Sustainability in Mixed-Criticality Scheduling

Zhishan Guo Sai Sruti Bryan C. Ward* Sanjoy Baruah†
Department of Computer Science, Missouri University of Science and Technology
*Lincoln Laboratory, Massachusetts Institute of Technology
†Department of Computer Science & Engineering, Washington University in St. Louis
{guozh,sxq6}@mst.edu, bryan.ward@ll.mit.edu, baruah@wustl.edu

***Abstract*—*Sustainability* is a formalization of the requirement for scheduling algorithms and schedulability tests that a system deemed to be correctly schedulable should remain so if its run-time behavior is better than anticipated. The notion of sustainability is extended to mixed-criticality systems, and sustainability properties are determined for a variety of widely-studied uniprocessor and multi-processor mixed-criticality scheduling algorithms.**

1. Introduction

Schedulability tests play an important role in the verification of safety-critical real-time systems. Given the specification of an instance comprising the abstraction of workload and the computing platform upon which the workload is to execute, a schedulability test determines whether all timing constraints (often specified by deadlines) are guaranteed to be met under specified scheduling policies. For safety-critical systems, schedulability analysis must be performed prior to run-time; in order to do so, parameters characterizing the run-time workload (such as worst-case execution time and, for recurrent sporadic tasks, minimum duration separating successive invocations) must be estimated prior to run-time. Different tools and techniques used for making such estimations may be more or less conservative (pessimistic) than each other; hence, the use of conservative estimates may result in systems exhibiting run-time behavior "better" than estimated. The notion of *sustainability* was introduced [4] to formalize the expectation that a system that is schedulable under its worst-case specifications should remain schedulable when its actual run-time behavior is better than the worst-case.

Mixed-criticality systems. There has been an increasing trend in safety critical systems towards *mixed-criticality* implementations, where components that are assigned different levels of criticality are integrated onto a common hardware platform. For example, different ASILs (Automotive Safety and Integrity Levels), DALs (Design Assurance Levels or Development Assurance Levels) and SILs (Safety Integrity Levels) are defined in industrial standards such as IEC 61508, DO-178B and DO-178C, DO-254, and ISO 26262) [9], [10], [13]. An abstract model for representing mixed-criticality workloads was proposed by Vestal [15] about ten years ago; this model has been widely adopted in the real-time scheduling theory community and a large body of research studying various aspects of scheduling and analysis of systems specified according to this model (and its extensions) has been generated — see, e.g., [8] for a review. Taking into account the rise in interest in mixed-criticality scheduling, we believe it is important to study the sustainability of typical mixed-criticality scheduling algorithms and schedulability tests.

Motivation and Challenges. One obvious aspect of sustainability for mixed-criticality workloads that we will examine in this paper is this: if a system is deemed mixed-criticality schedulable, does it remain so if the criticality level of some task within it is reduced (i.e., made less critical)? Sustainability with regards to criticality level is an intellectually interesting question, and we believe that obtaining answers to this question will enhance our insight into the process of criticality-cognizant resource-allocation. It is also potentially relevant to the design and analysis of mixed-criticality systems as the basis for "*what if*" design-space exploration as part of an interactive design process — suppose that a mixed-criticality system design is deemed schedulable, is it safe to re-specify a part of the system as being of lower criticality? Alternatively, if a system is deemed non-schedulable, does it always help reduce the criticality of some task? The answers to these questions are important as the certification process can be expensive, both monetarily and in terms of time and labor. For a non-sustainable mixed-criticality scheduler, a developer may be required to certify a task at a higher criticality than needed (based upon functionality and actual criticality) solely in order to make the system schedulable. For a sustainable mixed-criticality scheduler, however, the decision to certify a task at a given criticality level is predicated exclusively on its actual criticality.

In addition to questions regarding sustainability with respect to criticality-level, sustainability analysis of mixed-criticality systems throws up some other interesting issues. A key aspect of the Vestal model [15] of mixed-criticality

SCHEDULER	CRIT. LEVEL	WCET	PERIOD	DEAD--LINE
Crit. Mono. (Sec 3.1)	N	Y	Y	Y
EDF-VD (Sec 3.2)	Y	Y	Y	Y
AMC (Sec 3.3)	Y	Y	Y	Y
OCBP (Sec 3.4)	Y	Y	Y*	Y
MC^2 (Sec 4.1)	N	Y	Y	Y
MC Fluid (Sec 4.2)	Y	Y	Y	Y

*Please note that OCBP is for scheduling MC job sets and thus the 'Y' in the period column represents sustainability over release time, while others are for MC task set scheduling.

TABLE 1: Summary of sustainability results for some MC scheduling algorithms with respect to various parameters. A 'Y' / 'N', denotes that the scheduler is / is not sustainable with respect to that parameter. The section numbers denote the section in the text where the algorithms are briefly described, and the sustainability properties derived. (The first four listed algorithms are uniprocessor algorithms; the remaining two, multiprocessor ones.)

workloads is that task parameters are dependent on the criticality levels. For example, a piece of code is characterized with a larger WCET estimate if it is defined to be safety-critical (that requires a higher level of assurance) than it would if it is of lower criticality. This feature of mixed-criticality systems undermines many of the sustainability results that have previously been obtained, since prior sustainability analyses were conducted in a manner that is agnostic of criticality levels. As a result, not only does sustainability with regards to criticality levels need to be studied, sustainability with regards to other parameters must also be revisited. Accordingly in this paper, we perform sustainability analysis upon several uniprocessor and multiprocessor mixed-criticality scheduling algorithms. For the most part, we limit ourselves in this paper to two criticality levels – although many results are easily extended to more than two levels, we leave filling in the details as future work. The algorithms that we study, and their sustainability properties over various parameters, are summarized in Table 1.

Organization. The rest of this paper is organized as follows. Section 2 establishes the system model and formally defines the terminologies and notations. The analysis of uniprocessor mixed-criticality scheduling algorithms is covered in Section 3, and of multiprocessor algorithms in Section 4. Section 5 briefly reflects on some related works and Section 6 concludes the paper.

