EDF-Based Mixed-Criticality Scheduling with Graceful Degradation by Bounded Lateness

Kecheng Yang
Texas State University
yangk@txstate.edu

Zhishan Guo
University of Central Florida
zsguo@ucf.edu

***Abstract*—Mixed-criticality (MC) scheduling has been proposed for embedded real-time systems to alleviate the dilemma between runtime resource utilization and worst-case temporal guarantees for critical functions. The approach of dropping all low-criticality tasks upon a mode switch has been criticized for potentially over-degraded performance. In this paper, we focus on the graceful degradation for MC scheduling by providing bounded lateness for certain low-critical tasks. We define MCQOS-schedulability that massages the required bounded lateness into the definition of conventional MC-schedulability. Based on EDF-VD, we then propose algorithm EDF-VDS for systems to achieve MCQOS-schedulable. A utilization-based MCQOS-schedulability test and closed form lateness bounds are derived for EDF-VDS.**

***Index Terms*—real-time systems, mixed-criticality scheduling, graceful degradation, lateness bounds, server task**

I. Introduction

Due to size, weight, and power (SWaP) constraints, embedded system designs often demand high efficiency and resource utilization. On the other hand, real-time system designs often cap the system utilization, in order to provide provable worst-case temporal guarantees that might take overestimated pessimism into account. Worst-case execution time (WCET) estimates are one of the places where significant pessimism that is potentially overestimated could reside in.

To alleviate the dilemma between runtime resource utilization and worst-case temporal guarantees for critical functions, mixed-criticality (MC) scheduling has been proposed for embedded real-time systems [4, 30]. Under MC scheduling, each high-critical task may have a relatively optimistic WCET estimate in addition to the traditional most pessimistic one. During runtime, the scheduler monitors the execution time of tasks. If those high-critical ones always complete within the optimistic WCET estimate, then the scheduler would also schedule some low-critical tasks to execute and all tasks are guaranteed to meet their deadline. In contrast, if any high-critical task over-executes its optimistic WCET estimate (but does not execute for more than the most pessimistic one), then the scheduler immediately drops all the low-critical tasks in order to still guarantee hi-critical tasks meet their deadlines.

One criticism MC scheduling has received is that dropping the entire workloads of the low-critical tasks may lead to a sharp performance degradation [10]. Therefore, much recent work on MC scheduling attempts to address this issue to provide a more graceful degradation when some high-critical task over-execute its optimistic WCET estimate. In other words, even in the most pessimistic scenario, certain provable quality of service (QoS) for low-critical tasks, instead of entirely dropping low-critical workloads, is desirable.

Related Work. Much of prior work interpreted and provided the graceful degradation of the QoS of low-critical tasks in the forms of reducing execution time budget for every low-critical job or dropping a fraction of jobs for every low-critical task. In order to provide degraded performance for low-critical tasks, several prior works [7, 11, 17, 22, 26, 28] attempted to improve the performance of the low critical tasks on a more best-effort basis with no provable guarantees. Most of the works that do provide some guarantees to low critical tasks focused on providing QoS for low-critical tasks by a reduced execution budget for every low-critical task [3, 16, 20, 21]. [23] proposed a semi-partitioned algorithm for multiprocessors with full-service guarantees for low-critical tasks by migrations when criticality mode switch happens on only one of the processors. [15] studied guaranteed completion rates for low-critical tasks, while [14, 24, 25] provided probabilistic guarantees for them. [8] and [12] applied the concepts of weakly-hard constraints. [18] proposed a framework to drop tasks dynamically. [13] and [29] used Sigmoid-shaped utility functions to represent the QoS for real-time tasks.

Alternatively, graceful degradation can also be interpreted by bounded tardiness or lateness.[1] [5, 6] provided bounded tardiness guarantees for low-critical tasks. However, this work focused on fixed-priority-based scheduling and lacked a utilization-based schedulability test and closed-form tardiness bounds. In contrast, we focus on EDF-based scheduling in this paper and will present a utilization-based schedulability test and lateness bounds in closed form.

We allow a subset of low-critical tasks (the ones that have QoS requirements and/or benefits) to always execute with hi-critical tasks together, without reducing execution budget or dropping certain fractions of jobs, but only guarantee bounded lateness (which implies bounded response times) instead of meeting all deadlines for those low-critical tasks in the most pessimistic scenarios[2] (while all deadlines of high-critical tasks still must be met). Such a graceful degradation is favorable by tasks that have more resilience on accommodating an increased constant latency by buffering than suffering from constantly losing some execution (by reducing execution budget or dropping a fraction of jobs).

Contributions. In this paper, we focus on the graceful degradation for MC scheduling by providing bounded lateness for certain low-critical tasks. More formally, we define MCQOS-schedulability that massages the required bounded lateness into the definition of conventional MC-schedulability. We then propose our scheduler, called EDF-VDS,[3] for systems that require to be MCQOS-schedulable, and present our analysis

[1]The only difference between tardiness and lateness is that lateness can go negative. In scenarios where a deadline is missed, tardiness and lateness are exactly the same.

[2]Note that, a task that cannot live with any lateness and require all deadlines to be met even in the most pessimistic scenario should be categorized as a high-critical task.

[3]"EDF-VDS" stands for "earliest-deadline-first with virtual deadlines in the LO-mode and a QoS server task in the HI-mode."

and a sufficient MCQOS-schedulability test for EDF-VDS. Later on, we comment on several interesting issues related to MCQOS-schedulability and EDF-VDS.

