# Improving QoS for Global Dual-Criticality Scheduling on Multiprocessors

Lin Huang*, I-Hong Hou*, Sachin S. Sapatnekar† and Jiang Hu*‡
*Department of Electrical and Computer Engineering, Texas A&M University
‡Department of Computer Science and Engineering, Texas A&M University
†Department of Electrical and Computer Engineering, University of Minnesota
liushui0820@tamu.edu, ihou@tamu.edu, sachin@umn.edu and jianghu@tamu.edu

***Abstract*—Mixed-criticality system is a popular model for reducing pessimism in real-time scheduling while providing guarantee for critical tasks in presence of unexpected overrun. However, it is controversial due to some drawbacks. First, a single high-criticality job overrun leads to the pessimistic mode for all high-criticality tasks and consequently resource utilization becomes inefficient. Second, all low-criticality tasks are dropped in high-criticality mode, although they are still needed. These two issues have been addressed in several recent works, which are mostly focused on uniprocessor scheduling. In this work, we attempt to tackle these two issues in multiprocessor scheduling for dual-criticality systems. A deferred switching protocol is introduced so that the chance of switching to high-criticality mode is significantly reduced. Moreover, a service preserving technique is developed such that all low-criticality tasks can continue to execute in high-criticality mode. Further, the two techniques are unified into a single framework. Schedulability of these methods is studied so that the Quality-of-Service is improved with guarantee of satisfying all deadline constraints. The effectiveness of the proposed techniques is confirmed through simulations.**

## I. Introduction

A well-known drawback of conventional realtime scheduling is its overly pessimistic WCET (Worst Case Execution Time) estimation. This drawback is largely overcome in Mixed-Criticality (MC) systems [1], where both tasks and system operation modes are categorized into different criticality levels. In normal low-criticality mode, high-criticality tasks are scheduled using less pessimistic estimate on execution time. In the event that the execution time of a high-criticality job exceeds its estimation in low-criticality mode, i.e., an overrun occurs, the system switches to high-criticality mode, where very pessimistic WCET is applied for all high-criticality tasks. As such, high-criticality tasks still have guarantee for meeting their deadlines. However, such appealing advantage of MC systems comes with expensive price [2].

1) All low-criticality tasks are dropped in high-criticality mode to facilitate the guarantee for high-criticality tasks. Despite their low-criticality, these tasks are still very much needed and completely abandoning them is a heavy loss of Quality of Service (QoS).
2) The overrun of a single high-criticality job can force all the other high-criticality jobs (even without overrun) into high-criticality mode with the very pessimistic WCET, which undermines efficiency of resource utilization.

Both of the limitations recently received research attention. New techniques have been proposed [3]–[8] to continue executing low-criticality tasks in high-criticality mode with graceful degradation. Other approaches [9], [10] define new protocols to prevent all high-criticality tasks from simultaneously entering high-criticality mode. Most of these works are geared toward uniprocessor scheduling. In reality, multiprocessor is a growing trend due to its performance advantage. There are few previous works addressing the aforementioned limitations for multiprocessors. A partitioning-based multiprocessor scheduling approach is introduced in [11]. However, after a load-balancing-based partitioning, its kernel techniques are actually for uniprocessor scheduling. Moreover, it can retain only some but not all low-criticality tasks in high-criticality mode. In [8] and [12], all low-criticality tasks can continue in high-criticality mode for multiprocessors, by improvements on fluid scheduling and global scheduling, respectively. However, none of them addresses the second limitation that all high-criticality tasks simultaneously enter the very pessimistic high-criticality mode.

There are three main approaches to MC multiprocessor scheduling: (1) partitioning-based [11], [13], (2) fluid-based [8], [14], [15] and (3) global scheduling [12], [13], [16]. Each of them has its strength and weakness, and no one is absolutely superior to the others. Partitioning-based scheduling is appealing for its simplicity. On the other hand, it lacks flexibility and tends to cause under-utilization of resources. Fluid-based scheduling can reach the theoretically optimal solution, but may incur frequent job preemptions and context switchings, which incur non-negligible overhead. Global scheduling has both its performance and practicality between partitioning and fluid-based scheduling.

In this work, we focus on how to address the two limitations of dual-criticality system, the basic version of MC system, in global scheduling. First, a service preserving technique is developed to let all low-criticality tasks execute in high-criticality mode with imprecise computing, while all deadline constraints are satisfied. This is to improve QoS for high-criticality mode. Second, a deferred switching scheme is proposed to prevent high-criticality tasks from simultaneously entering high-criticality mode while all deadlines are still guaranteed to be enforced. This is to extend the low-criticality mode, which has better QoS than high-criticality mode. Further, these two techniques are combined into a unified framework such that the two limitations are concurrently mitigated. To the best of our knowledge, this is the first work that simultaneously addresses both of the limitations

for multiprocessors. Our service preserving technique alone outperforms the previous approach of DVD (Dual Virtual Deadline) [12] in term of schedulability. The effectiveness of our work is confirmed through simulations.

The rest of this paper is organized as follows. Related previous works are briefly reviewed in Section II. Section III introduces system model and background knowledge. Our techniques for improving QoS are described in Section IV. Experimental results are shown in Section V. Finally, Section VI provides the conclusion.

## II. Related Previous Work

The MC model was first described in [1]. Early works on MC scheduling are mostly for uniprocessors. One such work is EDF-VD [17], which extends the Earliest Deadline First scheduling with Virtual Deadlines such that time resource is reserved for meeting deadlines during low-criticality mode, high-criticality mode and the transition time at the beginning of high-criticality mode. This method is designed for the classic MC model [1] where all low-criticality tasks are dropped in high-criticality mode. The limitations of the classic MC model have been identified [2] and several recent works are developed for corresponding mitigation. An online adjustment technique [9] is developed to reduce the number of low-criticality tasks that are dropped in high-criticality mode. In [10], multiple intermediate-criticality levels are introduced between low-criticality mode and high-criticality mode such that there is no need to simultaneously switch all high-criticality tasks to high-criticality mode. The work of [18] enables return to low-criticality mode from high-criticality mode. Another approach is to continue low-criticality tasks in high-criticality mode with graceful degradation, such as imprecise computing [7]. All these techniques are developed for uniprocessors.

For multiprocessors, an EDF-VD-based global scheduling method is proposed in [16] using the classic MC model. There are very few previous works on multiprocessor scheduling for addressing the limitations of the classic MC model. The work of [11] is a partitioning-based scheduling that aims to alleviate some limitations of the classic MC model. It first partitions tasks to processors in an effort for load balancing. Then, uniprocessor scheduling is performed for tasks assigned to the same processor. Tasks of the same processor are further divided into multiple groups, each of which contains only one high-criticality task so as to achieve isolation among high-criticality tasks. Processor time is allocated to different groups, where scheduling is performed separately. In high-criticality mode, low-criticality jobs are executed opportunistically without guarantee. As such, the limitation of dropping all low-criticality tasks is not well solved. It has an intermediate mode before entering high-criticality mode. However, all low-criticality tasks are suspended in this mode. In [19], new protocols are proposed for systems to return from high-criticality mode to low-criticality mode. A fluid-based scheduling [8] and a global scheduling [12] are introduced to allow execution of low-criticality tasks in high-criticality mode for multiprocessors. However, both of them follow the conservative model where all high-criticality tasks simultaneously enter high-criticality mode. Fluid-based scheduling tends to cause frequent job preemptions and context switchings, which cost performance overhead. The global scheduling method [12] enforces dual virtual deadline, which is quite conservative and thereby sacrifices schedulability.

## III. Preliminaries

### A. System Model

The baseline model used in this work is very similar as the classic MC model, although only two criticality levels are considered here. We suggest some changes to the model, which will be elaborated. In the baseline dual-criticality system, there are a set of independent sporadic tasks $\mathcal{T} = \{\tau_1, \tau_2, ...\}$ to be performed on $m$ identical processors. This task set is composed by two disjoint subsets of low-criticality tasks $\mathcal{T}_L$ and high-criticality tasks $\mathcal{T}_H$. The system may operate in either low-criticality mode or high-criticality mode. In this paper, we use subscript in $\{L, H\}$ to indicate task criticality and superscript in $\{lo, hi\}$ to differentiate low-criticality and high-criticality modes.

