Improving the Quality-of-Service for Scheduling Mixed-Criticality Systems on Multiprocessors*

Risat Mahmud Pathan

Chalmers University of Technology, Göteborg, Sweden
risat@chalmers.se

Abstract

The traditional Vestal's model of Mixed-Criticality (MC) systems was recently extended to Imprecise Mixed-Critical task model (IMC) to guarantee some minimum level of (degraded) service to the low-critical tasks even after the system switches to the high-critical behavior. This paper extends the IMC task model by associating specific Quality-of-Service (QoS) values with the low-critical tasks and proposes a fluid-based scheduling algorithm, called MCFQ, for such task model. The MCFQ algorithm allows some low-critical tasks to provide full service even during the high-critical behavior so that the QoS of the overall system is increased. To the best of our knowledge MCFQ is the first algorithm for IMC task sets considering multiprocessor platform and QoS values.

By extending the recently proposed MC-Fluid and MCF fluid-based multiprocessor scheduling algorithms for IMC task model, empirical results show that MCFQ algorithm can significantly improve the QoS of the system in comparison to that of both MC-Fluid and MCF. In addition, the schedulability performance of MCFQ is very close to the optimal MC-Fluid algorithm. Finally, we prove that the MCFQ algorithm has a speedup bound of 4/3, which is optimal for IMC tasks.

1998 ACM Subject Classification C.3 Real-Time and Embedded Systems, D.4.1 Scheduling

Keywords and phrases mixed-criticality systems, real-time systems, multiprocessor scheduling, quality of service, imprecise computation

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2017.19

1 Introduction

The computation power of multicore processors offers real-time embedded system designers the opportunity to integrate multiple components with different levels of criticality on a common hardware platform. Such Mixed-Criticality (MC) systems are often certified by certification authorities (CAs). This paper proposes a new multiprocessor scheduling algorithm for implicit-deadline dual-criticality sporadic tasks where a task is either high critical (HI) or low critical (LO).

In a dual-criticality system, the correctness of the high-critical tasks needs to be demonstrated under rigorous (often pessimistic) assumptions. Based on Vestal's model for MC tasks [21], the worst-case execution time (WCET) of each HI-critical task, according to the assumptions of the CA, is larger than or equal to that considered by the system designer. Each high-critical task τ_i has two different WCETs: $C_i^{\mathtt{L}}$ and $C_i^{\mathtt{H}}$ where $C_i^{\mathtt{L}} \leq C_i^{\mathtt{H}}$ and the WCET of each LO-critical task τ_i is $C_i^{\mathtt{L}}$. Most of the earlier works on scheduling MC systems [5, 4, 6, 17, 10, 8] consider that if some HI-critical task does not complete execution after

* This research has been funded by the MECCA project under the ERC grant ERC-2013-AdG 340328-MECCA.

executing for at most $C_i^{\mathtt{L}}$ time units – the system is said to switch from LO- to HI-critical behavior in such case – then all the LO-critical tasks are aborted (definition of critical behavior is formally presented in Section 2). Such "abortion" of the LO critical tasks may not be acceptable, for example, in many control applications as pointed out in [14]. Moreover, the system designer considers the execution of the LO-critical tasks important to achieve the mission of the system.

Some works addressed this limitation by allowing the LO-critical tasks to provide delayed results, for example, by executing them less frequently after the system switches to HI-critical behavior (e.g., weakly hard MC task model [9], elastic MC task model [20, 19, 11]). However, such delayed results may not be acceptable for some applications that prefer to have results on time even if such results are imprecise (e.g., degraded). Based on the imprecise computation model [15, 16], some recent works [14, 7, 2] have proposed a new MC task model, called the Imprecise Mixed-Critical (IMC) task model, in which each LO-critical task is also guaranteed to provide some (degraded) service even after the system switches to the HI-critical behavior. The IMC task model considers two different WCET for each task: $C_i^{\mathtt{L}}$ and $C_i^{\mathtt{H}}$ where $C_i^{\mathtt{L}} \leq C_i^{\mathtt{H}}$ if τ_i is a HI-critical task or $C_i^{\mathtt{L}} \geq C_i^{\mathtt{H}}$ if τ_i is a LO-critical task. The works in [14, 7, 2] proposed scheduling algorithms for IMC task model in which each LO-critical task executes at least $C_i^{\mathtt{H}}$ time units (i.e., it provides imprecise or degraded service rather than "no" service) during the HI-critical behavior.