Organization. In the rest of the paper, we introduce our system model (Section II), review algorithm EDF-VD (Section III), present our proposed algorithm EDF-VDS and its schedulability test (Section IV), discuss related issues (Section V), and conclude (Section VI).

II. System Model

In this paper, we consider the uniprocessor preemptive scheduling of the classic and widely studied dual-criticality MC implicit-deadline sporadic task model [1, 2]. Each task τ_i in the task set $\mathcal{T}$ is either of high(HI) or low (LO) criticality, and we denote the set of high-criticality tasks (HI-tasks) by $\mathcal{T}_{\text{HI}}$ and the set of low-criticality tasks (LO-tasks) by $\mathcal{T}_{\text{LO}}$. That is, $\mathcal{T}_{\text{HI}} \cup \mathcal{T}_{\text{LO}} = \mathcal{T}$ and $\mathcal{T}_{\text{HI}} \cap \mathcal{T}_{\text{LO}} = \emptyset$. Each (HI- or LO-) task $\tau_i \in \mathcal{T}$ releases a sequence of (potentially infinite) *jobs*, with a minimum separation of T_i time units between consecutive job releases. Each job of task τ_i may execute for up to its worst-case execution time (WCET) to complete. Each HI-task $\tau_i \in \mathcal{T}_{\text{HI}}$ has two WCET estimates—a most pessimistic one (with the larger value) denoted by C_i^{HI} and a more optimistic one (with the smaller value) denoted by C_i^{LO}; in contrast, each LO-task τ_i has only a single (more optimistic) WCET estimate, which is denoted by C_i. Every job (no matter of a HI- or LO-task) has an absolute deadline at T_i time units after its release. For any job J that has an absolute deadline at time d and finishes its execution at time f, we define its *lateness* by $f - d$. Note that bounded lateness implies bounded response time, and, in particular, non-positive lateness (*i.e.*, bounded by 0) implies that the deadline is met.

A job is called *pending* if it is released but not finished. We also define the lateness of a task by the maximum lateness among all its jobs. In addition, we may call a job of a HI-task as a HI-job and a job of a LO-task as a LO-job for short.

In short, a HI-task τ_i is characterized by three parameters $(C_i^{\text{LO}}, C_i^{\text{HI}}, T_i)$, whereas a LO-task τ_i is characterized by two parameters (C_i, T_i). We also define the HI- and LO-utilizations of a HI-task τ_i by $u_i^{\text{HI}} = C_i^{\text{HI}}/T_i$ and $u_i^{\text{LO}} = C_i^{\text{LO}}/T_i$, respectively; and define the (LO-) utilization of a LO-task τ_i by $u_i = C_i/T_i$. Furthermore, $U_{\text{HI}}^{\text{HI}}$, $U_{\text{HI}}^{\text{LO}}$, and U_{LO} are defined as follows:

$$U_{\text{HI}}^{\text{HI}} = \sum_{\tau_i \in \mathcal{T}_{\text{HI}}} u_i^{\text{HI}}, \quad U_{\text{HI}}^{\text{LO}} = \sum_{\tau_i \in \mathcal{T}_{\text{HI}}} u_i^{\text{LO}}, \quad U_{\text{LO}} = \sum_{\tau_i \in \mathcal{T}_{\text{LO}}} u_i.$$

Please note that, for a system to be feasible, the following conditions must hold; otherwise, the system might be overutilized.

$$U_{\text{LO}} + U_{\text{HI}}^{\text{LO}} \leq 1 \tag{1}$$

$$U_{\text{HI}}^{\text{HI}} \leq 1 \tag{2}$$

MC-schedulability. In order to define the correctness criteria of this work, we first give a widely-used MC schedulability definition—later on, a different yet similar concept will be introduced based on this. In the literature of dual-criticality MC scheduling, the MC-schedulability of a system is often defined as follows, by whether and which of the WCET estimates remain true during runtime.

1) If all job complete within its more optimistic estimates (*i.e.*, C_i^{LO} for a HI-task or C_i for a LO-task), then the deadlines for all (HI- and LO-) tasks must be met.
2) If some job(s) of some HI-task(s) need to execute for more than their optimistic estimate (C_i^{LO}) but can still complete within the most pessimistic estimate (C_i^{HI}), then the deadlines for all HI-tasks must be met.
3) If any job of a HI-task τ_i needs to execute for more than C_i^{HI} time units or any job of a LO-task τ_i needs to execute for more than C_i time units, then the system is considered as erroneous and no temporal guarantee would be provided.[4]

In light of the above definition of MC-schedulability, many MC schedulers would monitor the execution time of each job and consider a *mode switch* when a job of some HI-task τ_i has executed for C_i^{HI} without signaling completion, and the mode before and after the mode switch is often called the LO-mode and the HI-mode, respectively. Upon the mode switch, the scheduler immediately drops any incomplete and to-be-released jobs of all LO-tasks, as they are not a criterion for the MC-schedulability anymore. That is, no temporal guarantee of any LO-task will be provided in the HI-mode.

