Schedulability analysis of sporadic tasks with multiple criticality specifications

Sanjoy Baruah[*]
The University of North Carolina

Steve Vestal
Honeywell Labs

Abstract

In a paper that was presented at the recently-concluded Real-Time Systems Symposium, Vestal proposed a new real-time task model that is able to represent the fact that the worst-case execution time (WCET) of a single task may be determined to different levels of accuracy with different degrees of confidence. In systems with multiple criticality requirements — different tasks need to be assured of meeting their deadlines with different levels of confidence — such multiple specifications of WCET may be exploited to obtain better processor utilization.

This paper conducts a thorough study of the feasibility and schedulability questions for such multi-criticality real-time task systems when implemented upon preemptive uniprocessor platforms.

Keywords. Safety-critical systems; Sporadic task systems; Feasibility analysis; Schedulability analysis; Hybrid-priority scheduling.

1 Introduction

In a recent paper [9], Vestal presented an interesting, and practically significant, formal model for representing real-time tasks. Based on the observation that "the more confidence one needs in a task execution time bound, the larger and more conservative that bound tends to be in practice," several different worst-case execution time (WCET) parameters may be specified for each task in this model, each such parameter being determined at a different level of assurance or *criticality*. For example the largest execution time observed during tests of normal operating mode scenarios can be specified as the WCET at a low level of assurance; the largest execution time observed during more exhaustive "code-coverage" tests are more appropriate as the WCET at a higher level of assurance; while at the highest levels of assurance code flow analysis and worst-case instruction cycle counting may need to be done in order to find suitable bounds.

Different tasks in the system may, in general, perform tasks of different levels of criticality and hence need to be guaranteed to meet their timing constraints to different levels of assurance. For instance, the RTCA DO-178B software standard specifies several different criticality levels, with the system designer expected to assign one of these criticality levels to each task — Table 1 lists the criticality levels, and intended interpretations, that are specified in this standard.

The criticality levels assigned to the different tasks must be taken into account during system validation and testing. The intended interpretation is as follows: As in any hard-real-time system, each task must be guaranteed to meet all its deadlines; furthermore, *all the WCET values that have an impact on whether a task meets its deadline or not must be of the same criticality level as that required by the task*.

This research. Vestal's multi-criticality sporadic task model represents a potentially very significant advance in the modeling of safety-critical real-time systems. Accordingly, it is appropriate that a thorough analysis of the scheduling-theoretic issues related to this model be conducted; in particular, the large body of research into the traditional sporadic task model should be revisited in order to determine which parts apply to this more general model as well, and which parts do not. This paper represents an attempt at initiating such a study in the context of preemptive uniprocessor systems of independent tasks. Fundamental scheduling-theoretic issues – feasibility; schedulability; expressiveness; etc. – are revisited for this model. Exact feasibility conditions are derived. The relative abilities of different kinds of scheduling algorithms (EDF, fixed-priority, etc.) in scheduling such task systems are determined, and a scheduling algorithm proposed that generalizes the algo-

[*]Supported in part by NSF Grant Nos. CNS-0408996, CCF-0541056, and CCR-0615197, ARO Grant No. W911NF-06-1-0425, and funding from IBM and the Intel Corporation.

Level	Failure Condition	Interpretation
A	Catastrophic	Failure may cause a crash
B	Hazardous	Failure has a large negative impact on safety or performance, or reduces the ability of the crew to operate the plane due to physical distress or a higher workload, or causes serious or fatal injuries among the passengers
C	Major	Failure is significant, but has a lesser impact than a Hazardous failure (for example, leads to passenger discomfort rather than injuries)
D	Minor	Failure is noticeable, but has a lesser impact than a Major failure (for example, causing passenger inconvenience or a routine flight plan change)
E	No Effect	Failure has no impact on safety, aircraft operation, or crew workload.

Table 1. DO-178B is a software development process standard, *Software Considerations in Airborne Systems and Equipment Certification*, published by RTCA, Incorporated. The United States Federal Aviation Authority (FAA) accepts the use of DO-178B as a means of certifying software in avionics applications. RTCA DO-178B assigns criticality levels to tasks, categorized by effects on commercial aircraft.

rithm proposed by Vestal in [9].

Organization. The remainder of this paper is organized as follows. The multi-criticality sporadic task model is formally defined in Section 2. Feasibility analysis of multi-criticality sporadic task systems is studied in Section 3, and schedulability by different classes of scheduling algorithms is discussed in Section 4. A particular scheduling algorithm that generalizes the algorithm proposed by Vestal [9] is derived and evaluated in Section 5.