Each task $\tau_i \in \mathcal{T}$ consists of an infinite sequence of jobs $\{J_i^1, J_i^2, ...\}$, and is characterized by $(T_i, \chi_i, Q_i^{lo}, Q_i^{hi})$, where $T_i$ is the minimal time interval between two consecutive jobs of task $\tau_i$, $\chi_i \in \{L, H\}$ indicates its criticality level, and $Q_i^{lo}$ and $Q_i^{hi}$ are estimated job execution time in low-criticality and high-criticality mode, respectively. If job $J_i^j$ is released at time $a_i^j$, its deadline is $a_i^j + T_i$. Therefore, $T_i$ implicitly specifies job deadline, and is also called period for convenience.

A system starts with low-criticality mode, which is also the ordinary operation mode. In order to reduce the pessimism of WCET (Worst Case Execution Time) in conventional real-time scheduling, execution time $Q_i^{lo}$ is chosen to be less conservative for high-criticality tasks. Consequently, there is a low but non-zero probability that the actual execution time exceeds $Q_i^{lo}$. The condition for switching to high-criticality mode and the treatment of high-criticality mode are where several recent works, as well as our work, diverge from the classic MC model. In the classic MC model, as long as any high-criticality job $J_i^j$ has actual execution time exceeding $Q_i^{lo}$, the system switches to high-criticality mode. In high-criticality mode, all high-criticality tasks are scheduled with very pessimistic $Q_i^{hi}$, i.e., $Q_i^{hi} > Q_i^{lo}$, such that there exists a guarantee for meeting deadlines of all high-criticality tasks.

Similar to the imprecise computation model, we assume that each low-criticality job consists of two parts: a mandatory part and an optional part. Since we need to allocate more processing time for high-criticality tasks when the system is running in the high-criticality mode, we assume that low-criticality tasks only require the completions of mandatory parts in the high-criticality mode, while they require the completions of both mandatory and optional parts in the low-criticality mode. As a result, we have $Q_i^{lo} > Q_i^{hi} \geq 0$ for each low-criticality task $\tau_i$. We note most classic MC models assume that all low-criticality tasks are dropped when the system enters the high-criticality mode. Effectively, these models correspond to the special case when $Q_i^{hi} = 0$ for all low-criticality tasks. By allowing $Q_i^{hi}$ to be larger than 0, our model enables a much richer description about the system requirements in the high-criticality mode.

The scheduling is to decide when to execute each job on which processor. We consider global scheduling, where the

jobs of a task can be assigned to different processors. The scheduling is preemptive such that a low-priority job can be preempted by a high-priority job during its execution.

For each task $\tau_i$, its utilizations in low and high-criticality modes are defined as

$$u_i^{lo} = \frac{Q_i^{lo}}{T_i}, \text{ and } u_i^{hi} = \frac{Q_i^{hi}}{T_i},$$

respectively. Then, the total utilizations of all low-criticality tasks are given by

$$U_L^{lo} = \sum_{\tau_i \in \mathcal{T}_L} u_i^{lo} \text{ and } U_L^{hi} = \sum_{\tau_i \in \mathcal{T}_L} u_i^{hi}.$$

Likewise, the total utilizations for high-criticality tasks are defined as

$$U_H^{lo} = \sum_{\tau_j \in \mathcal{T}_H} u_j^{lo} \text{ and } U_H^{hi} = \sum_{\tau_j \in \mathcal{T}_H} u_j^{hi}.$$

Please note in classic MC model, $U_L^{hi} = 0$ since $Q_i^{hi} = 0$ for all low-criticality tasks $\tau_i \in \mathcal{T}_L$.

### B. Global Scheduling and Fluid-Based Scheduling

Our work mainly improves a conventional global scheduling, fpEDF-VD [16], for mitigating the two limitations of classic MC model. DP-Fair scheduling [20], which is an implementation of fluid-based scheduling, is also adopted in a very limited fashion. Both of these techniques are summarized here for the completeness of the description.

*1) Global fpEDF Scheduling on Multiprocessors:* The method of fpEDF (fixed-priority EDF) [21] is a state-of-art global scheduling approach for multiprocessor in traditional realtime systems without mixed-criticality. Fixed-priority means the priority of one job can not be changed during execution. For a task set $\bigcup_{\tau_i \in \mathcal{T}} (T_i, Q_i)$ to be scheduled on $m$ identical processors, fpEDF first chooses a subset $\mathcal{T}_{hp} \subset \mathcal{T}$ of at most $m-1$ tasks, each with utilization greater than $\frac{1}{2}$, and assigns them on $m_{hp}$ processors with the highest priority. The priorities of remaining tasks $\mathcal{T}_{lp} \subset \mathcal{T}$ are lower and scheduled according to EDF (Earliest Deadline First) principle on the other $m_{lp} = m - m_{hp}$ processors. The schedulability condition for fpEDF is given in Lemma 1( [21]).

**Lemma 1.** *Consider a task set $\bigcup_{\tau_i \in \mathcal{T}} (T_i, Q_i)$ to be scheduled on $m$ identical processors. Let $U_{lp}^{total}$ be the total utilization of the tasks in $\mathcal{T}_{lp}$, and $U_{lp}^{max}$ be the maximum utilization of tasks in $\mathcal{T}_{lp}$. If $U_{lp}^{total} \leq m_{lp} - (m_{lp} - 1) \cdot U_{lp}^{max}$ is satisfied, this task set $\mathcal{T}$ is schedulable by fpEDF method.*

*2) Global Scheduling by fpEDF-VD on Multiprocessors:* fpEDF-Virtual Deadline (fpEDF-VD) [16] is an extension of fpEDF to mixed-criticality systems. Virtual deadlines are enforced for high-criticality tasks. Each high-criticality task $\tau_j$ is mapped to $(\hat{T}_j, Q_j^{lo})$ in low-criticality mode, where $\hat{T}_j = x \cdot T_j (0 < x < 1)$ is the virtual deadline that is enforced in both offline schedulability test and online execution. Each low-criticality task $\tau_i$ is mapped to a regular implicit deadline task $(T_i, Q_i^{lo})$ in low criticality mode, and all low-criticality tasks are dropped in high-criticality mode. The schedulability conditions for fpEDF-VD are as follows.

- Task set $(\bigcup_{\tau_i \in \mathcal{T}_L} (T_i, Q_i^{lo})) \bigcup (\bigcup_{\tau_j \in \mathcal{T}_H} (x \cdot T_j, Q_j^{lo}))$ is schedulable on $m$ processors in low-criticality mode according to Lemma 1.
- Task set $\bigcup_{\tau_j \in \mathcal{T}_H} ((1-x) \cdot T_j, Q_j^{hi})$ is schedulable on $m$ processors in high-criticality mode according to Lemma 1.

For a high-criticality task $\tau_j \in \mathcal{T}_H$ in high-criticality mode, its implicit deadline $(1-x) \cdot T_j$ is used in the offline schedulability check. However, only its original deadline $T_j$ needs to be enforced during online execution. By default, virtual deadline of a high-criticality task $\tau_j \in \mathcal{T}_H$ refers to $x \cdot T_j$.

The schedulability condition in high-criticality mode leads to the following important conclusion, which is heavily used in our work.

**Lemma 2.** *( [16]) If a mixed-criticality task set $\mathcal{T}$ is schedulable by fpEDF-VD, each of its high-criticality job $J_j^k$ in high-criticality mode can start from its virtual deadline $\hat{d}_j^k$ and guarantee to finish by actual deadline $d_j^k$ with execution time $Q_j^{hi}$ following fpEDF-VD scheduling.*

*3) DP-Fair Scheduling on Multiprocessors:* DP (Deadline Partition)-Fair [20] is a scheduling method based on proportional fairness for regular (non-MC) multiprocessor systems. In DP-Fair, the density of each task $\tau_i$ is computed as $\delta_i = \frac{Q_i}{T_i}$, where $Q_i$ and $T_i$ are the worst case execution time and period of task $\tau_i$, respectively.