QOS task set $\mathcal{T}_{\text{QOS}}$. In order to reduce the performance gap upon the mode switch and provide some temporal guarantees for some LO-tasks to achieve certain provable QoS, we introduce a new set denoted by $\mathcal{T}_{\text{QOS}}$, which is a subset of $\mathcal{T}_{\text{LO}}$, *i.e.*, $\mathcal{T}_{\text{QOS}} \subseteq \mathcal{T}_{\text{LO}}$. The set $\mathcal{T}_{\text{QOS}}$ should be specified by the system designer, according to which LO-tasks are meaningful and/or desirable to be provided certain QoS in the form of bounded lateness, regardless of the behaviors of HI-criticality tasks. We may call the LO-tasks in $\mathcal{T}_{\text{QOS}}$ as QOS-tasks and call a job of a QOS-task as a QOS-job for short. We also define $U_{\text{QOS}} = \sum_{\tau_i \in \mathcal{T}_{\text{QOS}}} u_i$, which must satisfy the following in order to be meaningful:

$$0 < U_{\text{QOS}} < 1. \tag{3}$$

To see this, $U_{\text{QOS}} = 0$ implies no QOS-task in the system at all; $U_{\text{QOS}} > 1$ implies that the system is overutilized and therefore is not feasible; and $U_{\text{QOS}} = 1$, together with (1) and the fact that $\mathcal{T}_{\text{QOS}}$ is a subset of $\mathcal{T}_{\text{LO}}$, implies no HI-task in the system at all and all deadlines of the only LO-tasks must be met by applying an ordinary preemptive EDF scheduler.

MCQOS-schedulability. With the introduction of task set $\mathcal{T}_{\text{QOS}}$, we define the following MCQOS-schedulability, which dominates the conventional MC-schedulability discussed earlier (*i.e.*, any system that is MCQOS-schedulable must also be MC-schedulable).

1) If all job complete within its more optimistic estimates (*i.e.*, C_i^{LO} for a HI-task or C_i for a LO-task), then the deadlines for all (HI- and LO-) tasks must be met.
2) If some job(s) of some HI-task(s) need to execute for more than their optimistic estimate (C_i^{LO}) but can still complete within the most pessimistic estimate (C_i^{HI}), then the deadlines for all HI-tasks must be met ***and all*** QOS-***tasks have bounded lateness***.
3) If any job of a HI-task τ_i needs to execute for more than C_i^{HI} time units, or any job of a LO-task τ_i needs to

[4]Alternatively, instead of being erroneous, any such jobs could be immediately terminated and considered as completed.

execute for more than C_i time units, then the system is considered as erroneous and no temporal guarantee would be provided.[4]

Thus, in contrast to the MC schedulers discussed earlier that tends to drop all tasks in $\mathcal{T}_{\text{LO}}$ upon the mode switch, an MCQOS scheduler would only drop tasks in $\mathcal{T}_{\text{LO}} \setminus \mathcal{T}_{\text{QOS}}$ (LO but not QOS) while must allow tasks in $\mathcal{T}_{\text{QOS}}$ to continue their (unfinished) execution, release new jobs in the HI-mode, and experience bounded lateness (that is provable).

III. Algorithm EDF-VD

In this section, we review the scheduler EDF-VD, which was first proposed in [1, 2] to validate the MC-schedulability. Therefore, there is no task set $\mathcal{T}_{\text{QOS}}$ but only task sets $\mathcal{T}_{\text{HI}}$ and $\mathcal{T}_{\text{LO}}$ under EDF-VD. Later on, in Section IV, we will discuss how to build a new scheduler EDF-VDS, which is based on EDF-VD, to cope with the introduction of $\mathcal{T}_{\text{QOS}}$.

We first provide an overview describing the preprocessing that is done by EDF-VD, then we discuss the manner in which it makes run-time scheduling decisions.

Given the MC implicit-deadline sporadic task set $\mathcal{T}$ to be scheduled on a preemptive uniprocessor, prior to run-time, EDF-VD performs a schedulability test to determine whether τ can be successfully scheduled by it or not. If $\mathcal{T}$ is deemed schedulable, then a virtual deadline ($T_i^{'} \leq T_i$) is computed for each HI-criticality task. For simplicity, one may choose to calculate one system-level deadline shrinking parameter $x \in (0, 1]$ for all HI tasks; i.e., $T_i^{'} = T_i \times x, \forall \tau_i \in \tau_{\text{HI}}$. Those virtual deadlines will be used for the purposes of determining (EDF) scheduling priority under the LO-mode.

During run time, suppose that a job of task τ_i arrives at time instant t. If τ_i is a LO-task, then this job is assigned a virtual deadline same as its actual deadline, $t + T_i$; whereas if this is a HI-task, it is assigned a virtual deadline of $t_0 + T_i^{'}$. If some job executes for a duration exceeding its HI-criticality WCET without signaling that it has completed execution, the system triggers a mode switch. In response, all LO-jobs are immediately discarded and subsequent execution of HI-criticality tasks (including the jobs that are currently active) continue to be scheduled according to EDF. But the actual job deadlines (release time plus period) are used on and beyond the mode switch point.

As mentioned earlier in this section, one may choose to shrink all HI-criticality deadlines by the same factor x. Baruah *et al.* [1, 2] proposed the following assignment for this factor:

$$x = \frac{U_{\text{HI}}^{\text{LO}}}{1 - U_{\text{LO}}}. \tag{4}$$

They also provided a utilization-based sufficient MC-schedulability test accordingly:

$$xU_{\text{LO}} + U_{\text{HI}}^{\text{HI}} \leq 1. \tag{5}$$

These results serve as a basis of our approach and analysis.

IV. Algorithm EDF-VDS

Based on algorithm EDF-VD, we design our proposed algorithm, called EDF-VDS. In addition to EDF-VD which was proposed for systems that need to be MC-schedulable, EDF-VDS further addresses the systems where MCQOS-schedulability is desirable or required.