2 Task model

Although the RTCA DO-178B standard specifies just five criticality levels A-E, there is no particular reason why systems should be required to have at most five levels. Accordingly, we consider here a system model in which there are arbitrarily many distinct criticality levels, denoted by the positive integers, with larger integers denoting greater criticality. (The RTCA DO-178B criticality level A would thus be mapped to 5, and level E to 1.)

We consider the preemptive, uniprocessor scheduling of multi-criticality sporadic task systems. Such a multi-criticality sporadic task system τ is comprised of n multi-criticality sporadic tasks $\tau_1, \ldots, \tau_n$. Each task τ_i is characterized by the following parameters

- A WCET function $C_i : \mathbf{N}^+ \to \mathbf{R}^+$, specifying the WCET's for different criticality levels: the WCET for criticality level ℓ is equal to $C_i(\ell)$. Without any loss of generality, we assume that $C_i(\ell) \leq C_i(\ell + 1)$ for all i and all ℓ.
- A relative deadline parameter D_i.
- A minimum inter-arrival separation or period parameter T_i.
- A criticality level L_i, $L_i \in \mathbf{N}^+$.

The intended interpretation of these parameters is similar to that for standard (non multi-criticality) sporadic task systems [8]. That is, each task τ_i generates a potentially infinite sequence of jobs, with successive job arrivals separated by at least T_i time units and with each job's deadline being D_i time-units after its arrival time. The WCET of any job of τ_i, at criticality level ℓ, is $C_i(\ell)$. We restrict ourselves in this paper to systems of independent tasks – it is assumed that the different tasks in τ do not interact in any manner other than executing upon a shared preemptive processor.

Observe that the multi-criticality sporadic task model is a strict generalization of the traditional sporadic task model, since a traditional sporadic task system can be specified in this model by specifying exactly the same criticality level for all tasks ($L_i = L_j$ $(\forall i, j)$).

The multi-criticality sporadic task model turns out to be a remarkably powerful and expressive model. For instance, observe that it allows for the modeling of tasks τ_i for which WCET estimates at criticality levels greater than L_i are unknown, by setting these values equal to ∞. This is important – in order to obtain WCET bounds at high levels of assurance (i.e., at high criticalities), it is typically necessary to limit the kinds of programming constructs that may be used in implementing the task (for instance, loops may be required to have static bounds). But low-criticality tasks should not be subject to these same stringent programming restrictions; by allowing the corresponding (high-assurance) WCET's for these low-criticality tasks to be set equal to ∞, we are not forbidding low-criticality tasks from using these "unsafe" constructs. (Of course, care must be taken in scheduling systems containing such tasks to ensure that no job of some task of any criticality ℓ is ever assigned lower priority than a job that has its level-ℓ WCET equal to ∞.)

Several notions from the analysis of traditional sporadic task systems, including feasibility and schedulability, and the synchronous arrival sequence, generalize in a straightforward manner to multi-criticality sporadic task systems:

Definition 1 (Feasibility and schedulability) *A multi-criticality sporadic task system τ is said to be* schedulable by a scheduling algorithm A *if algorithm A will always meet all deadlines of τ to the desired level of assurance. τ is said to be* feasible *if there exists some scheduling algorithm A such that τ is schedulable by A.*

Definition 2 (synchronous arrival sequence (SAS)) *The* synchronous arrival sequence of jobs *for a sporadic task τ_i consists of the first job of τ_i arriving at time-instant zero, and subsequent jobs arriving exactly T_i time units apart. The synchronous arrival sequence for a collection of tasks consists of the union of the synchronous arrival sequences of each individual task in the collection of tasks.*

3 Feasibility analysis

Vestal [9] considered the scheduling of multi-criticality sporadic task systems using the restricted class of scheduling algorithms called *fixed priority* algorithms (discussed below in Section 4 under the name Fixed Task Priority, or FTP, algorithms). If no such restrictions are placed upon the scheduling algorithms that may be used, it is straightforward to show that the feasibility analysis problem for multi-criticality sporadic task systems is equivalent to the feasibility analysis problem for traditional sporadic task systems. This is formalized in Theorem 1 below. But first, a definition:

Definition 3 *For any multi-criticality sporadic task system τ, we define the* **corresponding traditional sporadic task system** *to be*

$$\bigcup_{\tau_i \in \tau} \Big\{ \Big(C_i(L_i), D_i, T_i \Big) \Big\}$$

— here, every traditional sporadic task system is represented by the 3-tuple (WCET, relative deadline, period). ■

Theorem 1 *Multi-criticality sporadic task system τ is feasible if and only if the corresponding traditional sporadic task system is feasible.*

Proof Sketch: Consider a scheduling algorithm that is able to enforce *temporal isolation* (see Section 4) between different jobs, allowing each job to execute only for a predetermined amount of time. Such an algorithm could be made to execute each job of τ_i for at most $C_i(L_i)$ time units, thereby essentially ignoring the multiple specifications of the WCET and treating each multi-criticality sporadic task as a traditional sporadic task. ■

Since algorithms for determining the feasibility of traditional sporadic task systems are known (see, e.g. [4]), Theorem 1 immediately yields algorithms for determining the feasibility of multi-criticality sporadic task systems.

