leq 0.4$; equivalently. $C_3^L \leq 0.4 \times T_3 = 0.4 \times 30 = 12$. The system of Table I can therefore be scheduled in a robust manner by only terminating the servers associated with τ_1 and τ_2 only upon some job of τ_3 executing beyond 12 time units (rather than $C_3^L = 3$ time units). A reasonable quantitative metric of the robustness of this schedule is the ratio of these two quantities: $12/3$, or $\mathbf{4}$.

Achieving Resilience. Robustness refers to the ability of the system to provide full (i.e., non-degraded) service despite violation of assumptions; resilience, to the ability to provide some degraded level of service upon such violation.

Rather than providing robustness so long as τ_3's job's do not execute beyond 12 time units but no resilience upon their doing so, we could instead have chosen to provide a degraded level of service upon their execution time exceeding $C_3^L = 3$ (clearly this is only beneficial if we can continue to provide such degraded service upon these jobs executing beyond 12 time units).

We will see below that if we were to reduce the sum of the capacities of the servers associated with the LO-criticality tasks τ_1 and τ_2 to $3/8$ (i.e., 0.375) from 0.5 — a reduction to $\frac{3}{4}$ of the desired level of service — upon some job of τ_3 executing for beyond 3 time units, we would not need to degrade service to τ_1 and τ_2 any further as long as τ_3's jobs do not exceed their HI-criticality WCET estimate of 18 time units. This factor of $\frac{3}{4}$ may be considered a quantitative metric of the resilience of this schedule.

We now justify the claim in the paragraph above. First, we observe (as we had done previously, whilst incorporating robustness into our example) that since $\theta_1^L + \theta_2^L = u_1^L + u_2^L = 0.2 + 0.3 = 0.5$, we may assign the remaining processor capacity to θ_3^L (i.e., we may choose $\theta_3^L \leftarrow 0.5$). Suppose that a job of τ_3 does not complete execution despite having executed for its LO-criticality WCET estimate of 3 time units. Since $\theta_L^3 = 0.5$, this implies that this job has been executing for $3/0.5$ or 6 time units, which in turn implies that there is an interval of duration $(T_3 - 6) = (30 - 6) = 24$ time units remaining until its deadline. Assigning it a fraction $\theta_3^H \leftarrow \frac{5}{8}$ of the processor over this duration, we see that the total number of units of computation this job receives by its deadline is

$$6 \times \theta_3^L + 24 \times \theta_3^H = 6 \times 0.5 + 24 \times \frac{5}{8} = 3 + 15 = 18,$$

which is equal to its HI-criticality WCET estimate. Upon τ_3 being assigned $\theta_3^H = \frac{5}{8}$ of the processor, the remaining $(1 - \frac{5}{8}) = \frac{3}{8}$ of the processor capacity is assigned to the LO-criticality servers.

Both Robustness and Resilience. We now discuss how both robustness and resilience can be achieved simultaneously (in contrast to the two schemes described above, in each of which we had achieved one but not the other). Specifically, suppose that we are given a desired value for the robustness metric: what degree of resilience can be achieved? We present an illustrative example below.

Suppose we wish to ensure a robustness of 2: i.e., we seek robust behavior so long as jobs of τ_3 complete upon receiving no more than $2 \times C_3^L = 2 \times 3 = 6$ units of execution, and resilience thereafter.

As before, we choose $\theta_3^L \leftarrow 0.5$. That is, we start out assigning τ_3 half the processor capacity (and serving τ_1 and τ_2 with servers that have capacities 0.2 and 0.3 respectively).

If a job of τ_3 does not complete despite having received 6 units of execution,

- It must be the case that this job has executed for $6/0.5$ or 12 time units.
- Hence, there is an interval of duration $(T_3 - 12) = (30 - 12) = 18$ before this job's deadline.
- We assign τ_3's job a $\frac{2}{3}$'rds share of the processor over this interval, thereby enabling it to get $18 \times \frac{2}{3}$ or 12 units of execution by its deadline. (Note that this is enough to meet its HI-criticality WCET estimate C_3^H of 18 by its deadline.)
- The remaining $\frac{1}{3}$ of the processor is apportioned between the servers serving τ_1 and τ_2, with their processor shares reduced to $2/15$ and $1/5$ respectively (i.e., to $\frac{2}{3}$'rds of their desired levels of service). The resilience of this schedule is therefore $\frac{2}{3}$.

This analysis is easily extended to arbitrary values of the desired robustness. Suppose we desire a robustness of ρ_o. A job of τ_3 will execute for a duration $(C_3^L \times \rho_o)/0.5$ or 6ρ in order to have completed $C_3^L \times \rho_o = 3\rho_o$ units of execution.