In the LO-mode, EDF-VDS is identical to EDF-VD. Note the fact that there is in fact no difference between MC-schedulability and MCQOS-schedulability in the LO-mode, so it is clear that the analysis and MC-schedulability test for EDF-VD in the LO-mode applies to EDF-VDS as well. Thus, we focus on the HI-mode in the rest of this section, and assume that all deadlines before the mode switch have been met by checking the MC-schedulability test for EDF-VD.

Time interval $[t_1, t_2)$. Let t_1 denote the time instant of the mode switch. At time t_1, EDF-VDS drops the LO-tasks in $\mathcal{T}_{\text{LO}} \setminus \mathcal{T}_{\text{QOS}}$ and "hold" the LO-tasks in $\mathcal{T}_{\text{QOS}}$, or QOS-tasks. The QOS-tasks are not dropped and allowed to continue to release new jobs, but they are not to be scheduled by EDF-VDS until time instant t_2 (to be defined later). Instead, EDF-VDS schedules the HI-tasks by EDF according to their actual deadlines until time instant t_2, which is defined as the first time instant such that all HI-criticality jobs released before time instant t_2 have completed their execution by time instant t_2. That is, intuitively t_2 is the first "HI-task idle" instant after the mode switch.

Server task τ_{QOS}. At time instant t_2, EDF-VDS introduces "server task" τ_{QOS}, which *periodically* releases "server jobs" starting from time instant t_2 and with a period T_{QOS}. Each "server job" has an *exact* execution time $C_{\text{QOS}} = U_{\text{QOS}} * T_{\text{QOS}}$, which ensures that τ_{QOS} has a utilization of U_{QOS}. Recall that U_{QOS} is defined as the total utilization of the tasks in $\mathcal{T}_{\text{QOS}}$. *i.e.*, $U_{\text{QOS}} = \sum_{\tau_i \in \mathcal{T}_{\text{QOS}}} u_i$. Starting from time instant t_2, the "server task" τ_{QOS} is scheduled with all the HI-tasks together by EDF, where τ_{QOS} is also assigned implicit deadlines (*i.e.*, T_{QOS} after each release). Whenever the server task τ_{QOS} is scheduled, EDF-VDS schedules the pending job of any QOS-tasks with the earliest deadline if there is any, or leave the processor idle (but $\mathcal{T}_{\text{QOS}}$ is still viewed as executing, *i.e.*, the "budget" drains) if no pending job from QOS-tasks exists.

Parameter T_{QOS}. In EDF-VDS, T_{QOS} is actually a tunable parameter. The system designer can tune and pick arbitrary positive values for T_{QOS} as a tradeoff between the lateness bounds that can be guaranteed and runtime overheads. As we will see later, a smaller T_{QOS} would lead to lower lateness bounds while might induce more runtime scheduling overheads (*e.g.*, more preemptions).

Return to LO-mode. Note that in much prior work on MC-schedulability, t_2 would actually be the time instant for the scheduler to switch back to the LO-mode. This is because all the LO-tasks are dropped in the HI-mode when considering MC-schedulability only. As a result, "HI-task idle" instant is the truly idle time instant. When considering MCQOS-schedulability, in addition to the HI-tasks, QOS-tasks also need to execute in the HI-mode. Therefore, the truly idle time instant, *i.e.*, the time instant that can safely allow scheduler EDF-VDS switch back to the LO-mode, would be the first time instant where all jobs of HI-tasks *and* QOS-*tasks* released before this time instant have completed their execution.

A. Meeting Deadlines of HI-tasks

EDF-VDS is designed based on EDF-VD, and the need to schedule τ_{QOS} together is the only difference that could have an impact on the schedule of HI-tasks. Here we show that this

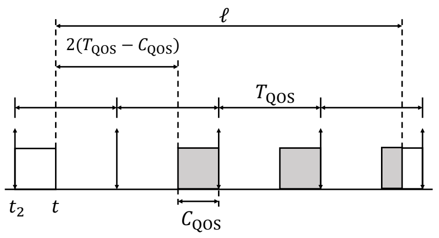

Fig. 1: Minimum supply by τ_{QOS} during any time interval after time t_2 and of length ℓ.

difference between EDF-VDS and EDF-VD will not cause any deadline miss for HI-tasks.

Theorem 1. *If a system is MC-schedulable under EDF-VD and $U_{\text{HI}}^{\text{HI}} + U_{\text{QOS}} \leq 1$, then all deadlines of* HI*-tasks will be met in the* HI*-mode under EDF-VDS.*

Proof. As a matter of fact, EDF-VDS has exactly the same behaviour as EDF-VD in the LO-mode and during time interval $[t_1, t_2)$ in the HI-mode. Therefore, the system being MC-schedulable under EDF-VD implies that all deadlines of HI-tasks in the time interval $[t_1, t_2)$ (and those in the LO-mode as well) will be met. By the definition of time t_2, all HI-jobs that were released before time t_2 must have completed at the time instant t_2. In addition, τ_{QOS} in EDF-VDS does not release any job until time t_2. Therefore, the schedule starting from time t_2 is equivalent to an ordinary EDF schedule for a set of ordinary sporadic tasks from time zero, and such sporadic task set has total utilization $U_{\text{HI}}^{\text{HI}} + U_{\text{QOS}}$, which is at most 1 as the lemma states. Thus, by the classic schedulability results for preemptive EDF on a uniprocessor [19], starting from t_2, all deadlines of HI-tasks and τ_{QOS} must be met, and the lemma follows. ☐

B. Lateness Bounds for QOS-tasks

In the HI-mode, the QOS-tasks execute their jobs based on the processor *supply* that the server task τ_{QOS} provides. In this context, we define the *supply* by the number of time units in which τ_{QOS} is scheduled under EDF-VDS. Note that the last part of the proof for Theorem 1 above has in fact also proves that every job of the server task τ_{QOS} must meet its deadline given that $U_{\text{HI}}^{\text{HI}} + U_{\text{QOS}} \leq 1$, which we assume for the rest of this section. As a result, starting from time t_2, the supply provided by τ_{QOS} follows the periodic resource model [27].