4 Schedulability analysis

Theorem 1 above illustrates that from the perspective of feasibility analysis, multi-criticality sporadic task systems are identical to traditional sporadic task systems. Nevertheless as Vestal's paper [9] illustrates, there are many interesting unresolved issues concerning the scheduling of multi-criticality sporadic task systems; essentially, these issues arise from the fact that it is *schedulability by specific algorithms*, rather than feasibility (i.e., schedulability by some hypothetical optimal algorithm), that is the important analysis question with respect to the implementation of actual safety-critical application systems. In this section, we delve deeper into schedulability analysis of multi-criticality sporadic task systems; we will see that when restrictions are placed upon the kinds of scheduling algorithms that may be used, multi-criticality sporadic task systems are very different from traditional sporadic task systems.

§1. A classification of scheduling algorithms. Uniprocessor run-time scheduling algorithms operate as follows. At each instant they (implicitly or explicitly) assign a priority to each job that is awaiting execution, and choose for execution the greatest-priority waiting job. Within the context of such priority based scheduling algorithms, the semantic interpretation of multi-criticality may be stated in this manner: *in determining whether a task meets its deadline, the WCET values of all jobs that could be directly or transitively prioritized over a job of this task, and thereby impact this job's execution, must be of the same level of assurance as the assurance required by the task.*

Depending upon the restrictions that are placed upon the manner in which priorities may be assigned to jobs, we distinguish [5] between three classes of algorithms for scheduling sporadic task systems:

1. *Fixed task-priority (FTP)* scheduling: All the jobs generated by a given task are assigned the same priority.

2. *Fixed job-priority (FJP)* scheduling: Different jobs of the same task may have different priorities. However, the priority of each job may not change between its arrival time and its completion time.

3. *Dynamic priority (DP)* scheduling: Priorities of jobs may change between their release times and their completion times.

It is evident from the definitions that FJP scheduling is a generalization of FTP scheduling, and DP scheduling is a generalization of FJP scheduling. In the uniprocessor scheduling of traditional (i.e., not multi-criticality) task systems, the FJP scheduling algorithm Earliest Deadline First (EDF) is known to be *optimal* in the sense that EDF always meets all deadlines for all feasible traditional sporadic task system; hence, DP scheduling algorithms (which typically incur greater run-time implementation overhead) are not commonly used in scheduling such task systems.

In his initial work [9] introducing multi-criticality systems, Vestal restricted himself to FTP scheduling algorithms. In this research, we wish to consider the more general FJP and DP algorithms as well.

§**2. DP scheduling.** Since the class of DP algorithms includes all scheduling algorithms, all feasible multi-criticality sporadic task systems are, by definition, schedulable by some DP scheduling algorithm. Consequently, a multi-criticality sporadic task system τ is DP-schedulable if and only if the corresponding traditional sporadic task system (see Definition 3) is feasible, and *the test of Theorem 1 also tests for DP-schedulability*.

§**3. Limitations of FJP and FTP scheduling.** We now turn our attention to FJP and FTP scheduling. The first question to ask here is: what (if any) features do DP algorithms possess that FJP and FTP algorithms do not, that may affect their ability to schedule multi-criticality sporadic task systems?

Some schedulers are able to enforce *temporal isolation* between different jobs at run-time. That is, each job is assigned a maximum amount of execution, and the scheduler is able to ensure that no job exceeds the amount of execution assigned it. In such systems no job depends upon the scheduling of another, and hence a job's completion is not impacted by the accuracy of the execution time estimates of other jobs. Examples include static table-driven or time-triggered schedulers (in which individual jobs are only allowed to execute for predetermined amounts of time) and real-time variants of weighted fair-queueing.

Now, any algorithm that implements temporal isolation between jobs can be implemented within the framework of a DP priority-driven scheduling algorithm by simply raising and lowering the priorities of jobs at the appropriate instants. However, this cannot be done in FJP and FTP scheduling algorithms: *FJP and FTP algorithms cannot by themselves guarantee temporal isolation among jobs*[1].