**Theorem 1.** *(Lemma 14 in [14]) A non-MC task set $\mathcal{T}$ is schedulable under DP-Fair iff $\sum_{\tau_i \in \mathcal{T}} \delta_i \leq m$, where $m$ is the number of processors.*

MC-DP-Fair [14] is an extension of DP-Fair for MC systems. It is a fluid-based scheduling and can be speedup-optimal for dual-criticality system scheduling [14], [15]. In this work, we make use of DP-Fair in the proposed service preserving technique, while MC-DP-Fair is implemented for comparison in the experiment.

### C. Imprecise Computing

Imprecise computing, also known as approximate computing, intentionally allows limited computing errors so as to shorten computing time. In high-criticality mode, high-criticality tasks are allocated with increased processor time. In order to maintain schedulability, processor time allocated to low-criticality tasks must decrease. Instead of completely dropping low-criticality tasks in high-criticality mode as in classic MC model, one can execute low-criticality tasks with imprecise computing for graceful service degradation. Recent technology progress on imprecise computing makes such approach increasingly realistic. Imprecise computing can be implemented at circuit, architecture and algorithm levels for datapath or numerical computations. At circuit level approximation, people developed techniques on imprecise arithmetic circuit designs [22], [23] and voltage overscaling [24]. Recently, new methods also make computing accuracy runtime configurable [25]. Architectural approximation techniques include approximate data types [26], dedicated instructions [27] and approximate storage [28]. Algorithm level approximation has been extensively studied as well. For example, relaxing the convergence criterion for iterative algorithms can significantly reduce computing time under allowable computing errors. Overall, the significant progress [23] has made imprecise computing ready for use in mixed-criticality systems.

## IV. QoS Driven Global Scheduling

We introduce two main techniques to mitigate the limitations of classic MC model and thereby improve QoS of dual-criticality systems. Both techniques are mostly built upon the fpEDF-VD (fixed priority Earliest Deadline First - Virtual Deadline) framework [16]. The first is a service preserving technique (Section IV-A) that allows all low-criticality tasks to execute in high-criticality mode with imprecise computing. Compared to the DVD (dual virtual deadline) approach [12], the proposed service preserving technique is less conservative and thereby facilitates improved schedulability. The second is a deferred switching scheme (Section IV-B), which has not been studied for multiprocessors, to the best of our knowledge. It will reduce the chance that a system switches into the very pessimistic high-criticality mode. The two techniques are unified into a single method, which is described in Section IV-C.

### A. Service Preserving Method

The goal of this technique is to continue executing low-criticality tasks in high-criticality mode while all task deadlines are guaranteed to be met. To facilitate this goal, all low-criticality tasks are executed with imprecise computing in high-criticality mode. The imprecise computing costs shorter execution time than precise computing and therefore $Q_i^{lo} > Q_i^{hi} > 0, \forall \tau_i \in \mathcal{T}_L$.

The key issue is how to guarantee schedulability while low-criticality tasks are continued and consume processor time. In this technique, fpEDF-VD scheduling policy is used in low-criticality mode and high-criticality mode, and DP-Fair scheduling policy is used during the transition. In order to ensure schedulability for low-criticality mode, task set

$$(\bigcup_{\tau_i \in \mathcal{T}_L} (T_i, Q_i^{lo})) \bigcup (\bigcup_{\tau_j \in \mathcal{T}_H} (x \cdot T_j, Q_j^{lo}))$$

must be schedulable on $m$ processors according to Lemma 1. Please note by scaling $T_i$ by $x \in (0, 1)$, virtual deadline $x \cdot T_i$ is applied for all high-criticality tasks. We define the high-criticality mode condition as that task set

$$(\bigcup_{\tau_i \in \mathcal{T}_L} (T_i - P, Q_i^{hi})) \bigcup (\bigcup_{\tau_j \in \mathcal{T}_H} ((1-x) \cdot T_j, Q_j^{hi})) \quad (1)$$

must be schedulable on $m$ processors according to Lemma 1, where $P$ is a service preserving interval we introduce and will be elaborated later.

The transition from low-criticality to high-criticality mode is subtle and deserves a lot of attention [7]. The treatment of high-criticality tasks is the same as fpEDF-VD [16]. Consider a high-criticality job $J_j^k$ that is active at moment $t^*$ of mode switching. Its virtual deadline satisfies $\hat{d}_j^k = a_j^k + x \cdot T_j \geq t^*$, otherwise this job would have finished. Right after time $t^*$, the system enters high-criticality mode and the actual deadline $d_j^k = a_j^k + T_j$ is enforced. According to Lemma 2, the extra time budget $(1-x) \cdot T_j$ is sufficient for $J_j^k$ to finish with execution time $Q_j^{hi}$. Therefore, high-criticality tasks are guaranteed to satisfy their deadlines.

The challenging part is how to handle low criticality tasks during the transition, where they can no longer be dropped as in the classic MC model. The non-zero $Q_i^{hi}$ for low-criticality tasks makes the schedulability guarantee quite difficult. In [12], two techniques were proposed. The first technique may result in a one-time dropping of low-criticality jobs during transition. This is against the original intention of continuing all low-criticality tasks. The other technique in [12] is dual virtual deadline (DVD) for avoiding such job loss. Unlike the original fpEDF-VD, where virtual deadline is applied only for high-criticality tasks, the DVD approach enforces virtual deadline for low-criticality tasks as well. Virtual deadline is effective for providing guarantee on meeting deadlines. However, it is basically a conservative resource reservation approach that makes schedulability condition more strict and hence causes under-utilization of resources. Applying virtual deadlines for both low-criticality tasks and high-criticality tasks would exacerbate the inefficiency and is an expensive price paid for avoiding one-time loss of low-criticality jobs.

We suggest a *service preserving interval* $[t^*, t^* + P]$, when only the active (carry-over) low-criticality jobs are executed by DP-Fair scheduling, all active high-criticality jobs are suspended and no newly arrival jobs are started. This is to facilitate that all active low-criticality jobs can be finished with imprecise computing during the transition. Meanwhile, the interval $P$ is designed in a way that the schedulability of all the other jobs are still maintained. A critical basis for the service preserving interval is that execution time $Q_j^{hi}$ is accommodated after virtual deadline $\hat{d}_j^k$ for a high-criticality job $J_j^k$ in high-criticality mode according to Lemma 2 by fpEDF-VD [16]. The service preserving interval length is defined as

$$P = \min_{\forall \tau_j \in \mathcal{T}_H} Q_j^{lo} \quad (2)$$

Next, we will discuss schedulability of active jobs and those involving the service preserving interval.

**Lemma 3.** *By following fpEDF-VD, all high-criticality jobs can guarantee to meet their deadlines in high-criticality mode even if they are not executed in* $[t^*, t^* + P]$.

*Proof:* The high-criticality jobs involving the service preserving interval $[t^*, t^* + P]$ can be categorized into three cases, all of which will be discussed as follows.

**Case 1:** Overrun jobs. These are the high-criticality jobs that have executed $Q^{lo}$ time but have not finished (see Figure 1). At the end of the $Q^{lo}$ time, the system enters high-criticality mode when the moment is $t^*$. According to the schedulability conditions of fpEDF-VD, the virtual deadline $\hat{d}_j^o$ of an overrun job $J_j^o$ satisfies $\hat{d}_j^o \geq t^*$. The method of fpEDF-VD (Lemma 2) also indicates that all high-criticality jobs can execute $Q^{hi}$ after their virtual deadlines and finish before their actual deadlines in high-criticality mode. Since time $Q_j^{lo}$ has already been executed for job $J_j^o$ at $t^*$, deferring the rest of its execution by $Q_j^{lo}$ maintains the schedulability. In other words, the rest of the overrun job can start from $\hat{d}_j^o + Q_j^{lo} = \hat{d}_j^o + Q_j^{lo} + t^* - t^* = t^* + P_j$, where $P_j = \hat{d}_j^o + Q_j^{lo} - t^*$. Since $\hat{d}_j^o \geq t^*$, $P_j \geq Q_j^{lo}$. Therefore, postponing the execution of the rest of $J_j^o$ by $Q_j^{lo}$ will maintain the schedulability of overrun jobs.