As shown in [27], Figure 1 illustrates the scenario for a periodic resource to have minimum supply for an arbitrary time interval of length ℓ. This result also yields the following linear lower bound on the supply for an arbitrary time interval of length ℓ. Please note that $T_{\text{QOS}} - C_{\text{QOS}} = (1 - U_{\text{QOS}})T_{\text{QOS}}$.

Property 1. *For any time instant $t \geq t_2$, the supply provided by τ_{QOS} within the time interval $[t, t + \ell)$ is at least $(\ell - 2(1 - U_{\text{QOS}})T_{\text{QOS}})U_{\text{QOS}}$ for all ℓ.*

The above property applies to any *arbitrary* time interval of length ℓ. Nonetheless, for such time intervals that begin exactly at time t_2, more supply can actually be guaranteed. For a time interval of length ℓ starting from time t_2, the scenario in Figure 1 cannot happen, because the first period of the server task τ_{QOS} must start right at time t_2. In contrast, Figure 2 illustrates the scenario that yields minimum supply for a time interval of length ℓ starting from time t_2, and as a result, it has the following linear lower bound.

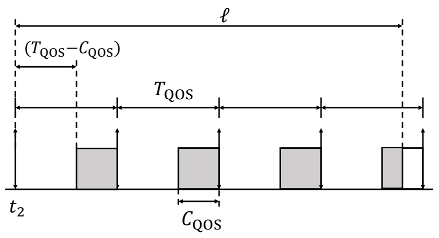

Fig. 2: Minimum supply by τ_{QOS} during time interval $[t_2, t_2 + \ell)$.

Property 2. *The supply provided by τ_{QOS} within the time interval $[t_2, t_2 + \ell)$ is at least $(\ell - (1 - U_{\text{QOS}})T_{\text{QOS}})U_{\text{QOS}}$ for all ℓ.*

The above properties provide a basis to derive the minimum processor supply the QOS-tasks would receive during a certain time interval, and then we are able to bound the lateness for the QOS-tasks. We focus on an arbitrary job J of some QOS-task and let time instants d and f denote the absolute deadline and complete time of J, respectively. Also, please note that we are only interested in such QOS-job J that has not completed by the mode switch and has a deadline after the mode switch, *i.e.*, $d \geq t_1$ and $f \geq t_1$ (because all deadlines before the mode switch have been met). We then derive lateness bounds for J (*i.e.*, bounding $f - d$) in two cases for whether a QOS-*idle* time instant exists in the time interval $[t_2, f)$.

QOS-idle time instant. We say a time instant t is QOS-idle if and only if τ_{QOS} is scheduled by EDF-VDS at time t but there is no pending QOS-job with a deadline at or before d at time t (so that the processor is left physically idling or executing some QOS-job with a deadline after d).

Lemma 1. *If some* QOS-idle *time instant exists in the time interval $[t_2, f)$, then*

$$f - d \leq 2(1 - U_{\text{QOS}})T_{\text{QOS}}. \tag{6}$$

Proof. We let $s \geq t_2$ denote the latest such time instant. Then, there must be some pending QOS-job with a deadline at or before d at every time instant in $[s, f)$; otherwise, either s would have been a later time or J would have finished earlier than time f. Therefore, within time interval $[s, f)$, some QOS-job with a deadline at or before d is being executed at every time instant whenever $\mathcal{T}_{\text{QOS}}$ is scheduled by EDF-VDS, and we let W denote the total length of time for $\mathcal{T}_{\text{QOS}}$ is scheduled within $[s, f)$. Then,

$$\begin{aligned} W &\leq \sum_{\tau_i \in \mathcal{T}_{\text{QOS}}} \left(\left\lfloor \frac{d-s}{T_i} \right\rfloor C_i \right) \\ &\leq \sum_{\tau_i \in \mathcal{T}_{\text{QOS}}} \left(\left(\frac{d-s}{T_i} \right) C_i \right) \\ &= (d-s) \sum_{\tau_i \in \mathcal{T}_{\text{QOS}}} \frac{C_i}{T_i} \\ &= (d-s) \sum_{\tau_i \in \mathcal{T}_{\text{QOS}}} U_i \\ &= (d-s)U_{\text{QOS}}, \end{aligned} \tag{7}$$

because by the definition of s, any QOS-job released before s and with a deadline at or before d must have completed its execution by s, and at or after time s, each task τ_i can release at most $\left\lfloor \frac{d-s}{T_i} \right\rfloor$ jobs with deadlines at or before d.