Does this inability to provide temporal inter-job isolation cost us anything in terms of schedulability? As stated above, in traditional sporadic task systems it is the case that all feasible task systems are also schedulable using some FJP scheduling algorithm (specifically, EDF). The following example illustrates that this is not the case with multi-criticality sporadic task systems:

Example 1 Consider the task system comprised of the two tasks τ_1 and τ_2, with the following parameters:

$$\begin{array}{llll} C_1(1) = 5, & C_1(2) = 5, & D_1 = T_1 = 6, & L_1 = 2, \\ C_2(1) = 0.5, & C_2(2) = 5, & D_2 = T_2 = 5, & L_2 = 1, \end{array}$$

By considering both possibilities – (i) τ_1 has the greater priority, and (ii) τ_2 has the greater priority — it may be verified that this system is not FTP-schedulable. Neither is it FJP schedulable, as is seen by the following argument. Let us consider the first job of each task when scheduling the SAS, and consider both possibilities: τ_1's first job has greater priority than τ_2's first job, or vice versa.

1. *When τ_1's first job has greater priority than τ_2's first job:* In this case, τ_2's first job cannot be guaranteed to meet its deadline with assurance level 1: since $C_1(2) = 5$, τ_1's first job would execute over the interval $[0, 5)$, thereby allowing τ_2's job no execution at all.

2. *When τ_2's first job has greater priority than τ_1's first job:* In this case, τ_1's first job cannot be guaranteed

[1]This is not to claim that FJP and FTP schedulers cannot be enhanced to provide inter-job temporal isolation – indeed, that is exactly what the various *servers* and *open environments* built around EDF and FTP schedulers do. But all such servers have additional "budgeting" components which are responsible for ensuring that each job observes the preset limits on its execution time.

to meet its deadline with assurance level 2: since $C_2(2) = 5$, τ_2's first job would execute over the interval $[0, 5)$, and leave only one unit of execution for τ_1's job before its deadline.

However, since $C_1(L_1)/T_1 + C_2(L_2)/T_2 = 5/6 + 0.5/5 < 1$, this system is feasible by the result in Theorem 1 and consequently DP-schedulable. ■

The result in Example 1 is formalized in the following theorem.

Theorem 2 *There are multi-criticality sporadic task systems schedulable using DP scheduling algorithms, that cannot be scheduled using any FTP or FJP scheduling algorithms.*■

§**4. Comparing FTP and EDF.** For traditional sporadic task systems, it is known that EDF *dominates* FTP scheduling from the perspective of schedulability – all task systems schedulable by FTP scheduling are also schedulable by EDF, while there are task systems schedulable using EDF for which no FTP schedule can exist. (This is a direct consequence of the optimality of EDF, and the fact that no FTP algorithm is optimal in the sense of being able to schedule all feasible sporadic task systems.) However, this dominance does not carry over to multi-criticality sporadic task systems, as illustrated by the following simple example.

Example 2 Consider the task system comprised of the two tasks τ_1 and τ_2, with the following parameters:

$$\begin{array}{llll} C_1(1) = 2, & C_1(2) = 2, & D_1 = T_1 = 4, & L_1 = 2, \\ C_2(1) = 2, & C_2(2) = 5, & D_2 = T_2 = 7, & L_2 = 1. \end{array}$$

Thus, τ_1 is a higher-criticality task. By assigning τ_1 greater priority than τ_2, it may be verified that both tasks meet their deadlines at the desired levels of criticality; hence, the system is FTP-schedulable.

Now consider EDF, and let us focus on the second job of τ_1 in the SAS. Recall that the semantics of the multi-criticality task model require that all the WCET values used in ensuring that a task meets its deadline be of the same level of assurance as that required by the task. The jobs that have greater priority than τ_1's second job are τ_2's first job and, by transitivity, τ_1's first job. The WCET's $C_1(2)$ and $C_2(2)$ must be used to determine whether τ_1 meets its deadlines. But in simulating EDF on the SAS with these WCET estimates, it is easy to see that τ_1's second job misses its deadline at time-instant 8 (since τ_1's first job executes over $[0, 2)$, and τ_2's job over $[2, 7)$, thereby leaving just one unit of execution, rather than the required two units, for τ_1's second job before its deadline). ■

Example 2, in conjunction with the fact that there are EDF-schedulable multi-criticality sporadic task systems that are not FTP-schedulable (any feasible traditional sporadic task system that is not FTP-schedulable is an example), immediately yields the following result:

Theorem 3 *FTP and EDF are* incomparable *scheduling strategies in the scheduling of multi-criticality sporadic task systems.*■

§**5. The Vestal algorithm [9].** Intuitively, the result of Theorem 3 is not particularly surprising — since any task's job may get prioritized over any other task's job in EDF scheduling, all the WCET values used must be of the level of assurance required by the task with the greatest criticality level. Hence, systems with low-criticality tasks for which high-criticality WCET estimates are overly pessimistic are highly unlikely to be deemed EDF-schedulable since the overly pessimistic WCET estimate adversely impacts the likelihood that the high-criticality tasks will meet their deadlines at their desired level of assurance.