**Case 2:** Active high-criticality jobs without overrun (see Figure 2). A high-criticality job $J_j^k$ has been executed $q_j^k < Q_j^{lo}$ by $t^*$. Then, its rest portion can start from $\hat{d}_j^k + q_j^k$ with guarantee of meeting its deadline according to fpEDF-VD. Like Case 1, $\hat{d}_j^k + q_j^k = t^* + P_j$, where $P_j = \hat{d}_j^k + q_j^k - t^*$. By schedulability

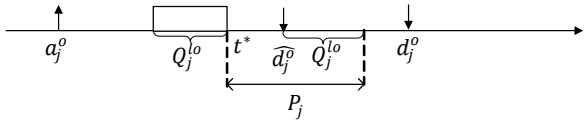


Fig. 1. Case 1: service preserving interval for an overrun job.

condition in low-criticality mode, $q_j^k + \hat{d}_j^k - t^* \geq Q_j^{lo}$, then $P_j \geq Q_j^{lo}$. Hence, such a job can be suspended in $[t^*, t^* + Q_j^{lo}]$ without affecting its schedulability.

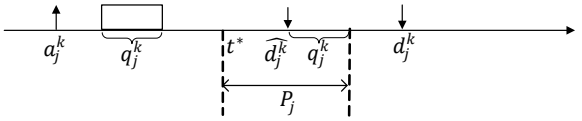


Fig. 2. Case 2: service preserving interval for an active high-criticality job without overrun.

**Case 3:** High-criticality jobs arriving during the service preserving interval (see Figure 3). The arrival time $a_j^k$ of such a job $J_j^k$ satisfies

$$t^* \leq a_j^k \leq t^* + P. \tag{3}$$

The schedulability conditions in fpEDF-VD require that

$$a_j^k + Q_j^{lo} \leq \hat{d}_j^k. \tag{4}$$

Combing inequality (3) and (4), we have

$$Q_j^{lo} \leq \hat{d}_j^k - a_j^k \leq \hat{d}_j^k - t^* = P_j$$

Since $J_j^k$ can guarantee finish before its deadline even if it starts from $\hat{d}_j^k$ according to fpEDF-VD, its start time can be deferred by $P_j$, which is lower bounded by $Q_j^{lo}$.

Overall, all high-criticality jobs involving the service preserving interval can be deferred by $Q^{lo}$ without affecting their schedulability. Hence, deferring by $P = \min_{\forall \tau_j \in \mathcal{T}_H} Q_j^{lo}$ for all these jobs can still guarantee to meet their deadlines.

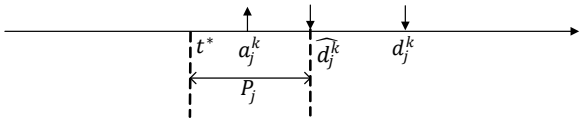


Fig. 3. Case 3: service preserving interval for an immediate newly coming high-criticality job.

■

Next, we describe schedulability conditions for active low-criticality jobs during the service preserving interval $[t^*, t^* + P]$. At $t^*$, if a low-criticality job $J_i^k$ has already been executed for at least $Q_i^{hi}$, it is terminated with imprecise computing result. An active (carry-over) low-criticality job $J_i^k$ means that it has been executed for $q_i^k < Q_i^{hi}$ by $t^*$. The active low-criticality jobs are scheduled by the fluid-based DP-Fair method (see Section III-B3) in the service preserving interval, while fpEDF-VD is employed all the other time. Although fluid scheduling tends to entail frequent job preemptions, it is utilized only within the limited service preserving interval. The schedulability for DP-fair method is largely decided by the job density.

**Lemma 4.** *The density $\delta_i^k$ of an active low-criticality job $J_i^k$ in $[t^*, t^* + P]$ is no greater than* $\max(\frac{Q_i^{hi}}{P}, \frac{Q_i^{hi}}{Q_i^{lo}})$.

*Proof:* This bound is derived from two cases. In one case, deadline $d_i^k \geq t^* + P$ as shown in Figure 4. In the worst case for the service preserving interval, entire job $Q_i^{hi}$ is executed by $t^* + P$, the density of this case is upper bounded as

$$\delta_i^k|_{d_i^k \geq t^* + P} \leq \frac{Q_i^{hi}}{P}. \tag{5}$$

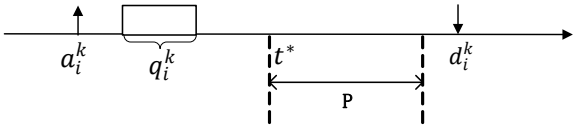


Fig. 4. Active low-criticality job with deadline after $t^* + P$.

In the other case, $d_i^k < t^* + P$ as shown in Figure 5. If $q_i^k$ has been executed by $t^*$, the density is estimated by

$$\delta_i^k|_{d_i^k < t^* + P} = \frac{Q_i^{hi} - q_i^k}{d_i^k - t^*} \tag{6}$$

By the schedulability condition in low-criticality mode, $Q_i^{lo} - q_i^k \leq d_i^k - t^*$. Therefore,

$$\delta_i^k|_{d_i^k < t^* + P} = \frac{Q_i^{hi} - q_i^k}{d_i^k - t^*} \leq \frac{Q_i^{hi} - q_i^k}{Q_i^{lo} - q_i^k} \tag{7}$$

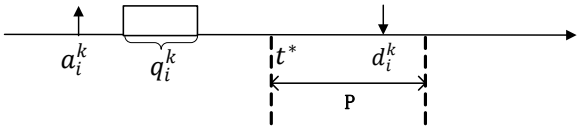


Fig. 5. Active low criticality job with deadline before $t^* + P$.

Consider a function

$$f(x) = \frac{Q_i^{hi} - x}{Q_i^{lo} - x}, \quad 0 \leq x < Q_i^{hi} < Q_i^{lo}. \tag{8}$$

Since derivative $f'(x) = \frac{Q_i^{hi} - Q_i^{lo}}{(Q_i^{lo} - x)^2} < 0$, $f(x)$ is a monotone decreasing function and its maximum is at $x = 0$. Hence,

$$\delta_i^k|_{d_i^k < t^* + P} \leq \frac{Q_i^{hi} - q_i^k}{Q_i^{lo} - q_i^k} \leq \frac{Q_i^{hi}}{Q_i^{lo}} \tag{9}$$

By combining the two cases, we have

$$\delta_i^k \leq \max(\frac{Q_i^{hi}}{P}, \frac{Q_i^{hi}}{Q_i^{lo}}) \tag{10}$$

■

In the worst case, every low-criticality task has an active job at $t^*$. According to Theorem 1 and Lemma 4, a sufficient condition for DP-Fair method to successfully schedule all the active jobs on $m$ processors in $[t^*, t^* + P]$ is

$$\sum_{\forall \tau_i \in \mathcal{T}_L} \max(\frac{Q_i^{hi}}{P}, \frac{Q_i^{hi}}{Q_i^{lo}}) \leq m \tag{11}$$

Last, we discuss new low-criticality jobs that arrive in $[t^*, t^* + P]$. Our method does not allow such jobs to be executed until $t^* + P$. In other words, their execution is deferred by at most $P$. We specify that task set

$$(\bigcup_{\tau_i \in \mathcal{T}_L} (T_i - P, Q_i^{hi})) \bigcup (\bigcup_{\tau_j \in \mathcal{T}_H} ((1-x) \cdot T_j, Q_j^{hi}))$$

must be schedulable according to Lemma 1 in high-criticality mode. More specifically, a low-criticality task $\tau_i$ is scheduled with period (implicit deadline) $T_i - P$. Thus, with deferral of $P$, a low-criticality job arriving in $[t^*, t^* + P]$ is still schedulable.