On the other hand, by Property 1,

$$W \geq ((f-s) - 2(1-U_{\text{QOS}})T_{\text{QOS}})U_{\text{QOS}}. \tag{8}$$

By (7) and (8),

$$((f-s) - 2(1-U_{\text{QOS}})T_{\text{QOS}})U_{\text{QOS}} \leq (d-s)U_{\text{QOS}},$$

which implies

$$f - d \leq 2(1-U_{\text{QOS}})T_{\text{QOS}}.$$

The lemma follows. □

Lemma 2. *If no* QOS-idle *time instant exists in the time interval* $[t_2, f)$*, then*

$$f-d \leq \frac{2\sum_{\tau_i \in \mathcal{T}_{\text{HI}}} C_i^{\text{HI}}}{1-U_{\text{HI}}^{\text{HI}}} + \frac{\sum_{\tau_i \in \mathcal{T}_{\text{QOS}}} C_i}{U_{\text{QOS}}} + (1-U_{\text{QOS}})T_{\text{QOS}}. \tag{9}$$

Proof. By the definition of t_2, it is clear that no time instant within time interval $[t_1, t_2)$ has τ_{QOS} scheduled. Also, the lemma states that no QOS-idle time instant exists in the time interval $[t_2, f)$. Therefore, within time interval $[t_1, f)$, some QOS-job with a deadline at or before d is being executed at every time instant whenever $\mathcal{T}_{\text{QOS}}$ is scheduled by EDF-VDS, and we let W' denote the total length of time for $\mathcal{T}_{\text{QOS}}$ is scheduled within $[t_1, f)$. For each QOS-task τ_i, all of its jobs with deadline before t_1 must have completed by time instant t_1 because all deadlines have been met in the LO-mode; at most one job of τ_i is released before t_1 but has a deadline at or after t_1; and at or after time t_1, τ_i can release at most $\left\lfloor \frac{d-t_1}{T_i} \right\rfloor$ jobs with deadlines at or before d. Thus, we have

$$\begin{aligned} W' &\leq \sum_{\tau_i \in \mathcal{T}_{\text{QOS}}} \left(\left(1 + \left\lfloor \frac{d-t_1}{T_i} \right\rfloor \right) C_i \right) \\ &\leq \sum_{\tau_i \in \mathcal{T}_{\text{QOS}}} \left(\left(1 + \frac{d-t_1}{T_i} \right) C_i \right) \\ &= \sum_{\tau_i \in \mathcal{T}_{\text{QOS}}} C_i + (d-t_1) \sum_{\tau_i \in \mathcal{T}_{\text{QOS}}} \frac{C_i}{T_i} \\ &= \sum_{\tau_i \in \mathcal{T}_{\text{QOS}}} C_i + (d-t_1) \sum_{\tau_i \in \mathcal{T}_{\text{QOS}}} U_i \\ &= \sum_{\tau_i \in \mathcal{T}_{\text{QOS}}} C_i + (d-t_1) U_{\text{QOS}}. \end{aligned} \tag{10}$$

On the other hand, since t_2 is within time interval $[t_1, f)$, by Property 2, we have

$$W' \geq ((f-t_2) - (1-U_{\text{QOS}})T_{\text{QOS}})U_{\text{QOS}}. \tag{11}$$

By (10) and (11),

$$((f-t_2) - (1-U_{\text{QOS}})T_{\text{QOS}})U_{\text{QOS}} \leq \sum_{\tau_i \in \mathcal{T}_{\text{QOS}}} C_i + (d-t_1)U_{\text{QOS}},$$

which, by (3), implies

$$f-d \leq t_2 - t_1 + \frac{\sum_{\tau_i \in \mathcal{T}_{\text{QOS}}} C_i}{U_{\text{QOS}}} + (1-U_{\text{QOS}})T_{\text{QOS}}. \tag{12}$$

Bounding $t_2 - t_1$. Given (12) above, we only need to bound $t_2 - t_1$ so that $f-d$ will be bounded, too. For each HI-task τ_i, it can have at most one job released before the mode switch (*i.e.*, time instant t_1) but with a deadline after t_1, and can release at most $\left\lceil \frac{t_2-t_1}{T_i} \right\rceil$ jobs (no matter with deadlines at, before, or after t_2) within time interval $[t_1, t_2)$. Moreover, by the definition of time instant t_2, some HI-job must be executing at every time instant in $[t_1, t_2)$. Because only released jobs can be scheduled to execute, it must hold that

$$\begin{aligned} t_2 - t_1 &\leq \sum_{\tau_i \in \mathcal{T}_{\text{HI}}} \left(\left(1 + \left\lceil \frac{t_2-t_1}{T_i} \right\rceil \right) C_i^{\text{HI}} \right) \\ &\leq \sum_{\tau_i \in \mathcal{T}_{\text{HI}}} \left(\left(2 + \frac{t_2-t_1}{T_i} \right) C_i^{\text{HI}} \right) \\ &= 2 \sum_{\tau_i \in \mathcal{T}_{\text{HI}}} C_i^{\text{HI}} + (t_2-t_1) \sum_{\tau_i \in \mathcal{T}_{\text{HI}}} \frac{C_i^{\text{HI}}}{T_i} \\ &= 2 \sum_{\tau_i \in \mathcal{T}_{\text{HI}}} C_i^{\text{HI}} + (t_2-t_1) U_{\text{HI}}^{\text{HI}}, \end{aligned}$$

which implies

$$t_2 - t_1 \leq \frac{2\sum_{\tau_i \in \mathcal{T}_{\text{HI}}} C_i^{\text{HI}}}{1-U_{\text{HI}}^{\text{HI}}}, \tag{13}$$

because $U_{\text{HI}}^{\text{HI}} + U_{\text{QOS}} \leq 1$ and (3) imply $U_{\text{HI}}^{\text{HI}} < 1$. By combining (12) and (13), the lemma follows. □

Given that Lemma 1 and Lemma 2 have been proven and the fact that QOS-job J was chosen arbitrarily, it is straightforward to show the following theorem, which provides a lateness bound for any QOS-task.