To avoid this phenomenon, one possible strategy is to prioritize tasks according to their criticality levels – the greater the criticality of a task, the higher the (fixed) priority assigned to this task. However, Vestal showed [9] that this algorithm is provably non-optimal even among fixed-priority algorithms; neither is deadline-monotonic (DM) priority assignment [6] (even for those task systems – $D_i \leq T_i \forall i$ – for which DM is an optimal FTP strategy for regular sporadic task systems). Vestal [9] proposed a priority-assignment algorithm based upon the priority assignment strategy of Audsley [1], and proved that *this algorithm is optimal within the class of FTP scheduling algorithms*: if a multi-criticality sporadic task system is schedulable using any FTP algorithm, then it is schedulable using the Vestal priority-assignment scheme.

Although the Vestal algorithm is an optimal FTP scheme, it is by no means optimal if we are not required to restrict ourself to FTP algorithms only. Indeed, since multi-criticality sporadic task systems in which all tasks happen to have the same criticality level are essentially equivalent to regular sporadic task systems, EDF would dominate the Vestal algorithm in the scheduling of such multi-criticality sporadic task systems.

5 Hybrid-priority scheduling

As discussed in Theorem 1 above, the Vestal algorithm (which yields an optimal FTP priority assignment) and EDF are incomparable when it comes to multi-criticality sporadic task systems — there are multi-criticality sporadic task systems schedulable by each algorithm that the other fails to schedule. Our objective in this section is to obtain a scheduling algorithm that generalizes both EDF and the Vestal algorithm, and is provably superior to both in the scheduling of multi-criticality sporadic task systems. To this end, we propose to explore the use of *hybrid* scheduling policies[2], which incorporate features of both FTP scheduling and EDF. In order to schedule multi-criticality sporadic task system τ using such a hybrid scheduling policy, we must assign each task in τ a (not necessarily unique) priority. These priorities are, by definition, totally ordered with respect to each other: as long as jobs in one priority are awaiting execution during run-time, no lower-priority jobs may execute. Within each priority, tasks will be scheduled using EDF.

An algorithm for performing schedulability analysis of (traditional – not multi-criticality) sporadic task systems under such a hybrid scheduling policy is presented in [2]. There, it is shown that the synchronous arrival sequence (SAS) represents the worst-case arrival sequence — if the SAS of a sporadic task system is scheduled to meet all deadlines using hybrid-priority scheduling for a given priority assignment, then the sporadic task system will always meet all deadlines under hybrid-priority scheduling with the same priority assignment. (This result for hybrid-priority systems generalizes the previously-known results that the SAS represents the worst-case arrival sequence for FP [7, 6] and EDF [3] scheduling as well.) In Section 5.1 below, we will use this observation as the basis upon which to design a priority-assignment algorithm for the hybrid-priority scheduling of multi-criticality sporadic task systems.

5.1 Algorithm description

Our priority assignment algorithm, ASSIGNPRIORITIES(τ), is presented in pseudo-code form in Figure 1. It is a generalization of the Audsley algorithm [1] for assigning priorities in FTP-scheduled systems. The procedure AUGMENTEDAUDSLEY(τ_{cur}, p) accepts as input a set of tasks $\tau_{\text{cur}} \subseteq \tau$ and a priority p, and determines which of the tasks in τ_{cur} can be assigned priority p, and which must be assigned greater priority, in order to meet their timing constraints at their desired levels of assurance. At each step in AUGMENTEDAUDSLEY during consideration of the current priority level p, τ_{cur} retains the set of tasks that have not been ruled out for being assigned priority p and τ_{hi} contains the set of tasks that have been identified as needing to be assigned some greater priority. (It is assumed that all the tasks in $(\tau \setminus (\tau_{\text{cur}} \bigcup \tau_{\text{hi}}))$ have already been assigned some priority $< p$, prior to entering this call to AUGMENTEDAUDSLEY.) The initial call to AUGMENTEDAUDSLEY (from ASSIGNPRIORITIES(τ)) thus starts with the lowest priority and all the tasks (i.e., $p \leftarrow 1$, and $\tau_{\text{cur}} \leftarrow \tau$).