Putting everything together, the service preserving policy is stated as follows.

**Service preserving policy**: *From the moment $t^*$ switching to high-criticality mode to $t^* + P$, where $P = \min_{\forall \tau_j \in \mathcal{T}_H} Q_j^{lo}$, only active low-criticality jobs are executed with DP-Fair scheduling and all the other jobs can not be executed.*

From Lemmas 3 and 4, we can reach the following conclusion.

**Theorem 2.** *When applying the service preserving policy with fpEDF-VD scheduling, a task set $\mathcal{T}$ is schedulable on $m$ identical processors if $\mathcal{T}$ satisfies the following schedulability conditions.*

- *task set*
  $$(\bigcup_{\tau_i \in \mathcal{T}_L} (T_i, Q_i^{lo})) \bigcup (\bigcup_{\tau_j \in \mathcal{T}_H} (x \cdot T_j, Q_j^{lo}))$$
  *must be schedulable on $m$ processors according to Lemma 1.*
- $\sum_{\forall \tau_i \in \mathcal{T}_L} \max(\frac{Q_i^{hi}}{P}, \frac{Q_i^{hi}}{Q_i^{lo}}) \leq m$ *during* $[t^*, t^* + P]$, *where* $P = \min_{\forall \tau_j \in \mathcal{T}_H} Q_j^{lo}$.
- *task set*
  $$(\bigcup_{\tau_i \in \mathcal{T}_L} (T_i - P, Q_i^{hi})) \bigcup (\bigcup_{\tau_j \in \mathcal{T}_H} ((1-x) \cdot T_j, Q_j^{hi}))$$
  *must be schedulable on $m$ processors according to Lemma 1.*

### B. Deferred Switching Scheme

Although high-criticality mode guarantees that high-criticality tasks complete before deadlines in the worst case, it entails expensive price that the execution time estimation of all high-criticality tasks becomes overly pessimistic even if many of them do not have overrun. Moreover, low-criticality tasks would run in imprecise mode or are even dropped. The threshold of switching to high-criticality mode in the conventional protocol is quite low. That is, any single high-criticality job overrun triggers the mode switching. Generally, there are two approaches that can address this issue. One is to allow a system to switch back to low-criticality mode like bailout mode protocol [18], [19]. The other is to defer the switching into high-criticality mode like the work of [10]. We take the latter approach for multiprocessors while the work of [10] is for uniprocessor scheduling.

Our approach is built upon fpEDF-VD [16] with the observation that the conservativeness of fpEDF-VD allows room for such deferral. In the proposed scheme, a single high-criticality job overrun does not warrant immediate switching to high-criticality mode. Instead, the system enters a vigilant mode, which is almost identical as low-criticality mode except that the overrun job is monitored to decide if the system can recover back to low-criticality mode or must switch to high-criticality mode. The online monitoring and decision can still guarantee satisfaction of all deadline constraint even though $Q_i^{hi}$ is applied with the overrun job, while all low-criticality tasks are retained and all the other high-criticality tasks are scheduled with less pessimistic $Q_j^{lo}$.

*Vigilant mode* is defined by the following characteristics.

- At low-criticality mode, if any high-criticality job $J_j^o \in \tau_j \in \mathcal{T}_H$ does not finish after being executed $Q_j^{lo}$, i.e., has overrun, then the system enters vigilant mode at this moment $t'$.
- Every low-criticality task $\tau_i \in \mathcal{T}_L$ continues to execute with precise computing with estimated execution time $Q_i^{lo}$.
- Each non-overrun high-criticality job $J_j^k$ continues to execute with estimated execution time $Q_j^{lo}$.
- Each overrun high-criticality job $J_j^o$ is scheduled with estimated execution time $Q_j^{hi}$ and priority lower than any low-criticality job and non-overrun high-criticality jobs.
- Each overrun high-criticality job $J_j^o$ is assigned a series of checkpoints $c_h(J_j^o), h = 0, 1, 2, ...$, when some conditions are checked with constant time to decide if $J_j^o$ allows to return to low-criticality mode, demands high-criticality mode or needs to stay at vigilant mode.
- The system switches to high-criticality mode if any overrun high-criticality job demands high-criticality mode.
- When no overrun high-criticality job demands high-criticality mode, the system stays at vigilant mode as long as any overrun high-criticality job needs so.
- The system returns to low-criticality mode when all overrun high-criticality jobs allow so.

The initial checkpoint for an overrun job $J_j^o$ is defined as

$$c_1(J_j^o) = \hat{d}_j^o + Q_j^{lo} \tag{12}$$

where $\hat{d}_j^o$ is the virtual deadline by fpEDF-VD [16] for job $J_j^o$. An example of checkpoint is shown in Figure 6. Later on, it is likely that the checkpoint is updated to $c_2(J_j^o) > c_1(J_j^o)$ and $c_3(J_j^o) > c_2(J_j^o)$, and so on. For the convenience of representation, we specify $c_0(J_j^o) = \hat{d}_j^o$ and $q_0(J_j^o) = Q_j^{lo}$. When time reaches the checkpoint $c_h(J_j^o), h = 1, 2, ...$, the amount of execution of $J_j^o$ from the previous checkpoint

$c_{h-1}(J_j^o)$ to $c_h(J_j^o)$ is examined with constant time and the result determines three different outcomes.

1) If job $J_j^o$ is completed by $c_h(J_j^o)$, it allows the system to return to low-criticality mode.
2) Zero amount has been done, then high-criticality mode is demanded.
3) If $q_h(J_j^o)$ processor time has been spent on executing $J_j^o$ yet it is not completed, the job needs to stay at vigilant mode with the next checkpoint as $c_{h+1}(J_j^o) = c_h(J_j^o) + q_h(J_j^o), h = 0, 1, 2, ....$

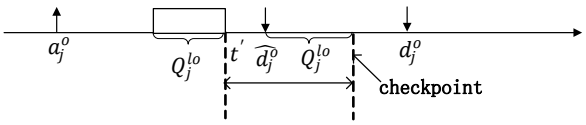


Fig. 6. Illustration of checkpoint.

By introducing the vigilant mode, our approach can defer the switching to high-criticality mode. As the system can return to low-criticality mode from vigilant mode, the overall probability of entering high-criticality mode is also reduced. As such, low-criticality tasks are executed with improved quality and less conservative execution time estimation is applied for high-criticality jobs without overrun. Next, we will show that all jobs are guaranteed to satisfy their deadline constraints under this scheme. The key idea is to let overrun jobs have low priority in the vigilant mode so that the schedulability of the other jobs are not affected. At the same time, an overrun job reclaims time slack at runtime in an opportunistic manner. The online checking has constant complexity.

**Lemma 5.** *The deferred switching scheme guarantees that all jobs complete before their deadlines if they satisfy the schedulability conditions of fpEDF-VD and $Q_i^{hi} = 0$ for all low-criticality tasks $\tau_i \in \mathcal{T}_L$ .*

*Proof:* We prove this for three kinds of jobs - all low-criticality jobs, non-overrun high-criticality jobs and overrun high-criticality jobs, in all three modes - low-criticality mode, vigilant mode and high-criticality mode.

The low-criticality mode in this scheme is handled in the same way as fpEDF-VD for all kinds of jobs. Hence, all tasks can guarantee to satisfy deadlines if they meet the fpEDF-VD schedulability conditions.

The high-criticality mode in this scheme is also identical to that for fpEDF-VD. As such, all non-overrun high-criticality jobs can finish before their deadlines. In the classic MC system model, all low-criticality tasks are dropped in high-criticality mode. In Section IV-C, we will show how to unify the deferred switching scheme with the service preserving technique such that low-criticality tasks can continue to execute with imprecise computing.