Theorem 2. *If a system is MC-schedulable under EDF-VD and* $U_{\text{HI}}^{\text{HI}} + U_{\text{QOS}} \leq 1$*, then the lateness of any* QOS*-task in the* HI*-mode under EDF-VDS will not greater than*

$$(1-U_{\text{QOS}})T_{\text{QOS}} \quad + \\ \max\left\{ (1-U_{\text{QOS}})T_{\text{QOS}}, \frac{2\sum_{\tau_i \in \mathcal{T}_{\text{HI}}} C_i^{\text{HI}}}{1-U_{\text{HI}}^{\text{HI}}} + \frac{\sum_{\tau_i \in \mathcal{T}_{\text{QOS}}} C_i}{U_{\text{QOS}}} \right\}.$$

Proof. Since the "if" conditions in Lemma 1 and Lemma 2 are complementary, one of these two lemmas must apply to any arbitrary QOS-job J with a deadline in the HI-mode. That is, the lateness must be bounded by (6) or (9). Thus, with some rearrangement, the theorem follows. □

MCQOS-schedulability test. By the MC-schedulability test for EDF-VD presented in Section III and Theorem 2 above, we can conclude the following sufficient utilization-based MCQOS-schedulability test for EDF-VDS.

Theorem 3. *A system with given* $\mathcal{T}_{\text{HI}}$, $\mathcal{T}_{\text{LO}}$*, and* $\mathcal{T}_{\text{QOS}} \subseteq \mathcal{T}_{\text{LO}}$ *is MCQOS-schedulable under EDF-VDS if*

$$\frac{U_{\text{HI}}^{\text{LO}}}{1-U_{\text{LO}}} U_{\text{LO}} + U_{\text{HI}}^{\text{HI}} \leq 1 \quad \textit{and} \quad U_{\text{HI}}^{\text{HI}} + U_{\text{QOS}} \leq 1.$$

V. Discussions

In Section IV, we just presented algorithm EDF-VDS and its MCQOS-schedulability test. In this section, we discuss several issues or questions that might arise.

Necessity of $U_{\text{HI}}^{\text{HI}} + U_{\text{QOS}} \leq 1$. Compared to the MC-schedulability conditions for EDF-VD, the only additional condition for EDF-VDS required to achieve MCQOS-schedulability is $U_{\text{HI}}^{\text{HI}} + U_{\text{QOS}} \leq 1$. So, one question might be: is it necessary? The following theorem shows that the answer is "Yes."

Theorem 4. *Any system such that $U_{\text{HI}}^{\text{HI}} + U_{\text{QOS}} > 1$ cannot be MCQOS-schedulable under any algorithm.*

Proof. $U_{\text{HI}}^{\text{HI}} + U_{\text{QOS}} > 1$ implies overutilization in the HI-mode, because no job of any task in $\mathcal{T}_{\text{HI}} \cup \mathcal{T}_{\text{QOS}}$ is ever dropped at all. Therefore, at least one task in $\mathcal{T}_{\text{HI}} \cup \mathcal{T}_{\text{QOS}}$ must have its lateness increase without bound regardless the scheduling algorithm, and this directly indicates the system is not MCQOS-schedulable no matter the task with unbounded lateness is a HI-task or QOS-task. □

Implication by $\mathcal{T}_{\text{QOS}} = \mathcal{T}_{\text{LO}}$. Another question that might arise is: how large the set $\mathcal{T}_{\text{QOS}} \subseteq \mathcal{T}_{\text{LO}}$ could be? Can every LO-task be included in $\mathcal{T}_{\text{QOS}}$, *i.e.*, $\mathcal{T}_{\text{QOS}} = \mathcal{T}_{\text{LO}}$, so that every LO-task is guaranteed bounded lateness? It should not be a problem, provided that the condition for MCQOS-schedulability in Theorem 3 hold. However, given that $U_{\text{HI}}^{\text{HI}} + U_{\text{QOS}} \leq 1$ is necessarily required, $\mathcal{T}_{\text{QOS}} = \mathcal{T}_{\text{LO}}$ implies that $U_{\text{HI}}^{\text{HI}} + U_{\text{LO}} \leq 1$. Thus, all deadlines must be met when apply an ordinary EDF scheduler to the entire task set $\mathcal{T}$ from the very beginning, and none of MC scheduling, virtual deadlines, mode switch, or server task is needed. Therefore, the cases where $\mathcal{T}_{\text{QOS}}$ is a *proper* subset of $\mathcal{T}_{\text{LO}}$, *i.e.*, $\mathcal{T}_{\text{QOS}} \subset \mathcal{T}_{\text{LO}}$, would be a more applicable domain for this research.

Extension to cope with entire EDF-VD family. Note that, EDF-VDS has exactly the same behavior as EDF-VD until time instant t_2 and the schedule at or after t_2 is irrelevant to any setting of the virtual deadlines. Thus, the approach we proposed here to construct EDF-VDS very likely has the potential to be extended to any virtual-deadline-based conventional MC scheduler (*e.g.*, [9]). In this paper, we chose the original EDF-VD scheduler to illustrate our approach for simplicity and clarity. We defer the potential extensions to more complicated virtual-deadline-based MC schedulers to future work.