During the execution of AUGMENTEDAUDSLEY, any task that is identified as not being guaranteed to meet its timing constraints at the current priority level must be assigned greater priority — such tasks are removed from τ_{cur} and added to τ_{hi}. In order to identify tasks that cannot meet their timing constraints at the current priority level, we simulate the hybrid-priority scheduling of the the SAS of $(\tau_{\text{cur}} \bigcup \tau_{\text{hi}})$ (as Audsley pointed out [1], tasks that have already been assigned priority $< p$ need not be considered, since they do not influence the scheduling of tasks with priority p or greater). More specifically, let $\pi_1, \pi_2, \ldots, \pi_\ell$ denote the different criticality levels of tasks in τ_{cur}, sorted in decreasing order. AUGMENTEDAUDSLEY simulates the hybrid-priority scheduling of the the SAS of $(\tau_{\text{cur}} \bigcup \tau_{\text{hi}})$ using first the $C_i(\pi_1)$'s as WCET's for the tasks; if all deadlines of jobs of the tasks in τ_{cur} with criticality level π_1 are met, it then simulates the hybrid-priority scheduling of the the SAS of $(\tau_{\text{cur}} \bigcup \tau_{\text{hi}})$ using the $C_i(\pi_2)$'s as WCET's for the tasks; if all deadlines of jobs of the tasks in τ_{cur} with criticality level π_2 are met, it next simulates the hybrid-priority scheduling of the the SAS of $(\tau_{\text{cur}} \bigcup \tau_{\text{hi}})$ using the $C_i(\pi_3)$'s as WCET's for the tasks; and so on.

If all deadlines of jobs of the tasks in τ_{cur} with the lowest criticality level (π_ℓ, in the pseudo-code) are met when the $C_i(\pi_\ell)$'s are used as WCET estimates, then we are done – the system can be scheduled to the desired degree of assurance when all tasks in τ_{cur} are assigned the current priority level p (this is represented by the return statement in Line 8 of the pseudo-code). Suppose, however, that some job of a task in τ_{cur} with criticality level π_j misses its deadline during the simulation using the $C_i(\pi_j)$'s as WCET's for the tasks. Let τ_k denote the task with criticality π_j generating the first such job to miss its deadline, and suppose that this deadline miss occurs at time-instant t_f. In order for this job to meet its deadline, it is necessary that it have a *greater* priority than some earlier-deadline job of some task in τ_{cur}

```
augmentedAudsley($\tau_{\rm cur}, p$)
    ▷ Assigns priorities $p$ or greater to all the tasks in $\tau_{\rm cur}$, such that all these tasks are deemed schedulable at
    ▷ their specified levels of assurance, if possible.
1           $\tau_{\rm hi} \leftarrow \emptyset$ ▷ Tasks in $\tau_{\rm hi}$ will be assigned priority $> p$.
            Let $\pi_1, \pi_2, \ldots, \pi_\ell$ denote the distinct criticality levels of tasks in $\tau_{\rm cur}$, in decreasing order.
2           for $j \leftarrow 1$ to $\ell$ do
3               while (a job of some task in $\tau_{\rm cur}$ of criticality $\pi_j$ misses a deadline in the simulation of the hybrid-priority
                scheduling of the SAS of $\tau_{\rm cur} \cup \tau_{\rm hi}$, using level-$\pi_j$ WCET's) do
4                   Let $\tau_k$ denote the task generating the criticality-$\pi_j$ job that first misses a deadline
5                   $\tau_{\rm hi} \leftarrow \tau_{\rm hi} \cup \{\tau_k\}$; $\tau_{\rm cur} \leftarrow \tau_{\rm cur} \setminus \{\tau_k\}$ ▷ $\tau_k$ gets "promoted" to a higher priority
                end while (line 3)
            end for (line 2)
6           if $\tau_{\rm cur} = \emptyset$ then return "could not schedule" end if ▷ All the tasks needed to be promoted.
7           All tasks in $\tau_{\rm cur}$ are assigned to the priority level $p$
8           if $\tau_{\rm hi} = \emptyset$ then return "successfully scheduled" end if ▷ all tasks have been assigned priorities.
            ▷ Now, recursively assign priorities to all the tasks that were "promoted."
9           augmentedAudsley($\tau_{\rm hi}, p+1$)
    end augmentedAudsley

assignPriorities($\tau$)
1           augmentedAudsley($\tau, 1$)
    end assignPriorities
```

Figure 1. Hybrid-priority scheduling of multi-criticality sporadic task systems: priority-assignment.

assigned priority p in the SAS. Within our hybrid-priority framework, the only way this can be achieved is by assigning τ_k a greater (fixed) priority than some other task in $\tau_{\rm cur}$; consequently, we conclude that τ_k must be assigned a priority that is greater than the lowest priority in the system, and move it from $\tau_{\rm cur}$ to $\tau_{\rm hi}$ (Line 5 of the pseudo-code).