The motivation of the research presented in this paper is the observation that punishing *all* the LO-critical IMC tasks by allowing them to provide only degraded service during the HI-critical behavior may not be necessary if the computing platform has slack capacity during the HI-critical behavior. No earlier work on scheduling IMC tasks considers the possibility of executing any LO-critical task to provide full service also during the HI-critical behavior. This paper proposes the Mixed-Criticality Fluid scheduling with QoS (MCFQ) algorithm for multiprocessor platform, considering a set of implicit-deadline IMC sporadic tasks, in which some (if possible all) LO-critical tasks can provide full service also during the HI-critical behavior. Allowing some of the LO-critical tasks to always provide full service would improve the overall Quality-of-Service (QoS) of the system – making the system designers "happy".

Consider an airplane or car that switches to the HI-critical behavior during its mission and all the LO-critical tasks start to provide degraded service. Such degraded service is perceived by the pilot or the driver, for example, by observing some light blinking in the cockpit/dashboard, the entertainment system being turned off, or some kind of performance loss. The pilot or driver may be uncomfortable in such situation or could even be stressed. The MCFQ algorithm considers improving such situations by allowing some LO-critical tasks to provide full service also during the HI-critical behavior.

This paper extends the IMC task model [14, 7, 2] by associating with each LO-critical task τ_i two QoS values $V_i^{\mathtt{L}}$ and $V_i^{\mathtt{H}}$ where $V_i^{\mathtt{L}} \geq V_i^{\mathtt{H}}$. The QoS value $V_i^{\mathtt{L}}$ is set to 100% based on the interpretation that if a LO-critical task τ_i is guaranteed (e.g., based on the underlying schedulability analysis) to provide full service in all possible criticality behaviors of the system, the QoS value that task τ_i provides is 100%; otherwise, the QoS value of τ_i is $V_i^{\mathtt{H}}$, which is smaller than or equal to $V_i^{\mathtt{L}}$ (the way the system designer sets these values will be presented in Section 2). The QoS values of all the high-critical tasks are assumed to be 100% for all criticality behaviors since no degradation in terms of their service is acceptable in any criticality behavior. Based on the outcome of the underlying schedulability analysis for a given MC task set, the QoS contribution of each task can be determined which in turn determines the QoS of the overall system.

The proposed MCFQ algorithm is based on a fluid-based scheduling model [12, 3, 2] in which each task τ_i has two execution rates $\theta_i^{\mathtt{L}}$ and $\theta_i^{\mathtt{H}}$ for executing task τ_i during the LO- and

HI-critical behaviors, respectively. If MCFQ is successful in determining $\theta_i^{\mathtt{L}}$ and $\theta_i^{\mathtt{H}}$, then it is guaranteed that each LO-critical task τ_i provides full and degraded service during the LO- and HI-critical behaviors of the system, respectively.

The overall objective of the MCFQ algorithm (unlike other fluid-based algorithms [12, 3]) is to maximize the sum of the LO-critical execution rates of all the tasks so that less computation is required during the HI-critical behavior. This maximization potentially implies higher slack capacity during the HI-critical behavior. Such slack is exploited to increase the HI-critical execution rate $\theta_i^{\mathtt{H}}$ of some LO-critical task so that this LO-critical task provides full service also during the HI-critical behavior, and thereby, can increase the QoS of the LO-critical task τ_i by $(V_i^{\mathtt{L}} - V_i^{\mathtt{H}})$. Given an amount of slack, the LO-critical tasks for which the HI-critical execution rates can be increased are determined based on Integer Linear Programming (ILP) to maximize the overall QoS of the system while ensuring correctness. Although the proposed MCFQ algorithm is based on a fluid-based scheduling model, there are some salient features of MCFQ that make this algorithm novel with respect to the recently proposed MC-Fluid [12] and MCF [3] algorithms. Neither the MC-Fluid [12] nor the MCF [3] algorithm considers the IMC task model, and therefore, such algorithms do not allow any LO-critical task to provide any (not even degraded) service during the HI-critical behavior. While the works in [2, 14] consider the IMC task model to allow the LO-critical tasks to provide degraded service during the HI-critical behavior, these works do not consider multiprocessors. Moreover, none of the works in [2, 14] allows any LO-critical task to provide full service in HI-critical behavior even if enough processing capacity is available. Common to all these works [12, 3, 2, 14] is that none considers maximizing the utilization of the platform during the LO-critical behavior in order to gain and exploit slack capacity during the HI-critical behavior to improve the overall QoS of the system.