In the vigilant mode, low-criticality tasks and non-overrun high-criticality jobs are treated in the same way as in low-criticality mode, except the presence of overrun high-criticality jobs. However, overrun high-criticality jobs have lower priority. Thus low-criticality tasks and non-overrun high-criticality jobs are not affected by those overrun high-criticality jobs, and their deadline can still be met with guarantee.

Last, we discuss overrun high-criticality jobs in the vigilant mode and high-criticality mode. The worst case execution time of an overrun job $J_j^o$ is $Q_j^{hi}$, of which $Q_j^{lo}$ has already been executed. According to fpEDF-VD [16], entire execution time $Q_j^{hi}$ can be accommodated from its virtual deadline $\hat{d}_j^k$ to its actual deadline $d_j^k$ for any high-criticality job $J_j^k$ in high-criticality mode. As the overrun job has already been executed $Q_j^{lo}$, time interval $[\hat{d}_j^o + Q_j^{lo}, d_j^o]$ is sufficient to accommodate the rest execution time $Q_j^{hi} - Q_j^{lo}$. Please note the first checkpoint $c_1(J_j^o) = \hat{d}_j^o + Q_j^{lo}$. Even if nothing of $J_j^o$ has been executed in $[t', c_1(J_j^o)]$ the remaining $Q_j^{hi} - Q_j^{lo}$ part of the overrun job can guarantee to meet its deadline if the system switches to high-criticality mode at $c_1(J_j^o)$.

When a new checkpoint is added with augmenting $q_h(J_j^o)$ to the previous checkpoint, the condition is that $q_h(J_j^o)$ has been executed from the previous checkpoint. As such, the amount of deferral of high-criticality mode start time is equal to the reduction of remaining execution time of the overrun job. Therefore, switching to high-criticality mode at the new checkpoint still guarantees the satisfaction of deadline constraint for this overrun job. Overall, if an overrun job finishes after switching to high-criticality mode, it can guarantee to be completed before its actual deadline.

Since each checkpoint update is extended by $q_h(J_j^o)$ that has been executed, the total extension after the virtual deadline cannot be greater than $Q_j^{hi}$. In other words, the maximum possible checkpoint is $\hat{d}_j^o + Q_j^{hi}$. If the system has not switched to high-criticality mode at $\hat{d}_j^o + Q_j^{hi}$, job $J_j^o$ must have finished by $\hat{d}_j^o + Q_j^{hi}$. Since fpEDF-VD entails that $\hat{d}_j^o + Q_j^{hi} \leq d_j^o$, the job must have finished before its deadline during vigilant mode.

■

Although we propose to perform low-criticality jobs with precise computing in vigilant mode, sometimes, it is beneficial to execute them with imprecise computing. Since the execution time of imprecise computing is usually shorter than precise computing, more processor time can be saved and the chance of switching to high-criticality mode is decreased. The choice between precise computing and imprecise computing may depend on how low-criticality tasks are treated in high-criticality mode. If they are dropped, then even imprecise computing in vigilant mode is a QoS improvement. If low-criticality tasks are continued with imprecise computing in high-criticality mode, then it makes more sense to run them with precise computing in vigilant mode to have the advantage of deferring the mode switching to high-criticality mode.

### C. Unified Deferred Switching and Service Preserving

When unifying the deferred switching and service preserving methods, a couple of changes need to made to the deferred switching part. Both of the techniques make use of Lemma 2 that fpEDF-VD accommodates $Q_j^{hi}$ after virtual deadline $\hat{d}_j^o$ for a high-criticality job $J_j^o$ in high-criticality mode. When

both the techniques are applied at the same time, the same property cannot be utilized twice. This is the key reason for changing the deferred switching method here. The first change is that each overrun job can only have one checkpoint as opposed to possibly multiple checkpoints described in Section IV-B. Since there is only one checkpoint, an overrun job not finished by the checkpoint immediately demands high-criticality mode. The second change is that the checkpoint is defined as $c(J_j^o) = \hat{d}_j^o$ in contrast to $c_1(J_j^o) = \hat{d}_j^o + Q_j^o$ in Section IV-B. In the unification, the service preserving part is the same as introduced in Section IV-A.

Since the change in the unified method is restricted to the deferred switching part, which is an online technique, the offline schedulability conditions of the unified method is the same as that for the service preserving method, which is stated in Theorem 2.

**Lemma 6.** *If the schedulability conditions in Theorem 2 are satisfied, the modified deferred switching in the unified method still maintains the schedulability.*

*Proof:* In the unified method, low-criticality mode and the high-criticality mode after the service preserving interval are identical to each stand-alone method. The vigilant mode is the same as low-criticality mode except the handling of overrun jobs. Overrun jobs are executed opportunistically in vigilant mode with their deadline guarantee provided by switching to high-criticality mode in time. The vigilant mode in the unified method is never longer than that in the stand alone deferred switching scheme. As such, the deadline guarantee of overrun jobs still relies on the scheduling in high-criticality mode. In the unified method, $t^*$ is at a checkpoint $c(J_j^o) = \hat{d}_j^o$, which is covered in Case 1 in the proof of Lemma 3. The other kinds of jobs and cases discussed in the proof of Lemma 3 still hold in the unified method. The schedulability condition for the service preserving interval in the unified method is unchanged. Therefore, the unified method can guarantee that all tasks meet their deadlines if the schedulability conditions in Theorem 2 are satisfied.

■

## V. Experimental Results

In the experiments, we evaluate and compare the following methods through software simulations.

- **Partitioning**: Partitioning-based scheduling method [13], where all low-criticality tasks are dropped in high-criticality mode.
- **fpEDF-VD**: fpEDF-VD scheduling [16], where all low-criticality tasks are dropped in high-criticality mode.
- **MC-DP-Fair**: The fluid-based MC-DP-Fair method [14], where all low-criticality tasks are dropped in high-criticality mode.
- **Dual-VD**: This is an extension to the fpEDF-VD scheduling such that virtual deadline is applied for both low-criticality and high-criticality tasks [12]. In this method, low-criticality tasks continue to execute with imprecise computing in high-criticality mode.
- **Deferred-Switching**: Our deferred switching method based on fpEDF-VD scheduling (Section IV-B). Low-criticality tasks are dropped in high-criticality mode.
- **Deferred-Switching-Apprx**: This method is the same as *Deferred-Switching* except that all low-critical tasks execute with imprecise computing in vigilant mode.
- **Service-Preserving**: Our service preserving method based on fpEDF-VD scheduling (Section IV-A). In this method, low-criticality tasks continue to execute with imprecise computing in high-criticality mode.
- **Unified**: The unified deferred switching and service preserving scheme based on fpEDF-VD scheduling (Section IV-C). In this method, low-criticality tasks continue to execute with imprecise computing in high-criticality mode.

### A. Testcase Generation

The testcases in the experiments are randomly generated. For each testcase, the probability of a task being low-criticality (high-criticality) is 0.5. For each low-criticality task, we set its low-criticality mode utilization randomly in $[0.1, 0.9]$ under uniform distribution. Likewise, the high-criticality mode utilization of each high-criticality task is also randomly generated in $[0.1, 0.9]$, under uniform distribution. The period $T_i$ of each task $\tau_i$ is randomly chosen in $[100, 500]$ according to uniform distribution. For each low-criticality task $\tau_i$, its execution times are set as $Q_i^{lo} = T_i \cdot u_i^{lo}$ and $Q_i^{hi} = k_L \cdot Q_i^{lo}$, where the scaling factor $k_L$ is randomly chosen in $[0.1, 0.9]$ following uniform distribution. For each high-criticality task $\tau_j$, we set its high-criticality mode execution time $Q_j^{hi} = T_j \cdot u_j^{hi}$. Its low-criticality mode execution time is obtained according to $Q_j^{hi} = k_H \cdot Q_j^{lo}$, where $1.1 \leq k_H \leq 7.5$.