2. System Model and Preliminaries

We will primarily model a mixed-criticality (MC) workload as comprising a finite specified collection of MC sporadic tasks, each of which may generate an unbounded number of MC jobs. (In Section 3.4 we consider the scheduling of specified collections of jobs rather than tasks.)

MC instance. An MC system instance is characterized by (i) a platform specification as either *uniprocessor* or identical *multiprocessors* (in which case the number m of processors is specified; and (ii) an *MC workload* with a collection of either *MC jobs* or *MC tasks*.

A job models a single piece of code, to be executed sequentially once upon a processor. An *MC job* J_i is characterized by a 5-tuple of parameters $(a_i, c_i^L, c_i^H, d_i, \chi_i)$, where

- $a_i \geq 0$ denotes its release time (after which the piece of code may start to execute),
- $c_i^L \leq c_i^H \in \mathbb{R}_+^L$ are per-mode WCET estimations,
- $d_i \geq a_i$ indicates the deadline, and
- $\chi_i \in \{\text{LO}, \text{HI}\}$ represents the criticality level.

An MC *task* $\tau_i = \{C_i^L, C_i^H, T_i, D_i, \chi_i\}$ characterizes a single piece of code, to be executed *repeatedly* for an indefinite length of time in an MC system. That is, a *sporadic task* gives rise to a potentially unbounded sequence of jobs — a release is triggered when the corresponding piece of code becomes ready for execution. The period parameter T_i represents the minimum inter-arrival time between any two consecutive job releases (by the same task). Another parameter D_i, denoting the *relative deadline*, is specified for the task, denoting that the deadline for each job is its release time plus the D_i value. We sometimes consider a special subclass of sporadic tasks: *implicit deadlines* task systems have $D_i = T_i$ for all i.

The following *parameters* capture important attributes of a collection of MC tasks and are widely used in schedulablity analysis. The least common multiple of the period parameters of all the tasks is referred to as the *hyper-period* of the task system and is denoted by H. The per-mode *utilizations* of each task τ_i and the whole set τ are defined as follows respectively.

$$u_i^y = \frac{C_i^y}{T_i}, \ y \in \{\text{LO}, \text{HI}\}, \tag{1}$$

$$U_x^y(\tau) = \sum_{\tau_i \in \tau \bigwedge \chi_i = x} u_i^y = \sum_{\tau_i \in \tau \bigwedge \chi_i = x} \frac{C_i^y}{T_i}, \ x, y \in \{\text{LO}, \text{HI}\}. \tag{2}$$

System behavior. An MC system is assumed to begin execution in LO-mode. If a job has executed for more than its LO-criticality WCET specification without signaling completion, a system-wide mode switch to HI-mode is said to occur. The system returns to LO-mode at the first idle instant after the mode switch [14]. In all other scenarios, the system is considered as an erroneous mode, where no correctness guarantees are made and thus is not considered in this work.

MC schedulability. MC schedulability incorporates correctness guaranteed under both LO- and HI-modes. A scheduling algorithm is *correct* if it is able to schedule every job sequence of a workload such that (i) jobs of both LO- and HI-criticality tasks execute for up to their LO-WCETs and are completed before their deadlines in LO-mode, and (ii) jobs of HI-criticality tasks execute for their HI-WCETs and are completed before their deadline in HI-mode.

MC sustainability. An MC scheduling policy is said to be sustainable if any MC instance that is MC-schedulable by the policy remains so if one or more of the parameters characterizing the instance is "improved". Analogously, a schedulability test for some MC scheduling policy is said to be sustainable if any MC instance that is deemed MC-schedulable by the schedulability test will continue to be deemed MC-schedulable by the test if one or more of its parameters is improved. We now list what we consider to be "improvements" to the characterizing parameters:

1) Decreasing WCET parameters (C_i^L and/or C_i^H).
2) Increasing periods for sporadic task systems; moving job release times forward (i.e., decreasing a_i) for collections of jobs.
3) Postponing relative deadlines (D_i).
4) Decreasing the criticality level assignment of a task/job (from HI to LO for dual-criticality systems).

A scheduling policy and/or schedulability test may be sustainable with respect to some but not all parameters. In each of the following subsections, we will consider one well-known MC scheduler, and examine the sustainability with respect to all four parameters.

3. Uniprocessor MC Sustainability

In this section, we study sustainability properties of four uniprocessor MC scheduling algorithms and their associated schedulability tests. The first three – Criticality Monotonic, EDF-VD, and AMC – are task-scheduling algorithms; the fourth, OCBP, schedules collections of jobs.

3.1. Criticality Monotonic

Criticality Monotonic (CM) [15] is a scheduling policy that schedules at each time instant an available job of highest criticality. Hence a task of criticality level ℓ cannot affect the scheduling of tasks of criticality greater than ℓ. In this paper we restrict ourselves with only two criticality levels, LO and HI. We study a sporadic task model where each task is characterized by $\tau_i = \{C_i^L, C_i^H, T_i, D_i, \chi_i\}$.

We assume that the (non MC) mechanism to schedule tasks within each criticality level is sustainable w.r.t. all parameters, and examine the sustainability of CM as a general MC scheduling framework.

Sustainability w.r.t. relative deadline, WCET, and period. Changing relative deadline, WCET, and period parameters will not affect the general CM framework since no criticality level is modified. As it is assumed that the scheduler used within each criticality level is sustainable to all parameters, the schedulability conditions will still hold within each criticality level, leading to sustainability of CM.

Sustainability w.r.t. criticality levels.

Theorem 1. *Criticality Monotonic scheduling algorithm is **not** sustainable with respect to criticality levels.*

Proof. Consider the task-set shown in Table 2, which is CM-schedulable (using deadline monotonic within each criticality level). Figure 1(a) illustrates the schedule of the task-set with its respective arrival times and deadlines.