In order to obtain the remaining $(C_3^H - 3\rho_o)$ or $(18 - 3\rho_o)$ units of execution by the deadline, we must have

$$\begin{aligned} & \theta_3^H \times (30 - 6\rho) = (18 - 3\rho_o) \\ \Leftrightarrow \quad & \theta_3^H = \frac{18 - 3\rho_o}{30 - 6\rho_o} = \frac{6 - \rho_o}{10 - 2\rho_o} \end{aligned}$$

The remaining $(1 - \theta_3^H)$ processor share is apportioned between the two LO-criticality servers; since they receive a $(1 - \theta_3^H)$ processor share rather than their desired share of $0.2 + 0.3 = 0.5$, the degradation in their service (and hence the resilience) is given by

$$\begin{aligned} & \left(1 - \theta_3^H\right) \div \frac{1}{2} \\ \equiv \quad & \left(1 - \frac{6 - \rho_o}{10 - 2\rho_o}\right) \times 2 \\ \equiv \quad & \left(\frac{4 - \rho_o}{10 - 2\rho_o}\right) \times 2 \\ \equiv \quad & \left(\frac{4 - \rho_o}{5 - \rho_o}\right) \end{aligned}$$

Hence for any selected value of $\rho_o \geq 1$, we are able to guarantee a resilience

$$\psi(\rho_o) = \left(\frac{4 - \rho_o}{5 - \rho_o}\right) \tag{5}$$

Since resilience can only take on non-negative values, it follows that ρ_o must be ≤ 4 — for $\rho_o = 5$, we have $\psi_o = 0$ — i.e., a non-resilient implementation with robustness factor 4, which was exactly the first implementation we had evaluated. Similarly choosing $\rho = 1$ yields a resilience of $\frac{3}{4}$, which is the second implementation — resilient but with no robustness — that we had considered.

More general formulations of resilience.. Let us continue with the example above: suppose that (as above) we continue to seek robustness so long as jobs of τ_3 complete upon receiving no more than 6 units of execution, but desire resilience only if these jobs complete upon receiving between 6 and 15 units of execution. If their execution exceeds 15 units, then no resilience is expected.

As before, start out with $\theta_3^L \leftarrow 0.5$. Upon executing for 12 time units, a job of τ_3 will have received 6 units of execution. If it does not complete,

- Assign it a 0.6 share of the processor. Over the next 15 time units (i.e., by $12 + 15 = 27$ time units of the job's arrival), it will have received $15 \times 0.6 = 9$ units of execution. Hence, it will have received a total of $6 + 9 = 15$ units of execution within 27 time units of its arrival.

  Since τ_3 receives a 0.6 share of the processor capacity, this leaves a 0.4 share to be apportioned between the servers that are servicing the LO-criticality tasks τ_1 and τ_2. The capacities assigned to these servers are therefore reduced to 0.8 of their desired capacities; hence this schedule has resilience 0.8.
- If the job of τ_3 has still not completed execution, assign it exclusive access to the processor for the remaining 3 time units, thereby ensuring that it receives $15+3 = 18$ units of execution by its deadline, thereby ensuring that it receives an amount of execution equal to its HI-criticality WCET estimate, C_3^H, by its deadline). The LO-criticality servers are suspended/ terminated if this happens; equivalently, the resilience is zero.

The situation described above may be quantified according to our robustness and resilience metrics by stating that with a robustness of 2 we guarantee a resilience of 0.8, while with a robustness of 5 we guarantee a resilience of 0 (i.e., no resilience).

A. Generalizing from the example

The running example that we have considered thus far has served to illustrate how we propose to incorporate quantitative metrics of robustness and resilience into the Vestal model; in this section, our proposed approaches are generalized to be applicable to systems with more than one HI-criticality task.

As previously stated, we are seeking to incorporate survivability into the mixed-criticality scheduling algorithm MCF, which is based on the fluid-scheduling paradigm. We modify Algorithm MCF for this purpose, incorporating servers for the execution of jobs from LO-criticality tasks.

Let us define an *implementation* of a system to comprise a given mixed-criticality task system τ, along with the particular algorithm that is used to schedule it during run-time. The correctness criterion for mixed-criticality scheduling requires that in any correct implementation the bandwidth of the server for each LO-criticality task τ_j is $\geq u_j^L$ as long as all jobs of each HI-criticality task τ_i completes upon executing for at most C_i^L units of execution. We now define robustness and resilience for correct implementations of the task system τ.