### B. Evaluation of Service Preserving

We evaluated the acceptance ratio of our service preserving technique and compared with the dual-VD method. Please note our unified scheme should have the same acceptance ratio as that of service preserving method, as both apply the same offline schedulability test. The deferred switching part is an online technique that does not affect the acceptance ratio. The other methods are not compared as they all drop low-criticality tasks in high-criticality mode. Therefore, they are not comparable with our method, which retains low-criticality tasks in high-criticality mode. The results for 4 processors and 8 processors are shown in Figure 7 and Figure 8, respectively. At each utilization, 10,000 testcases are randomly generated for evaluation. The difference between the two methods mainly exhibit around utilization 0.6, where our service preserving can improve as much as 50%.

### C. Evaluation of Deferred Switching and Unified Method

In this part of experiment, we compare our deferred switching and the unified method with several previous works. The comparison is performed on two metrics. One is the number of completed low-criticality jobs before switching to high-criticality mode and the other is the mode switching time. Compared to methods that drop low-criticality tasks in high-criticality mode, the deferred switching or more completed low-criticality jobs improves overall QoS. For each utilization, 1000 *schedulable* testcases are generated and each case is

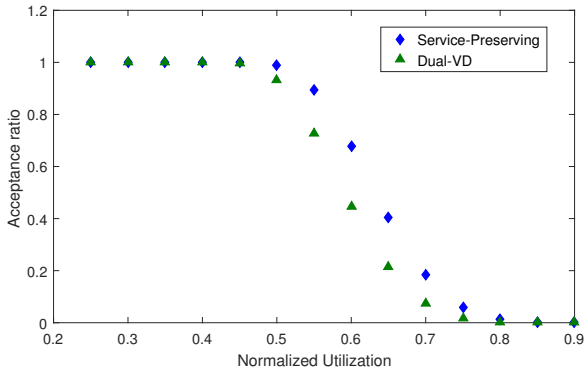


Fig. 7. Acceptance ratio vs normalized utilization of 4 processors.

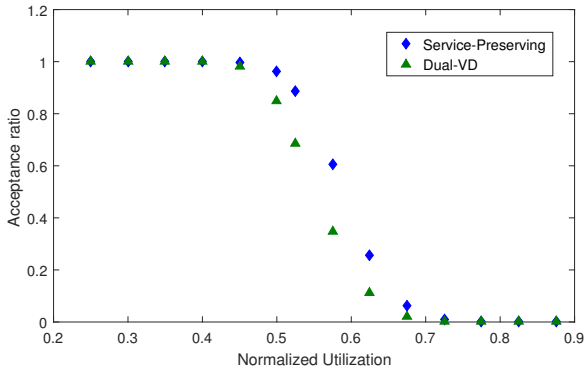


Fig. 8. Acceptance ratio vs normalized utilization of 8 processors.

simulated 10 times to account for the random job execution time.

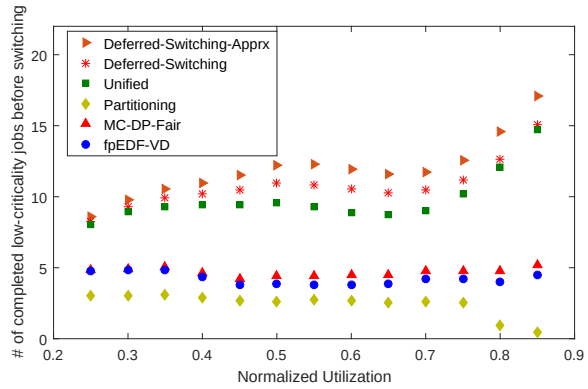


Fig. 9. Number completed low-criticality jobs before mode switching vs normalized utilization of 4 processors, with overrun rate 0.2.

Figures 9 and 10 show the number of completed low-criticality jobs before mode switching for 4 processors with overrun rate of 0.2 and 0.5, respectively. The *overrun rate* is the probability that a high-criticality job has overrun. Both our deferred switching and unified methods complete significantly more low-criticality jobs than the previous works. Since the unified method needs to continue low-criticality jobs in high-criticality mode, it is more conservative than the stand alone deferred switching method. When the utilization is high, task

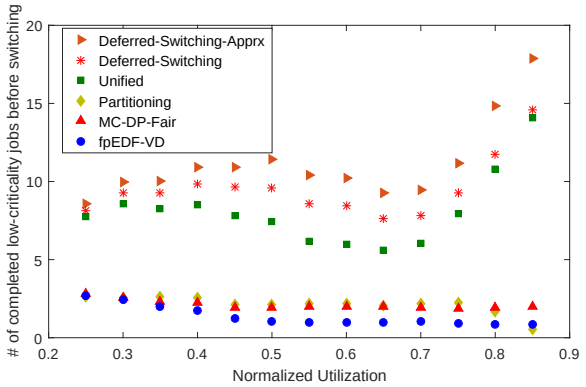


Fig. 10. Number of completed low-criticality jobs before mode switching vs normalized utilization of 4 processors, overrun rate 0.5.

period $T$ is relatively short. Consequently, the number of low-criticality jobs active in vigilant mode becomes large and the number of completed low-criticality jobs also increase accordingly. Therefore, the advantage of our proposed method is much more significant in heavily loaded cases. The results for 8-processor cases are shown in Figure 11 and 12, and similar trends as 4-processor cases can be observed, since the schedulability condition on 8-processor is relatively more conservative and it does not produce so much data as 4-processor. These figures also indicate that performing low-criticality tasks with imprecise computing in vigilant mode allows more low-criticality jobs to be completed than precise computing. Hence, our framework provides options with different tradeoff between computing precision and number of completed jobs.

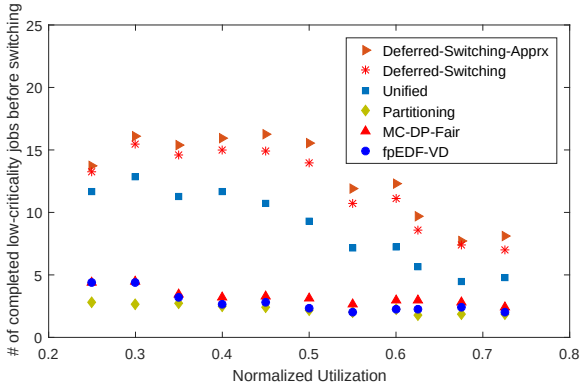


Fig. 11. Number of completed low-criticality jobs before mode switching vs normalized utilization of 8 processors with overrun rate 0.2.

In Figures 13 and 14, we compare the switching time to high-criticality mode among different methods. The overrun rate is 0.5 for the cases in both of the figures. Again, our methods can delay the switching for a long time compared to the previous works. This delay reduces the QoS loss in high-criticality mode. In choosing the value for $x$, which is the scaling factor for virtual deadline, we sweep from a small value toward large values, and stop whenever schedulability conditions are satisfied. When utilization is low, even small $x$ values are sufficient for schedulability. When utilization is high, $x$ must be chosen as greater or more balanced value.

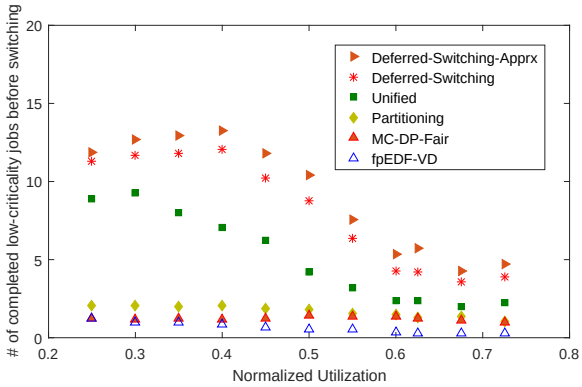


Fig. 12. Number of completed low-criticality jobs before mode switching vs normalized utilization of 8 processors, with overrun rate 0.5.

Since the checkpoints in deferred switching and the unified method largely depend on virtual deadlines, a relatively late virtual deadline contributes to greater deferral by our methods. By imprecise computing for low-criticality jobs in vigilant mode, the amount of deferral is increased according to the results shown in the figures.