Task	C_i^L	C_i^H	T_i	D_i	Criticality	Priority
τ_1	20	25	120	40	HI $\rightarrow$LO	$1 \rightarrow 3$
τ_2	28	60	200	160	HI	2
τ_3	12	12	120	100	LO	4

TABLE 2: An MC task-set that is not sustainable under criticality monotonic scheduling policy.

We now decrease the criticality level of task τ_1 from HI to LO and observe the outcome schedule in Figure 1(b), where τ_1 misses its deadline. Thus we conclude that the CM scheduling policy is not sustainable w.r.t. criticality levels. □

3.2. Earliest Deadline First with Virtual Deadlines (EDF-VD)

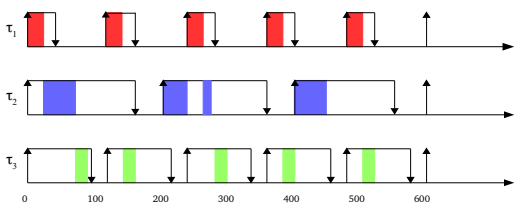
The Earliest Deadline First with Virtual Deadline scheduling policy (EDF-VD) [3] is an adaptation of the Earliest Deadline First (EDF) algorithm to dual-criticality implicit-deadline sporadic task systems. It is proved in [3] that EDF-VD correctly schedules any dual-criticality task system $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ upon a unit-speed preemptive processor if

$$x\, U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{HI}}(\tau) \leq 1 \tag{3}$$

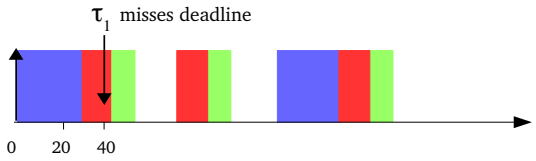
where x is defined as follows:

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

Condition 3, in fact, constitutes a schedulablity test for EDF-VD: EDF-VD computes x according to Equation 4 above and determines whether Condition 3 is satisfied. In the remainder of this section, we establish that this schedulability test for EDF-VD is sustainable with respect to criticality level, WCETs, and period. (Since this schedulability test is for implicit-deadline task systems, its sustainability with respect to relative deadlines trivially follows from the observation that EDF-VD does not make use of the relative deadline parameter.)



(a) Criticality-Monotonic schedule for tasks in Table 2 under LO-criticality mode.



(b) Criticality-Monotonic schedule for tasks in Table 2, when criticality level of τ_1 is changed from HI to LO, where τ_1 misses its deadline.

Figure 1: Schedule demonstration of the sample task set (shown in Table 2) under Criticality-Monotonic before and after the change of criticality level of one of the tasks (τ_1).

Recall the various *utilization* parameters defined in Expression 2. Let us introduce some simplifying notations:

$$\begin{aligned} u_l &\leftarrow U_{\text{LO}}^{\text{LO}}(\tau) \\ u_h &\leftarrow U_{\text{HI}}^{\text{LO}}(\tau) \\ u'_h &\leftarrow U_{\text{HI}}^{\text{HI}}(\tau) \end{aligned}$$

While executing in LO-criticality mode, the deadlines of the high criticality tasks are determined by scaling down the original period of a HI-criticality task with a factor x (x $\leq$ 1) to obtain a virtual deadline. The scaling factor x is calculated off-line as $x = u_h/(1 - u_l)$.

For the EDF-VD scheduling policy to correctly schedule a dual-criticality implicit deadline task system on a single unit-speed processor, the sufficient conditions for tasks to be scheduled in both LO- and HI-mode respectively are [3]:

$$x \geq \frac{u_h}{1 - u_l}, \tag{5}$$

$$x \cdot u_l + u'_h \leq 1. \tag{6}$$

We now determine the sustainability of the scheduling policy by making favorable alterations in the parameters and verify if the schedulability condition still persists.

Lemma 2. *EDF-VD is sustainable w.r.t criticality levels; i.e., when changing the criticality of a task from HI to LO, Conditions* (5) *and* (6) *will continue to hold if they used to be so.*

Proof. The change to the criticality level of a task from HI to LO will result in an increase of the utilization of LO tasks (u_l) and decreases in the utilization of HI tasks (u_h, u'_h), all with the same amount (assumed to be δ) i.e.,

$$u_h = u_h - \delta,$$

$$u'_h = u'_h - \delta,$$

$$u_l = u_l + \delta.$$

Now, on substituting these notations in Equations (5) and (6), the scaling term x can then be denoted as:

$$\text{x} \leftarrow \frac{u_h}{1 - u_l}.$$

The equation for the HI-criticality schedulability test can be written as:

$$\frac{u_h.u_l}{1 - u_l} + u'_h \leq 1. \tag{7}$$

On modifying the utilization values with δ in the Equation (7) we get:

$$\frac{(u_h - \delta)(u_l + \delta)}{1 - (u_l + \delta)} + (u'_h - \delta) \leq 1. \tag{8}$$

To determine if the schedulability condition in Equation 8 still holds, we show the following proof: By demonstrating that the difference between Equations (7) and (8) is positive, i.e.,

$$\frac{(u_h - \delta)(u_l + \delta)}{1 - (u_l + \delta)} + (u'_h - \delta) \leq \frac{u_h.u_l}{1 - u_l} + u'_h \leq 1. \tag{9}$$