Having assigned τ_k a greater priority in this manner, augmentedAudsley must re-examine the entire system once again (the while loop in Lines 3-5). That is, it re-simulates the scheduling of the SAS at each criticality level $\pi_1, \pi_2, \ldots$, except that τ_k is now assigned a greater priority than the tasks remaining in $\tau_{\rm cur}$. In doing so, it may be determined that some other task τ'_k also needs to be promoted to a greater priority, in order to be able to meet its deadline at the desired level of assurance. In that case, it will also promote this task, and repeat the entire process (the while loop in Lines 3-5) on the tasks remaining in $\tau_{\rm cur}$.

As in the original Audsley approach [1], augmentedAudsley is essentially identifying the tasks that are to be assigned the current priority p; hence, it does not (yet) assign the promoted tasks (τ_k and τ'_k, in the example scenario above) priorities relative to each other — this will be done later, during future recursive calls to augmentedAudsley (Line 9 of the pseudo-code), after all the tasks that can be assigned the priority p are identified. Hence in checking for deadline misses, only deadlines of (jobs of) tasks that have not yet been promoted are considered: augmentedAudsley does not check whether promoted tasks make their deadlines or not in the simulation, regardless of their criticality levels.

It is also important to point out that the decision to promote, to a greater priority, the task that has the earliest deadline miss is strictly necessary, in the sense that this task cannot remain in the current priority level regardless of which (if any) other tasks are chosen for promotion. This is a direct consequence of the fact that a task's "interference" — the amount of execution it is able to consume over any interval $[0, t)$ during the scheduling of the SAS — can only *increase* or remain unchanged upon promotion to a greater priority. Since the task chosen for promotion is the one with the earliest deadline miss, promoting any other tasks cannot possibly result in this deadline getting met — in other words, promotion is necessary if the task is to be guaranteed to meet its deadline.

augmentedAudsley repeats this process – identifying some task that cannot be guaranteed to meet its deadline at the desired criticality level and promoting it to a higher priority level – until all deadlines are met by the remaining tasks, or all the tasks have been promoted out of the lowest priority level. If we ended up needing to thus promote all the tasks (i.e., τ_{cur} becomes empty), then we have failed to determine a priority assignment that guarantees that all deadlines will be met at the desired criticality levels — this flags the failure declaration on Line 6 of the pseudo-code. Otherwise, we assign the lowest priority level to all the tasks that were not promoted, since we have determined that they all meet all their timing constraints at this priority level (Line 7 of the pseudo-code).

We then consider all tasks that had been promoted (stored in the set τ_{hi}). Since the tasks at the lower priority level have no effect on the schedulability of these tasks, we recursively determine whether these tasks in τ_{hi} are schedulable or not (Line 9 of the pseudo-code), using the same procedure as the one described above but starting at a greater priority ($p + 1$, rather than p).

5.2 Evaluation

In Section 5.1 above, we presented an algorithm for assigning priorities to tasks in a multi-criticality sporadic task system such that the resulting system is scheduled to meet all timing constraints to their respective desired levels of assurance, using the hybrid-priority scheduling algorithm of [2]. We now compare this approach to other possible approaches to the scheduling of multi-criticality sporadic task systems.

First, observe that our algorithm in Section 5.1 above *dominates* EDF scheduling: any τ that is EDF-schedulable will be determined to be schedulable by our algorithm with all tasks in τ being assigned the same – lowest – priority. That is, the first call to AUGMENTEDAUDSLEY$(\tau, 1)$ declares success by executing the return statement of Line 8, and making no further recursive calls to AUGMENTEDAUDSLEY.

Our algorithm also dominates the Vestal algorithm [9]. The Vestal algorithm is more restrictive in the sense that each priority level must have exactly one task assigned to it; it is not hard to show that any τ for which the Vestal algorithm finds such a priority assignment will also have a priority assignment (perhaps the same one) found by our algorithm.

As shown in Theorem 2 (Section 3), there are feasible multi-criticality sporadic task systems that are schedulable using dynamic priority algorithms, but not schedulable using any FJP or FTP algorithms. A natural question to ask is: is the hybrid-priority algorithm described in Section 5.1 optimal among the class of FJP algorithms, for the scheduling of multi-criticality sporadic task systems? Example 3 below answers this question in the negative: there are task systems that are FJP-schedulable, but that cannot be scheduled using the algorithm of Section 5.1.