This paper has the following contributions.

- First, we present an extension (i.e. generalization) of the IMC task model where each LO-critical task has two QoS values depending on whether it can provide full service in all criticality behaviors or not. This new model allows the system designers to set the values of the QoS of the LO-critical tasks based on her level of "happiness" with the degraded or full service of such tasks. Based on the QoS values of the tasks, the overall QoS of the entire system can be determined.
- Second, a new algorithm, called MCFQ, is proposed for scheduling traditional IMC task systems on a multiprocessor platform. To the best of our knowledge, MCFQ is the first multiprocessor scheduling algorithm that considers the IMC task model. The main idea of developing the MCFQ scheduling algorithm, i.e., *fully* utilizing the processors during the LO-critical behavior, has the potential to be applied to other MC scheduling algorithms that are proposed in the literature to improve the overall QoS of the system.
- Third, we formulate an ILP to select some (if possible all) LO-critical tasks such that these tasks provide full service in all the criticality behaviors of the system while maximizing the overall QoS of the system.
- We compare the schedulability of MCFQ algorithm with the recently proposed MC-Fluid [12] and MCF [3] algorithms, by extending MC-Fluid and MCF for IMC tasks, using randomly generated task sets. It is found that the MCFQ scheduling algorithm has schedulability performance very close to the optimal MC-Fluid algorithm and can significantly improve the QoS of the system in comparison to *both* MC-Fluid and MCF algorithms.
- Finally, we prove that MCFQ has a speedup bound of 4/3 which is optimal for IMC tasks.

The remainder of this paper is organized as follows: Section 2 presents the system model. Section 3 presents an overview of the proposed MCFQ algorithm. The details of MCFQ algorithm and the proof of its correctness are presented in Section 4. The formulation of the ILP to improve the QoS of the system is presented in Section 5. Experimental results are presented in Section 6 before concluding in Section 7.

2 System Model

This paper considers the scheduling of n implicit-deadline dual-criticality sporadic tasks in set $\Gamma = \{\tau_1, \dots \tau_n\}$ on m processors. Each task $\tau_i \in \Gamma$ generates an infinite sequence of jobs. Each task τ_i is represented using 4 parameters $\{T_i, L_i, \mathcal{C}_i, \mathcal{V}_i\}$ where[1]

- $T_i \in \mathbb{R}^+$ is the minimum inter-arrival time of the jobs (also, called period) of the task. The relative deadline of task τ_i is also T_i.
- $L_i \in \{\mathtt{HI}, \mathtt{LO}\}$ is the criticality of the task: LO and HI respectively specifies low- and high-critical task.
- $\mathcal{C}_i = \{C_i^{\mathtt{L}}, C_i^{\mathtt{H}}\}$ is a list of WCETs of task τ_i at different criticality levels. The WCET of task τ_i at criticality level LO and HI are respectively $C_i^{\mathtt{L}}$ and $C_i^{\mathtt{H}}$. If $L_i = \mathtt{HI}$, then $C_i^{\mathtt{H}} \geq C_i^{\mathtt{L}}$ for a HI-critical task, whereas $C_i^{\mathtt{H}} \leq C_i^{\mathtt{L}}$ for a LO-critical task.
- $\mathcal{V}_i = \{V_i^{\mathtt{L}}, V_i^{\mathtt{H}}\}$ is a list of QoS values for each LO-critical task τ_i where $V_i^{\mathtt{L}} \geq V_i^{\mathtt{H}}$. If *all* the jobs of the LO-critical task τ_i are guaranteed to execute for $C_i^{\mathtt{L}}$ time units (i.e., it provides full service in all behaviors), then task τ_i's QoS contribution is $V_i^{\mathtt{L}}$; otherwise, τ_i's QoS contribution is $V_i^{\mathtt{H}}$. Although each LO-critical task has two different QoS values, the contribution of each such task's QoS to the overall QoS of the system is dependent on the outcome of the schedulability analysis.