$$\begin{aligned} &\frac{u_h \cdot u_l}{1 - u_l} + u'_h - \frac{(u_h - \delta)(u_l + \delta)}{1 - (u_l + \delta)} + (u'_h - \delta) \geq 0 \\ &\Rightarrow \frac{u_h \cdot u_l}{1 - u_l} - \frac{(u_h - \delta)(u_l + \delta)}{1 - (u_l + \delta)} - \delta \geq 0 \\ &\Rightarrow \frac{u_h.u_l}{1 - u_l} - \frac{(u_h - \delta)(u_l + \delta) - \delta + \delta(u_l + \delta)}{1 - (u_l + \delta)} \geq 0 \\ &\Rightarrow \frac{u_h \cdot u_l}{1 - u_l} - \frac{(u_l + \delta)u_h - \delta}{1 - (u_l + \delta)} \geq 0 \\ &\Rightarrow \frac{u_h \cdot u_l}{1 - u_l} - \frac{u_h.u_l + \delta u_h - \delta}{1 - (u_l + \delta)} \geq 0 \\ &\Rightarrow u_h \cdot u_l(1 - u_l + \delta) - u_h \cdot u_l(1 - u_l) - \delta(u_h - 1)(1 - u_l) \geq 0 \\ &\Rightarrow \delta(1 - u_h)(1 - u_l) - u_h \cdot u_l\delta \geq 0 \\ &\Rightarrow (1 - u_h)(1 - u_l) \geq u_h \cdot u_l \\ &\Rightarrow 1 - u_h - u_l \geq 0 \\ &\Rightarrow u_l + u_h \leq 1. \end{aligned} \tag{10}$$

The solution obtained in Equation (10) satisfies the schedulability conditions stated in Equations (5) and (6), thus establishing that the EDF-VD scheduling policy is sustainable w.r.t. to criticality levels. □

Lemma 3. *EDF-VD is sustainable w.r.t WCETs.*

Proof. According to the definition of sustainability, on decreasing the WCET of a task τ_i (either C_i^L or C_i^H), the schedulability conditions of the whole task system should still hold. To demonstrate the sustainability, we consider a small arbitrary value δ by which we decrease C_i^L or C_i^H values, and check the two sufficient conditions.

(1) Decrease of C_i^H.

Upon decreasing the C_i^H by a menial amount $\delta > 0$, the utilization of the set (u_h) will decrease by a value ($\delta' = \delta/T_i > 0$). Thus, the corresponding HI-mode schedulability condition for the new task set is:

$$x \cdot u_l + (u_h' - \delta') \leq 1, \tag{11}$$

which obviously holds from Condition (6) and the fact that $\delta' > 0$. This conveys that in HI mode, decreasing the C_i^H value does not have any adversary effect on the schedulability of the whole system.

Now we will check the schedulability under LO mode; i.e., if the condition in Equation (5) holds. Since C_i^H values have nothing to do with the condition for LO mode, it remains true. Thus we conclude that the decrease of C_i^H will not have any adversary effects on the schedulability of the whole system.

(2) Decrease of C_i^L.

Similarly, if C_i^L is diminished in such a way, the u_l value decreases by δ. The following schedulability test (in HI mode) will also hold as $x > 0$ and $\delta > 0$.

$$x \cdot (u_l - \delta) + u_h' \leq 1. \tag{12}$$

Now we inspect the schedulability under LO mode; i.e., if the condition in Equation (5) complies after substituting the modified value of u_l:

$$\frac{u_h}{1 - (u_l - \delta)} \leq \frac{u_h}{1 - u_l} = x, \tag{13}$$

and

$$\frac{u_h - \delta}{1 - u_l} \leq \frac{u_h}{1 - u_l} x. \tag{14}$$

This indicates that the condition for LO-mode correctness continues to prevail. □

Lemma 4. *EDF-VD is sustainable w.r.t period.*

Proof. This follows directly from the proof for sustainability over WCETs as an increasing period will lead to a decrease of per-mode utilization. □

Theorem 5. *EDF-VD is sustainable w.r.t all parameters.*

Proof. This follows from Lemmas 2, 3, and 4. □

3.3. AMC

The adaptive mixed criticality scheduling policy (AMC) [5] is a fixed-priority algorithm for scheduling MC sporadic task systems on preemptive uniprocessors. A priority order is achieved by applying Audsley's priority assignment algorithm [1], and has been demonstrated to be optimal [5], [15]; i.e., whenever a feasible priority order exists, the system will be AMC-schedulable.

Response Time Analysis (RTA) techniques are used to determine the schedulability of AMC scheduling policy. The analysis is done in three phases [5]:

1) Verifying schedulability of LO-criticality mode with:

$$R_i^L = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^L}{T_j} \right\rceil C_j^L, \tag{15}$$

where *hp(i)* is the set of all tasks with priority higher than that of task τ_i.

2) Verifying schedulability of HI-criticality mode with

$$R_i^H = C_i + \sum_{j \in hpH(i)} \left\lceil \frac{R_i^H}{T_j} \right\rceil C_j^H, \tag{16}$$

where *hpH(i)* is the set of HI-critical tasks with priority higher than, or equal to, that of task τ_i.

3) Verifying schedulability during criticality (mode) change in an iterative manner w.r.t. maximum response time R_i^* until it is stabilized with:

$$R_i^* = C_i^H + \sum_{j \in hpH(i)} \left\lceil \frac{R_i^*}{T_j} \right\rceil C_j^H + \sum_{j \in hpL(i)} \left\lceil \frac{R_i^L}{T_k} \right\rceil C_k^L. \tag{17}$$

Theorem 6. *AMC is sustainable w.r.t. to all parameters*

Proof. The proof will contain two parts – one for showing sustainability w.r.t. criticality levels, and the other for the remaining parameters:

Sustainability w.r.t WCETs, periods, and relative deadlines. It has been proved by Baruah and Burns in [7] that the response time analysis of fixed priority preemptive task system is sustainable w.r.t parameters such as execution requirements (C_i), relative deadlines (D_i) and periods (T_i). Thus Conditions (15) and (16) will hold when we adjust the parameters.

With respect to Condition (17), although the value of R_i^L is fixed, decreasing C_k^L and increasing T_k will deplete the overall value of response time R_i^*. The modified value of response time can be recursively determined until a value less than the initial response time is obtained. The altered value of R_i^* is acquired from recursive calculations and can be represented as:

$$new(R_i^*) \leq R_i^* \leq D_i$$

The above equation satisfies the schedulability condition for AMC scheduling algorithm. For all tasks $\tau_i \in \tau_{LO,HI}$ the response time R_i^L and R_i^H are no larger than the relative deadline D_i. It is observed that the amount of execution available to a task τ_i over a period [0,t) can only increase if job execution requirements decrease. The similar rationale is applied when job periods increase, i.e., modifying the parameters accordingly only guarantees the execution of the task in [0, R_i]. Consequently, the AMC scheduling model is sustainable with respect to execution-requirements, periods and relative deadlines.