Example 3 Consider the task system comprised of the two tasks τ_1 and τ_2, with the following parameters:

$$\begin{array}{lllll} C_1(1) = 5, & C_1(2) = 10, & D_1 = 12, & T_1 = \infty, & L_1 = 2, \\ C_2(1) = 1, & C_2(2) = 2, & D_2 = 5 & T_2 = 5, & L_2 = 1. \end{array}$$

First, it may be verified that it follows from Theorem 1 and the feasibility-testing algorithm for traditional sporadic task systems in [3] that this multi-criticality sporadic task system is feasible.

Next, it may be verified that is is not schedulable using the algorithm presented in Section 5.1 above:

- When both tasks are assigned the same priority, τ_1's only job in the SAS cannot be guaranteed to meet its deadline at the desired criticality level, since at criticality level 2 the total processor demand over the time-interval $[0, 12)$ is $(C_1(2) + 2C_2(2)) = 10 + 2 \times 2 = 14 > 12$.

- If τ_1 is instead assigned greater priority, then τ_2's first job in the SAS misses its deadline at time-instant 5.

However, the system is FJP-schedulable: the following FJP priority-assignment strategy ensures that all deadlines are met.

1. The only job of τ_1 is assigned priority level 2.
2. A job of τ_2 arriving at time-instant t_a is assigned priority 1 if τ_1's job has arrived and executed for ≥ 1 units prior to time-instant t_a, and priority 3 otherwise.

To see that this FJP scheduling strategy always meets all deadlines at the desired criticality levels, observe that

- Any task of τ_2 that is assigned priority 3 trivially meets its deadline (since there are no jobs of greater priority in the system).

- Suppose that a task of τ_2 is assigned priority 1. The only job of τ_1 must have completed at least one unit of execution; at τ_2's criticality level of 1, ($L_2 = 1$), this means that τ_1's job has $\leq (C_1(1) - 1) \leq 4$ units of execution remaining. Hence, this job of τ_2 can only be delayed by a further 4 time units, which still leaves it adequate execution to complete by its deadline.

- Now consider τ_1's sole job. At criticality level 2 ($L_1 = 2$), it can be delayed from executing for up to two time units by a priority-3 job of τ_2; however, if this happens, this job will have completed at least 3 units of execution before the arrival of the *next* job of τ_2. Consequently, this next job of τ_2 is assigned priority 1 and does not effect the execution of τ_1's sole job, which therefore suffers a total interference of at most 2 units and consequently completes on time.

Hence, this is a multi-criticality sporadic task system that is FJP-schedulable, but which the algorithm of Section 5.1 fails to schedule. ■

6 Conclusions

The multi-criticality task model presented by Vestal [9] represents a potentially very significant advance in the modeling and analysis of safety-critical real-time systems. In this paper, we have attempted to obtain a better understanding of the preemptive uniprocessor scheduling of multi-criticality sporadic task systems. We have studied the fundamental scheduling-theoretic issues — expressiveness; feasibility; schedulability by specific classes of scheduling algorithms; etc. — that must be understood in order to fully understand a new task model. We have learned much from this study that went against our prior experience with traditional sporadic task systems: for example, we learned that EDF is not an optimal algorithm for scheduling multi-criticality sporadic task systems, and that EDF and fixed-priority scheduling are incomparable. We have also derived, and evaluated, a new scheme for scheduling such multi-criticality sporadic task systems upon preemptive uniprocessors, and shown that this new scheme is superior to previously-proposed ones.

References

[1] AUDSLEY, N. C. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Tech. rep., The University of York, England, 1991.

[2] BARUAH, S., AND FISHER, N. Hybrid-priority scheduling of resource-sharing sporadic task systems. In *Proceedings of the Real-Time and Embedded Technology and Applications Systems Symposium* (St. Louis, April 2008), IEEE Computer Society Press.

[3] BARUAH, S., MOK, A., AND ROSIER, L. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the 11th Real-Time Systems Symposium* (Orlando, Florida, 1990), IEEE Computer Society Press, pp. 182–190.

[4] BUTTAZZO, G. C. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, second ed. 2005.

[5] CARPENTER, J., FUNK, S., HOLMAN, P., SRINIVASAN, A., ANDERSON, J., AND BARUAH, S. A categorization of real-time multiprocessor scheduling problems and algorithms. In *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, J. Y.-T. Leung, Ed. CRC Press LLC, 2003.

[6] LEUNG, J., AND WHITEHEAD, J. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation 2* (1982), 237–250.

[7] LIU, C., AND LAYLAND, J. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM 20*, 1 (1973), 46–61.

[8] MOK, A. K. *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.

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