VI. Conclusion

In this paper, we focused on the graceful degradation for MC scheduling by providing bounded lateness for certain low-critical tasks, for which we defined MCQOS-schedulability. We proposed a new scheduler EDF-VDS for systems to achieve MCQOS-schedulable. Furthermore, we derived a utilization-based MCQOS-schedulability test and closed form lateness bounds for EDF-VDS. Finally, we discussed the necessity of a condition in our MCQOS-schedulability test and the potential of this approach to be extended.

Acknowledgment

This work is supported by NSF grant CNS-1850851, a start-up grant from the University of Central Florida, and start-up and REP grants from Texas State University.

References

[1] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, S. Van Der Ster, and L. Stougie. The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems. In *24th ECRTS*, 2012.

[2] S. Baruah, V. Bonifaci, G. D'Angelo, A. Marchetti-Spaccamela, S. Van Der Ster, and L. Stougie. Mixed-criticality scheduling of sporadic task systems. In *19th ESA*, 2011.

[3] S. Baruah, A. Burns, and Z. Guo. Scheduling mixed-criticality systems to guarantee some service under all non-erroneous behaviors. In *28th ECRTS*, 2016.

[4] S. Baruah and S. Vestal. Schedulability analysis of sporadic tasks with multiple criticality specifications. In *20th ECRTS*, 2008.

[5] G. von der Bruggen, L. Schonberger, and J. Chen. Systems with dynamic real-time guarantees in uncertain and faulty execution environments. In *37th RTSS*, 2016.

[6] G. von der Bruggen, L. Schonberger, and J. Chen. Do nothing, but carefully: Fault tolerance with timing guarantees for multiprocessor systems devoid of online adaptation. In *23rd PRDC*, 2018.

[7] A Burns and S. Baruah. Towards a more practical model for mixed criticality systems. In *1st WMC*, 2013.

[8] H Choi, H Kim, and Q. Zhu. Job-class-level fixed priority scheduling of weakly-hard real-time systems. In *25th RTAS*, 2019.

[9] P. Ekberg and W. Yi. Bounding and shaping the demand of generalized mixed-criticality sporadic task systems. *Real-Time Systems*, 50:48–86, 2014.

[10] R. Ernst and M. Di Natale. Mixed criticality systems—a history of misconceptions? *IEEE Design & Test*, 33(5):65–74, 2016.

[11] T. Fleming and A. Burns. Incorporating the notion of importance into mixed criticality systems. In *2nd WMC*, 2014.

[12] O. Gettings, S. Quinton, and R. Davis. Mixed criticality systems with weakly-hard constraints. In *23rd RTNS*, 2015.

[13] Z. Guo and S. Baruah. A neurodynamic approach for real-time scheduling via maximizing piecewise linear utility. *IEEE transactions on neural networks and learning systems*, 27(2):238–248, 2016.

[14] Z. Guo, L. Santinelli, and K. Yang. EDF schedulability analysis on mixed-criticality systems with permitted failure probability. In *21st RTCSA*, 2015.

[15] Z. Guo, K. Yang, S. Vaidhun, S. Arefin, S. K. Das, and H. Xiong. Uniprocessor mixed-criticality scheduling with graceful degradation by completion rate. In *39th RTSS*, 2018.

[16] L. Huang, I.-H. Hou, S. Sapatnekar, and J. Hu. Graceful degradation of low-criticality tasks in multiprocessor dual-criticality systems. In *26th RTNS*, 2018.

[17] M. Jan, L. Zaourar, and M. Pitel. Maximizing the execution rate of low-criticality tasks in mixed criticality systems. In *1st WMC*, 2013.

[18] Jaewoo Lee, Hoon Sung Chwa, Linh TX Phan, Insik Shin, and Insup Lee. Mc-adapt: Adaptive task dropping in mixed-criticality scheduling. *ACM Transactions on Embedded Computing Systems*, 16(5s):163, 2017.

[19] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 30(1):46–61, 1973.

[20] 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 *37th RTSS*, 2016.

[21] R. Pathan. Improving the quality-of-service for scheduling mixed-criticality systems on multiprocessors. In *29th ECRTS*, 2017.

[22] R. Pathan. Improving the schedulability and quality of service for federated scheduling of parallel mixed-criticality tasks on multiprocessors. In *30th ECRTS*, 2018.

[23] S. Ramanathan and A. Easwaran. Mixed-criticality scheduling on multiprocessors with service guarantees. In *21st ISORC*, 2018.

[24] L. Santinelli and Z. Guo. A sensitivity analysis for mixed criticality: Trading criticality with computational resource. In *23rd ETFA*, 2018.

[25] L. Santinelli, Z. Guo, and L. George. Fault-aware sensitivity analysis for probabilistic real-time systems. In *29th DFT*, 2016.

[26] F. Santy, L. George, P. Thierry, and J. Goossens. Relaxing mixed-criticality scheduling strictness for task sets scheduled with FP. In *24th ECRTS*, 2012.

[27] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *Proceedings of the 24th IEEE Real-Time Systems Symposium*, pages 1–12, 2003.

[28] H. Su and D. Zhu. An elastic mixed-criticality task model and its scheduling algorithm. In *16th DATE*, 2013.

[29] S. Vaidhun, S. Arefin, Z. Guo, H. Xiong, and S. Das. Response time in mixed-critical pervasive systems. In *14th UIC*, 2017.

[30] S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *28th RTSS*, 2007.