Sustainability w.r.t criticality levels. As mentioned earlier, the AMC scheduling policy employs an optimal priority assignment technique before scheduling the jobs. [5] states that Audsley's priority assignment algorithm delivers an optimal priority ordering in polynomial time, i.e., Audsley's algorithm is guaranteed to find a priority assignment, if there exists one, which is AMC-schedulable. If we change the criticality level of a task from HI to LO, the priority of the task may remain the same or decrease; if another feasible priority assignment exists, it will be determined by the Audsley's algorithm. In case no other feasible priority order exists, the available order of the task-set before the criticality level modification can be used as the valid priority ordering. Since the order is already AMC-schedulable, we claim that it is sustainable w.r.t. criticality levels.

□

3.4. OCBP for MC Job Scheduling

The OCBP (Own Criticality Based Priority) scheduling policy [6] is a priority based MC-job scheduling algorithm. It derives a valid priority ordering of the jobs prior to run-time in order to guarantee a correct schedule. These priorities are assigned in a recursive manner following Audsley's approach [1]. That is, a job J_i is assigned lowest priority if it meets its deadline, while J_i and all other jobs (of higher priority) execute for a duration not exceeding their WCETs estimated at J_i's own criticality level χ_i. If such a job J_i is found, then it is assigned lowest priority and the process repeated on the remaining (higher-priority) jobs. Specifically if the candidate job J_i of LO-criticality is assigned lowest priority, the following set of conditions will be checked for any l such that $l \in hp(i)$[1] and $a_l \leq d_i$:

$$C_i^L + \sum_{j \in hp(i) \cap a_j \geq a_l} C_j^L \leq d_i - a_l. \quad (18)$$

1. $hp(i)$ indicates the set of jobs with higher priority assignment than J_i.

While if J_i is of HI-criticality, for any l such that: $l \in hp(i)$ and $a_l \leq d_i$, we check[2]:

$$C_i^H + \sum_{j \in hp(i) \cap \chi_j = \text{HI} \cap a_j \geq a_l} C_j^H \leq d_i - a_l. \quad (19)$$

It is relatively straightforward to implement this priority-assignment process in such a manner that the following assumption is satisfied:

Assumption 1: *Upon changing some parameter of a single job, the priority assigned to this particular job may be different from the original but the relative priority order of other jobs remains the same.*

This assumption can be achieved by restricting the order of the jobs in each iteration while determining a lowest priority job; e.g., in decreasing deadline order or simply following job indices.

Theorem 7. *OCBP is sustainable for all parameters under Assumption 1.*

Proof. Assume that an instance J of dual-criticality jobs is OCBP schedulable, and modify the parameter of a particular job $J_i \in J$ by one of the four actions: decrease its release time by δ, increase its deadline by δ, decrease its LO- or HI-WCET by δ, or change its criticality level from HI to LO ($C_i^L \leq C_i^H$) to obtain a new job set J' (while parameters of other jobs remains unchanged). According to Audsley's priority assignment algorithm and Assumption 1, there are three possible scenarios upon assigning priorities to J': (1) the job J_i is assigned a higher priority than before, (2) the priority order of all jobs does not change, and (3) the job J_i is assigned a lower priority than before.

We first show that Case (1) is not possible. Since OCBP is a fixed priority scheme, the schedulability of J_i is only affected by the higher priority jobs. If J_i is assigned priority p_i at a certain iteration before changing the parameter, we would make the same attempt to assign it the lowest priority at that round, with the same higher priority jobs left (according to Assumption 1). Since it is schedulable before the parameter change, the claim is that J_i will continue be assigned the lowest priority at that round (if not sooner). The reason is that changing J_i's parameters in the given manner will just relax Conditions (18) and (19) such that the schedulaiblity test on current priority assignment remains a success.

For Case (2), sustainability also holds as Conditions (18) and (19) will continue to subsist for other jobs. For J_i, again the condition are more relaxed and will continue to hold. As a result, OCBP will return success after change in the parameters.

For Case (3), since parameter changing is leading to relaxation of original conditions, it is possible that J_i can be

2. Note that none of the existing work stated the math conditions for OCBP to be schedulable – this is part of our contribution in this paper as sustainability proof requires clearly expressed equations.

assigned a priority earlier than before; i.e., a lower priority than the one before such change. Figure 2 depicts the priority assignment of the jobs before and after incorporating the criticality level change.

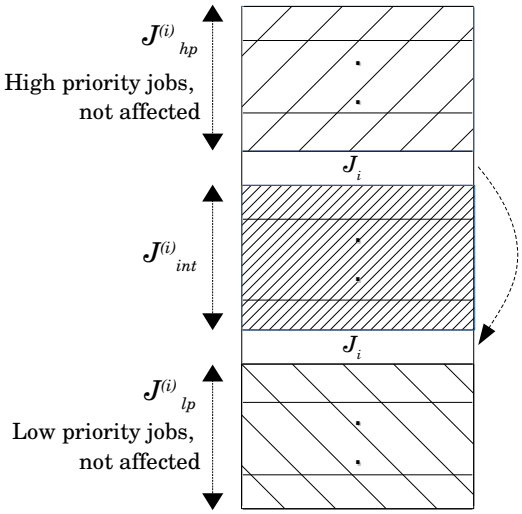


Figure 2: Priority assignment before and after the change of job J_i's criticality level from HI to LO.