Robustness. A correct implementation of τ is said to have robustness ρ, where ρ is a real number ≥ 1, if the bandwidth of the server for each LO-criticality task τ_j remains at least u_j^L as long as no job of any HI-criticality task τ_i executes for more than $\rho \times C_i^L$ without signaling completion.[3]

For our example, we have shown that an implementation that chooses $\theta_3^L \leftarrow 0.5$ and only suspends/ terminates the LO-criticality servers upon some job of τ_3 executing for more than $4 \times C_3^L$ or 12 units of execution, has a robustness of 4.

Resilience. A correct implementation of τ is said to have resilience ψ, where ψ is a non-negative real number ≤ 1, if the bandwidth of the server for each LO-criticality task τ_j remains at least $\psi \times u_j^L$ as long as no job of any HI-criticality task τ_i executes for more than C_i^H without signaling completion.

For our example, we have shown that an implementation that chooses $\theta_3^L \leftarrow 0.5$ and $\theta_3^H \leftarrow \frac{5}{8}$, and reduces the bandwidth of each LO-criticality server by a factor $\frac{3}{4}$ upon

[3]Alternatively, it is reasonable to bound the execution of jobs of HI-criticality task τ_i at $\min(\rho \times C_i^L, C_i^H)$ — i.e., to assume that the HI-criticality WCET estimates represent truly safe upper bounds on the actual WCET values. With this interpretation, a *fully* robust implementation is defined as an implementation with the robustness parameter equal to infinity: $\rho = \infty$.

some job of τ_3 executing for more than $C_3^L = 3$ units of execution, has a resilience of $\frac{3}{4}$.

Both Robustness and Resilience. A correct implementation of τ is said to have **both** robustness ϕ and resilience ψ, if the bandwidth of the server for each LO-criticality task τ_j remains at least $\psi \times u_j^L$ as long as no job of any HI-criticality task τ_i executes for more than $\rho \times C_i^L$ without signaling completion.

We showed an implementation of our example system with robustness 2 and resilience $\frac{2}{3}$. We also derived a relationship – Expression 5 – between the robustness and resilience parameter values that are achievable for our example system.

Multiple (ρ, ψ) specifications. A natural generalization is to specify multiple ordered pairs $(\rho_1, \psi_1), (\rho_2, \psi_2), \ldots$ with $\rho_{k+1} > \rho_k$ and $\psi_{k+1} \leq \psi_k$. The interpretation is that for each k, we require that the server for each LO-criticality job τ_j have bandwidth $\geq \psi_k \times u_k^L$ as long as no job of any HI-criticality task τ_i executes for more than $\rho_k \times C_i^H$ without signaling completion.

We showed an implementation of our example system with $(\rho_1, \psi_1) = (2, 0.8)$ and $(\rho_2, \psi_2) = (5, 0)$. That is, this implementation

- guarantees full service to the LO-criticality workload as long as no HI-criticality job executes for more than twice its LO-criticality WCET estimate;
- guarantees 0.8 of the desired level of service to the LO-criticality workload as long as each HI-criticality job executes for between twice and five times its LO-criticality WCET estimate; and
- makes no guarantees to the LO-criticality workload if any HI-criticality job executes for more than its LO-criticality WCET estimate.

(Correct execution of all HI-criticality jobs is assured, provided each completes upon executing for up to its HI-criti ality WCET estimate.)

We have seen above that multiple different implementations of a single system are possible, with the different implementations characterized by different robustness and resilience parameters. Many pairs of such implementations are *incomparable* in the sense that one offers greater robustness and the other, greater resilience. There is in general a potentially infinite design space of possible such incomparable implementations — choosing the most appropriate implementation for a specific system is a design choice that should be guided by the intended use of the system.

V. Conclusions and Future Work

Pre-runtime verification and run-time survivability are two distinct aspects of correctness in safety-critical systems. Mixed-criticality scheduling theory (MCSh) has, thus far, focused almost exclusively on the verification aspect; in this document we have described some of our ongoing efforts at extending MCSh to incorporate survivability considerations. We have proposed quantitative metrics of both aspects of survivability – robustness and resilience – for mixed-critical task systems that are represented using the Vestal model [1]. While we have illustrated the applicability of our proposed metrics by using them as the basis for the development of survivable implementations of a simple mixed-critical system under a particular mixed-criticality scheduling algorithm (Algorithm MCF), we are not claiming that our quantitative metrics are the only ones (or even the best ones) that can be defined. We believe the choice of metrics is an inherently social process in that buy-in from a larger research community is needed if the metrics are to see much use – we hope that this document will spur some discussion on the choice of metrics for robustness and resilience, and perhaps yield alternative proposals for metrics.