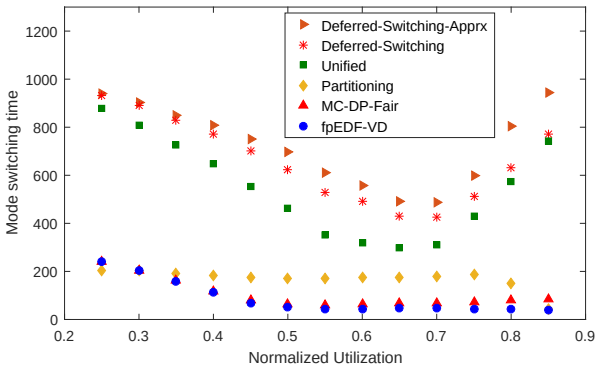


Fig. 13. Mode switching time vs normalized utilization of 4 processors.

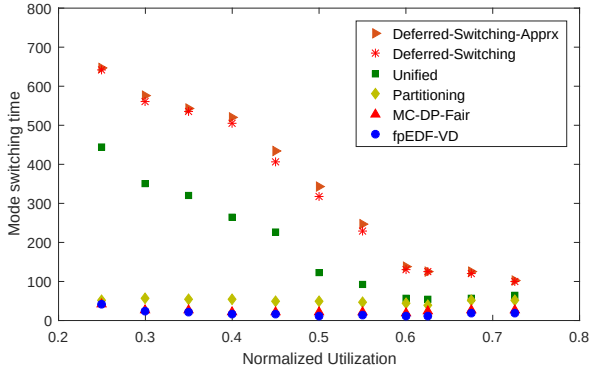


Fig. 14. Mode switching time vs normalized utilization of 8 processors.

## VI. Conclusions

The classic mixed-criticality model adopts a very pessimistic high-criticality mode where all low-criticality tasks are abandoned. Moreover, a single high-criticality job overrun causes immediate switching to high-criticality mode where all high-criticality tasks are scheduled with overly pessimistic execution time estimate. Most of previous works address these problems for uniprocessors while we focus on global scheduling on multiprocessors. We developed a service preserving technique in the fpEDF-VD framework that allows all low-criticality tasks to execute with imprecise computing in high-criticality mode. It is less conservative and remarkably improves acceptance ratio compared to the previous work of dual virtual deadline. Moreover, a vigilant mode and online checkpoint method is proposed to deferred the switching to high-criticality mode. These two techniques are further unified into a single method. These techniques significantly improve Quality of Service (QoS) for low-criticality tasks in mixed-criticality systems.

## Acknowledgement

This work was supported in part by the National Science Foundation (CCF-1525925 and CCF-1525749), and Office of Naval Research (N00014-18-1-2048).

## References

[1] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degree of execution time assurance," in *Proceedings of Real-Time Systems Symposium (RTSS)*, pp. 239–243, IEEE, 2007.

[2] A. Burns and R. Davis, "A survey of research into mixed criticality systems," *ACM Computing Surveys (CSUR)*, vol. 50, no. 6, p. 82, 2018.

[3] H. Su and D. Zhu, "An elastic mixed-criticality task model and its scheduling algorithm," in *Proceedings of Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pp. 147–152, IEEE, 2013.

[4] H. Xu and A. Burns, "Semi-partitioned model for dual-core mixed criticality system," in *Proceedings of the International Conference on Real Time and Networks Systems (RTNS)*, pp. 257–266, ACM, 2015.

[5] A. Burns and S. Baruah, "Towards a more practical model for mixed criticality systems," in *Workshop on Mixed-Criticality Systems*, 2013.

[6] S. Baruah, A. Burns, and Z. Guo, "Scheduling mixed-criticality systems to guarantee some service under all non-erroneous behaviors," in *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pp. 131–138, IEEE, 2016.

[7] 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 *Proceedings of Real-Time Systems Symposium (RTSS)*, pp. 35–46, IEEE, 2016.

[8] R. M. Pathan, "Improving the quality-of-service for scheduling mixed-criticality systems on multiprocessors," in *LIPIcs-Leibniz International Proceedings in Informatics*, vol. 76, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[9] J. Lee, H. S. Chwa, L. T. Phan, I. Shin, and I. Lee, "Mc-adapt: Adaptive task dropping in mixed-criticality scheduling," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, no. 5s, p. 163, 2017.

[10] G. Chen, N. Guan, B. Hu, and W. Yi, "EDF-VD scheduling of flexible mixed-criticality system with multiple-shot transitions," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2393–2403, 2018.

[11] J. Ren and L. T. X. Phan, "Mixed-criticality scheduling on multiprocessors using task grouping," in *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pp. 25–34, IEEE, 2015.

[12] L. Huang, I. Hou, S. S. Sapatnekar, and J. Hu, "Graceful degradation of low-criticality tasks in multiprocessor dual-criticality systems," in *Proceedings of the International Conference on Real-Time Networks and Systems (RTNS)*, pp. 159–169, ACM, 2018.

[13] S. Baruah, B. Chattopadhyay, H. Li, and I. Shin, "Mixed-criticality scheduling on multiprocessors," *Real-Time Systems*, vol. 50, no. 1, pp. 142–177, 2014.

[14] 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 Real-Time Systems Symposium (RTSS)*, pp. 41–52, IEEE, 2014.

[15] S. Baruah, A. Eswaran, and Z. Guo, "MC-Fluid: simplified and optimally quantified," in *Proceedings of Real-Time Systems Symposium (RTSS)*, pp. 327–337, IEEE, 2015.

[16] H. Li and S. Baruah, "Global mixed-criticality scheduling on multiprocessors," in *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pp. 166–175, IEEE, 2012.

[17] S. Baruah, V. Bonifaci, G. DAngelo, 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 Euromicro Conference on Real-Time Systems (ECRTS)*, pp. 145–154, IEEE, 2012.

[18] I. Bate, A. Burns, and R. I. Davis, "A bailout protocol for mixed criticality systems," in *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pp. 259–268, IEEE, 2015.

[19] F. Santy, G. Raravi, G. Nelissen, V. Nelis, P. Kumar, J. Goossens, and E. Tovar, "Two protocols to reduce the criticality level of multiprocessor mixed-criticality systems," in *Proceedings of the International conference on Real-Time Networks and Systems (RTNS)*, pp. 183–192, ACM, 2013.

[20] S. Funk, G. Levin, C. Sadowski, I. Pye, and S. Brandt, "DP-Fair: a unifying theory for optimal hard real-time multiprocessor scheduling," *Real-Time Systems*, vol. 47, no. 5, pp. 389–429, 2011.

[21] S. Baruah, "Optimal utilization bounds for the fixed-priority scheduling of periodic task systems on identical multiprocessors," *IEEE Transactions on Computers*, vol. 53, no. 6, pp. 781–784, 2004.

[22] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 1, pp. 124–137, 2013.

[23] J. Han and M. Orshansky, "Approximate computing: an emerging paradigm for energy-efficient design," in *Proceedings of IEEE European Test Symposium*, pp. 1–6, IEEE, 2013.

[24] D. Mohapatra, V. K. Chippa, A. Raghunathan, and K. Roy, "Design of voltage-scalable meta-functions for approximate computing," in *Proceedings of Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pp. 1–6, IEEE, 2011.

[25] R. Ye, T. Wang, F. Yuan, R. Kumar, and Q. Xu, "On reconfiguration-oriented approximate adder design and its application," in *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, pp. 48–54, IEEE, 2013.

[26] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "EnerJ: approximate data types for safe and general low-power computation," in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 164–174, 2011.

[27] S. Venkataramani, V. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Quality programmable vector processors for approximate computing," in *Proceedings of IEEE/ACM International Symposium on Microarchitecture*, pp. 1–12, 2013.

[28] A. Sampson, J. Nelson, K. Strauss, and L. Ceze, "Approximate storage in solid-state memories," in *Proceedings of IEEE/ACM International Symposium on Microarchitecture*, pp. 25–36, 2013.