The first shaded section $J^{(i)}_{hp}$ comprises jobs that initially have a higher priority than job J_i. Similarly, after J_i is assigned a lower priority, the jobs with still lower priority than J_i forms a set denoted by $J^{(i)}_{lp}$. The densely shaded region in the middle, denoted by $J^{(i)}_{int}$; is the rest of jobs that are originally assigned lower priority than J_i and then higher than J_i after its parameter gets changed.

The entire job set can be represented as:

$$J^{(i)}_{hp} \cap\ J_i \cap\ J^{(i)}_{int} \cap\ J^{(i)}_{lp}.$$

It is assumed that the priority order within each subset does not change (as we restrict OCBP to try the same order each round). We first know that J_i's schedualbility conditions are satisfied.

For the rest of the jobs in three sets:

- The schedulability conditions of jobs in $J^{(i)}_{hp}$ is never affected as lower priority jobs have nothing to do with their priority assignment check.
- The schedulability conditions of jobs in $J^{(i)}_{int}$ will hold as for any job in this set, there is one less higher priority job (J_i) after the change.
- The schedulability conditions of jobs in $J^{(i)}_{lp}$ will hold as well, since for them the higher priority job set remains the same, while one of them, J_i, has less interference than before due to the parameter change.

We can thus claim that the job-set is OCBP-schedulable after the parameter change of job J_i.

□

4. Sustainability Analysis of Multi-core Scheduling Algorithms

We now study the sustainability properties of two multiprocessor MC scheduling algorithms.

4.1. MC^2

The MC^2 algorithm [12] employs a hierarchical scheduling approach: special tasks called container tasks are scheduled alongside the higher-criticality tasks. LO-criticality tasks will be assigned to containers (i.e., servers) and will use the container's budget to execute only when the container task is scheduled for execution in the platform. Tasks at each criticality level are scheduled by different intra-container schedulers, and thus according to different scheduling policies. Four criticality levels are considered in [12] – A, B, C, and D. Level-A tasks adopt a table-driven approach modeled on a cyclic executive scheduler, with tasks statically assigned to processors and scheduling tables precomputed prior to runtime. Each processor also hosts a level-B container, to which level-B tasks are assigned. Partitioned EDF is used at level B, so each level B container is served by an EDF scheduler. The periods of all level-B tasks are required to be integer multiples of the level-A hyperperiod, and the sum of the utilizations of all level-A and level-B tasks must not exceed 1.0. Both level-A and level-B tasks are guaranteed to meet their deadlines. Level-C tasks are grouped into the Level-C container which is served by all processors and is scheduled using global EDF. Level-C tasks are guaranteed only for soft real-time correctness (i.e., with bounded tardiness). G-EDF is executed on any processor whenever some level-C task is eligible but no higher-criticality tasks (level-A or -B) are eligible. At level D, "best effort" jobs are scheduled by a server that is invoked whenever a processor would otherwise be idle – no guarantee is made to those.

Sustainability w.r.t WCET. As stated earlier, the cyclic executive execution of level A tasks is table driven and the execution order is determined off-line. Therefore on decreasing WCET of a level-A task, a modified schedule is established off-line according to which tasks are dispatched. For an existing MC^2-schedulable task set, on decreasing the WCET of a level B task by a small value $\delta > 0$, the utilization of the task decreases, which results in an easier partitioning problem to obtain a partitioned-EDF schedule. Thus, the schedulability of the set will be maintained. For level-C tasks, MC^2 only guarantees the tardiness bound instead of hard real-time constraints. We therefore conclude that MC^2 algorithm is schedulable w.r.t to execution time.

Sustainability w.r.t relative deadlines and periods. On increasing the period/deadline of the task by δ, the schedulability conditions should hold in order to establish sustainability. Since tasks of level A are statically scheduled while level B and C adopt partitioned-EDF and global-EDF respectively, on increasing the deadline by a small value δ, the schedulability conditions are not affected adversely and continue to persist.

Sustainability w.r.t criticality level. We separately consider the cases where a task's criticality level is lowered from A to B, B to C, and C to D.

Level A to B: Levels A and B are criticality-monotonically partitioned; since we have shown (theorem 1) that criticality monotonic is not sustainable, it follows that MC^2 is not sustainable w.r.t criticality level for level-A tasks.

Level B to C: At level C, tasks are allocated at instants when the processor is available and not consumed by tasks of levels A and B. The following theorem is proved in the Appendix.

Theorem 8. *The MC^2 scheduling algorithm is* not *sustainable with respect to the criticality-level change $B \rightarrow C$.*

Level C to D: Since no guarantee is made for level-D task, such change is trivially sustainable (although rather meaningless).

Overall, we conclude that MC^2 is **not** sustainable w.r.t criticality level in general.

4.2. MC-Fluid

In the MC Fluid scheduling algorithm [11], scheduling occurs under a fluid scheduling model which allows for schedules in which an individual task may be assigned a fraction of a processor at each time instant. Each job of each task τ_i is executed at a rate of θ_i^L under LO-criticality mode, and another at a rate of θ_i^H after a mode switch (with $\theta_i^H = 0$ for all LO tasks).

The MC-Fluid schedulability conditions [11] for a task set τ and associated LO- and HI-mode execution rates (θ_i^L and θ_i^H) is MC-schedulable under MC-Fluid if and only if the following set of conditions:

$$\forall \tau_i \in \tau, \theta_i^L \geq u_i^L \tag{20}$$

$$\forall \tau_i \in \tau_H, \frac{u_i^L}{\theta_i^L} + \frac{u_i^H - u_i^L}{\theta_i^H} \leq 1, \tag{21}$$

$$\sum_{\tau_i \in \tau} \theta_i^L \leq m, \tag{22}$$

$$\sum_{\tau_i \in \tau_H} \theta_i^H \leq m \tag{23}$$

We now establish the sustainability of the MC-Fluid scheduling algorithm with respect to different parameters.

Lemma 9. *MC-Fluid is sustainable w.r.t WCET and period.*

Proof. According to the definition of sustainability, on decreasing the WCET of a task τ_i (either C_i^L or C_i^H) and/or increasing the time period, the schedulability conditions of the whole task system will still hold.