As stated in Section IV (footnote 2), we have chosen to illustrate the applicability of our quantitative metrics on the mixed-criticality scheduling algorithmMCF primarily for reasons of simplicity: we are by no means suggesting that we believe it to be the definitive mixed-criticality scheduling algorithm. As future work we plan to subject other mixed-critical scheduling algorithms that have been proposed (such as AMC [6], EDF-VD [11], etc.) to the same form of analysis as we have done here with Algorithm MCF, and thereby develop survivable implementations of systems that are based upon these non-fluid mixed-criticality scheduling algorithms.

Also as future work, we plan to revisit some mixed-criticality scheduling algorithms such as the ones in [13], [14], [15], [16] that have previously been proposed for addressing the non-survivability of traditional mixed-criticality scheduling algorithms. We will seek to characterize the robustness and resilience properties of these algorithms on the basis of the metrics that we have proposed in this paper.

Acknowledgements

This research has been supported in part by NSF grants CNS 1409175 and CNS 1814739, and EPSRC grant MCCps (EP/P003664/1).

References

[1] Steve 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.

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

[3] Alexandre Esper, Geoffrey Nelissen, Vincent Nélis, and Eduardo Tovar. How realistic is the mixed-criticality real-time system model? In *Proceedings of the 23rd International Conference on Real Time and Networks Systems*, RTNS '15, pages 139–148, New York, NY, USA, 2015. ACM.

[4] Rolf Ernst and Marco Di Natale. Mixed criticality systems - A history of misconceptions? *IEEE Design & Test*, 33(5):65–74, 2016.

[5] Sanjoy K. Baruah, Vincenzo Bonifaci, Gianlorenzo D'Angelo, Haohan Li, Alberto Marchetti-Spaccamela, Nicole Megow, and Leen Stougie. Scheduling real-time mixed-criticality jobs. In Petr Hlinený and Antonín Kucera, editors, *Proceedings of the 35th International Symposium on the Mathematical Foundations of Computer Science*, volume 6281 of *Lecture Notes in Computer Science*, pages 90–101. Springer, 2010.

[6] Sanjoy Baruah, Alan Burns, and Robert 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.

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

[8] Sanjoy K. Baruah, Vincenzo Bonifaci, Gianlorenzo D'Angelo, Haohan Li, Alberto Marchetti-Spaccamela, Nicole Megow, and Leen Stougie. Scheduling real-time mixed-criticality jobs. *IEEE Transactions on Computers*, 61(8):1140–1152, 2012.

[9] 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.

[10] Jaewoo Lee, Kieu-My Phan, Xiaozhe Gu, Jiyeon Lee, A. Easwaran, Insik Shin, and Insup Lee. MC-Fluid: Fluid model-based mixed-criticality scheduling on multiprocessors. In *Real-Time Systems Symposium (RTSS), 2014 IEEE*, pages 41–52, Dec 2014.

[11] Sanjoy Baruah, Vincenzo Bonifaci, Gianlorenzo D'angelo, Haohan Li, Alberto Marchetti-Spaccamela, Suzanne Van Der Ster, and Leen Stougie. Preemptive uniprocessor scheduling of mixed-criticality sporadic task systems. *Journal of the ACM*, 62(2):14:1–14:33, May 2015.

[12] Sanjoy Baruah, Arvind Easwaran, and Zhishan Guo. MC-Fluid: simplified and optimally quantified. In *Real-Time Systems Symposium (RTSS), 2015 IEEE*, Dec 2015.

[13] Sanjoy Baruah an Alan Burns and Zhishan Guo. Scheduling mixed-criticality systems to guarantee some service under all non-erroneous behaviors. In *Proceedings of the 2016 28th EuroMicro Conference on Real-Time Systems*, ECRTS '16, Toulouse (France), 2016. IEEE Computer Society Press.

[14] D. Liu, J. Spasic, N. Guan, G. Chen, S. Liu, T. Stefanov, and W. Yi. Edf-vd scheduling of mixed-criticality systems with degraded quality guarantees. In *2016 IEEE Real-Time Systems Symposium (RTSS)*, pages 35–46, Nov 2016.

[15] Xiaozhe Gu and Arvind Easwaran. Dynamic budget management with service guarantees for mixed-criticality systems. In *2016 IEEE Real-Time Systems Symposium, RTSS 2016, Porto, Portugal, November 29 - December 2, 2016*, pages 47–56, 2016.

[16] Saravanan Ramanathan and Arvind Easwaran. Mixed-criticality scheduling on multiprocessors with service guarantees. In *21st IEEE International Symposium on Real-Time Distributed Computing, ISORC 2018, Singapore, Singapore, May 29-31, 2018*, pages 17–24, 2018.

[17] Alan Burns, Robert I. Davis, Sanjoy Baruah, and Iain Bate. Robust mixed-criticality systems. *IEEE Transactions on Computers*, 67(10):1478–1491, 10 2018.