To analyze sustainability w.r.t to execution amounts C_i and time period T_i, for a task $\tau_i \in \tau_L$, we decrease C_i^L by a small arbitrary value and/or increase period (T_i). As a result, all modifications can be modeled as a decrease of LO-utilization (u_i^L) by an amount of $\delta > 0$. We then examine the conditions one by one.

Equation (20) can be written as:

$$\forall \tau_i \in \tau, \theta_i^L \geq (u_i^L - \delta)$$

and is true for any value of $\tau_i \in \tau_L$.

Consider Equation (21) where $\tau \in \tau_H$, we determine the effect of decreasing C_i^L and C_i^H on Equation (21). On decreasing the value of C_i^L by δ, the u_i^L also decreases. Substituting in Equation (21) we get:

$$\forall \tau_i \in \tau_H, \frac{u_i^L - \delta}{\theta_i^L} + \frac{u_i^H - (u_i^L - \delta)}{\theta_i^H} \leq 1. \tag{24}$$

In order to prove that the condition still holds, we subtract the left hand side of Equation (24) from that of Equation (21) and establish that it is greater than zero.

$$\frac{u_i^L}{\theta_i^L} + \frac{u_i^H - u_i^L}{\theta_i^H} - \frac{u_i^L - \delta}{\theta_i^L} - \frac{u_i^H - (u_i^L - \delta)}{\theta_i^H} \geq 0$$

$$\Leftrightarrow \frac{\delta}{\theta_i^L} - \frac{\delta}{\theta_i^H} \geq 0$$

$$\Leftrightarrow \delta(\frac{1}{\theta_i^L} - \frac{1}{\theta_i^H}) \geq 0$$

The above equation can be easily validated since Equation (21) only considers HI-criticality tasks, where $\theta_i^L \leq \theta_i^H$ holds.

It is obvious that Conditions (22) and (23) will not get affected by utilization changes.

We can thus conclude that MC-Fluid scheduling is sustainable w.r.t the execution time (C_i) and time period (T_i). □

Lemma 10. *MC-Fluid is sustainable w.r.t criticality levels.*

Proof. To check the sustainability of the scheduling model w.r.t. the criticality levels. i.e., if we change the criticality of a task from HI to LO, then Condition (21) no longer needs to be validated for this task. Thus if the original conditions can be satisfied, the new condition is a strict relaxation of it, and so will be the execution rate. Since MC-Fluid is optimal in rate searching; i.e., whether there exist a feasible rate assignment, MC-Fluid will find it, we claim that it is sustainable with respect to criticality levels. □

Theorem 11. *MC-Fluid is sustainable w.r.t all input parameters.*

Proof. This follows from Lemmas 9 and 10. □

5. Related work

There is a significant body of research on mixed-criticality scheduling — we have briefly discussed several uniprocessor and multiprocessor scheduling algorithms in earlier sections of this (paper [8] has a comprehensive survey). However, to our knowledge this is the first work to address sustainability issues in mixed-criticality scheduling.

The formal concept of sustainability was introduced by Baruah and Burns [4], [7]. Prior work on this topic has focused upon sustainability analysis of periodic and sporadic (non-MC) task systems in uniprocessors, and demonstrated that numerous known schedulability tests in preemptive uniprocessor scheduling are not sustainable. The sustainability properties of global scheduling algorithms that employ sporadic task model such as EDF, Earliest-Deadline with Zero-Laxity and fixed priority scheduling are inspected against several parameters such as decreased execution time, later arrivals, and deadline relaxations in [2].

6. Conclusion

Sustainable schedulability tests ensure that a system that has been successfully verified will meet all its deadlines at run-time even if its operating parameters change for the better during system run-time. It has been argued [2], [7] that from an engineering perspective, sufficient and sustainable tests are more useful than exact but non-sustainable tests. Here we have analyzed, for the first time, the sustainability properties of a variety of widely studied mixed-criticality scheduling algorithms. While all are sustainable with respect to the parameters WCET, period, and deadline, which MC models "inherit" from traditional (i.e., non-MC) models, it turns out that Criticality-Monotonic and MC^2 schedulability analysis are not sustainable with respect to criticality level.

We have restricted attention here to sustainability of scheduling algorithms and schedulability tests where parameter changes occur prior to run-time. There is another aspect to sustainability, dealing with *dynamic* changes to parameters during run-time. It would be interesting to study sustainability properties of mixed-criticality scheduling algorithms under such a dynamic interpretation.

ACKNOWLEDGMENTS

The authors would like to thank Alan Burns and Haohan Li for the fruitful discussions. This research has been supported in parts by NSF grants CNS 1409175 and CPS 1446631, AFOSR grant FA9550-14-1-0161, ARO grant W911NF-14-1-0499, a startup grant from Missouri S&T, and a grant from the Intelligent System Center of Missouri S&T.

References

[1] N. C. Audsley. On priority assignment in fixed priority scheduling. *Information Processing Letters*, 79(1):39–44, 2001.

[2] T. P. Baker and S. K. Baruah. Sustainable multiprocessor scheduling of sporadic task systems. In *Proceedings of the 21st Euromicro Conference on Real-Time Systems (ECRTS)*, pages 141–150, 2009.

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

[4] S. Baruah and A. Burns. Sustainable scheduling analysis. In *Proceedings of the 27th IEEE Real-Time Systems Symposium (RTSS)*, pages 159–168, 2006.

[5] S. Baruah, A. Burns, and R. Davis. Response-time analysis for mixed criticality systems. In *Proceedings of the 32nd IEEE Real-Time Systems Symposium (RTSS)*, 2011.

[6] S. Baruah, H. Li, and L. Stougie. Towards the design of certifiable mixed-criticality systems. In *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2010.

[7] A. Burns and S. Baruah. Sustainability in real-time scheduling. *Journal of Computing Science and Engineering*, 2(1):74–97, 2008.

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

[9] A. Esper, G. Nelissen, V. Nélis, and E. 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)*, pages 139–148, 2015.

[10] P. Graydon and I. Bate. Safety assurance driven problem formulation for mixed-criticality scheduling. *Proceedings of the Workshop on Mixed Criticality (WMC)*, pages 19–24, 2013.

[11] 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 *Proceedings of the 35th IEEE Real-Time Systems Symposium (RTSS)*, pages 41–52. IEEE, 2014.

[12] M. S. Mollison, J. P. Erickson, J. H. Anderson, S. K. Baruah, and J. A. Scoredos. Mixed-criticality real-time scheduling for multicore systems. In *Proceedings of the 10th IEEE International Conference on Computer and Information Technology (CIT)*, pages 1864–1871, 2010.

[13] M. Paulitsch, O. M. Duarte, H. Karray, K. Mueller, D. Muench, and J. Nowotsch. Mixed-criticality embedded systems–a balance ensuring partitioning and performance. In *Proceedings of the 2015 Euromicro Conference on Digital System Design (DSD)*, pages 453–461, 2015.

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

[15] S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Proceedings of the 28th IEEE Real-Time Systems Symposium (RTSS)*, 2007.

Appendix

Proof of Theorem 8: The MC^2 scheduling algorithm is *not* sustainable with respect to criticality levels.

Proof. Consider the multi-criticality task-set shown in Table 3. We first show that the given example is MC^2-schedulable.

Calculating U_i^C for each task:

$$U_1^C = 1 - \left(\frac{1}{5} + \frac{1}{10} + \frac{6}{10} + \frac{1}{20}\right) = \frac{1}{20}$$

$$U_2^C = 1 - \left(\frac{3}{20} + \frac{2}{10} + \frac{2}{10} + \frac{1}{5}\right) = \frac{5}{20}$$

$$U_3^C = 1 - \left(\frac{3}{10} + \frac{3}{20} + \frac{2}{10} + \frac{1}{10}\right) = \frac{5}{20}$$

Substituting in Equation (25),

$$\frac{1}{20} + \frac{2}{20} + \frac{1}{20} = \frac{4}{20} < \frac{11}{20}$$

Substituting in Equation (26),

$$\frac{11}{20} - 2 \cdot \left(\frac{1}{20}\right) - \left(\frac{1}{20} + \frac{2}{20}\right) > 0$$

The tardiness is thus bounded at level C.

The next step is to decrease the criticality level of task τ_7 from B to C, and check if the schedulability condition (tardiness bounds) still holds. The schedulability conditions are given in Equations (25) and (26). On substituting the values from the table,

$$U_1^C = 1 - \left(\frac{1}{5} + \frac{1}{10} + \frac{1}{20}\right) = \frac{13}{20}$$

$$U_2^C = 1 - \left(\frac{3}{20} + \frac{2}{10} + \frac{2}{10} + \frac{1}{5}\right) = \frac{5}{20}$$

$$U_3^C = 1 - \left(\frac{3}{10} + \frac{3}{20} + \frac{2}{10} + \frac{1}{10}\right) = \frac{5}{20}$$

Substituting in Equation (25),

$$\frac{1}{20} + \frac{2}{20} + \frac{1}{20} + \frac{6}{10} = \frac{16}{20} < \frac{23}{20}$$

Substituting in Equation (26),

$$\frac{23}{20} - 2 \cdot \left(\frac{6}{10}\right) - \left(\frac{6}{10} + \frac{2}{20}\right) \not> 0$$

The second tardiness bound Condition (26) does not hold. Thus we conclude that the MC^2 scheduling policy is not sustainable w.r.t. criticality levels.

The example above illustrates the non-sustainability of MC^2 upon changes in criticality level – a task's criticality decreasing from level B to level C rendered a schedulable system unschedulable. We now provide some insight into why sustainability failed to hold in our example.

There are two conditions to demonstrate that the tardiness is bounded [12]. The first condition is:

$$\sum_{i:\chi_i=C} u_i^C \leq \sum_{k=1}^{m} (1 - u_{(k)}^{AB}), \tag{25}$$

where $\tau_{(k)}^{AB}$ denotes the set of tasks on processor *k* above level C, and $1 - u_{(k)}^{AB}$ is the available utilization on processor k after assigning level-A and level-B tasks. Thus, when changing the criticality level from B to C, the task is added to the level-C container (serving by all the processors). The increase in utilization available for level C tasks is proportional to the drop in $u_{(k)}^{AB}$ (utilization of tasks in level A and B). Thus the Equation (25) is always satisfied.

However, the second condition for the tardiness bound for level-C may not hold as we make such changes, which is originally given by:

$$\sum_{k=1}^{m} (1 - u_{(k)}^{AB}) > (m-1) \cdot \max_{i:\chi_i=C} u_i^C + U_{max(m-1)}^C. \tag{26}$$

Here $U_{max(m-1)}^C$ denotes the sum of the m-1 largest u_i^C values of tasks T_i which belong to task system τ. It is possible that the task of interest, whose criticality level is changed from B to C, has a maximum very high utilization value, such that the increase in right hand side of Equation (26) is much more significant than the gain on the left hand side (difference by *m-2* times its utilization), leading to a violation of the condition.

□

Task	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8	τ_9	τ_10	τ_11	τ_12	τ_13	τ_14	τ_15
Crit.	A	A	A	A	A	A	B	B	B	B	B	B	Global(C)	Global(C)	Global(C)
CPU	1	1	2	2	3	3	1	1	2	2	3	3	1	2	3
T_i	5	10	10	5	10	10	10	20	20	10	10	20	20	20	20
C_i^A	3	4	4	3	6	4	-	-	-	-	-	-	-	-	-
C_i^B	1	1	2	2	2	1	6	2	4	3	4	6	-	-	-
C_i^C	1	1	2	1	2	1	6	1	3	2	3	3	1	2	1

TABLE 3: A mixed-criticality task-set which is not sustainable under MC^2 scheduling policy.