How are the QoS values assigned? The system designer sets the values of $V_i^{\mathtt{L}}$ and $V_i^{\mathtt{H}}$ for each LO-critical task based on how "happy" she is with the full and degraded service of the task, respectively. If each of the jobs of a LO-critical task τ_i executes for $C_i^{\mathtt{L}}$ time units, then task τ_i provides full service in all the criticality behaviors and the QoS value $V_i^{\mathtt{L}}$ of τ_i is 100%, i.e., $V_i^{\mathtt{L}} = 1.0$. On the other hand, the value of $V_i^{\mathtt{H}}$ should reflect the level of degradation of the LO-critical task τ_i if all the jobs of such a task cannot be guaranteed to execute for $C_i^{\mathtt{L}}$ time units in HI-critical behavior. Note that although the HI-critical behavior does not necessarily require a LO-critical task to execute $C_i^{\mathtt{L}}$ time units to ensure correctness (definition of correctness will be presented shortly), this paper seeks the opportunity to do so in order to improve the overall QoS of the system.

If the output quality of a LO-critical task τ_i depends on how long the task executes (i.e, the longer a task executes, the better results it produces similar to the imprecise computation models [15, 16]), then the system designer sets the QoS value $V_i^{\mathtt{H}}$ of a LO-critical task τ_i as $V_i^{\mathtt{H}} = C_i^{\mathtt{H}}/C_i^{\mathtt{L}}$. Note that $C_i^{\mathtt{H}}/C_i^{\mathtt{L}} \leq 1$ for LO-critical task τ_i. On the other hand, if the output quality of a task is not directly related to how long the task executes, then value of $V_i^{\mathtt{H}}$ is assigned by the system designer based on her own interpretation (i.e., happiness) regarding the level of degradation of the LO-critical task τ_i. The system designer assigns $V_i^{\mathtt{H}} = hpy_i$ where hpy_i is her level of happiness with the degraded service of the LO-critical task τ_i where $V_i^{\mathtt{H}} = hpy_i \leq V_i^{\mathtt{L}} = 1.0$.

Useful Definitions: The set of all the HI-critical tasks in Γ is denoted by Γ_H where $\Gamma_H = \{\tau_i \mid \tau_i \in \Gamma \text{ and } L_i = \mathtt{HI}\}$. Similarly, the set of all the LO-critical tasks in Γ is denoted

[1] Each HI-critical task is represented using 3 parameters since the required QoS values for such tasks is always 100%.

by Γ_L where $\Gamma_L = \{\tau_i \mid \tau_i \in \Gamma \text{ and } L_i = \texttt{LO}\}$. The LO and HI-critical utilization of task τ_i are defined as $u_i^{\texttt{L}} = C_i^{\texttt{L}}/T_i$ and $u_i^{\texttt{H}} = C_i^{\texttt{H}}/T_i$. For all LO-critical tasks, the total LO- and HI-critical utilizations are $U_{\texttt{L}}^{\texttt{L}} = \sum_{\forall \tau_i \in \Gamma_L} u_i^{\texttt{L}}$ and $U_{\texttt{L}}^{\texttt{H}} = \sum_{\forall \tau_i \in \Gamma_L} u_i^{\texttt{H}}$, respectively. Similarly, for all HI-critical tasks, the total LO- and HI-critical utilizations are $U_{\texttt{H}}^{\texttt{L}} = \sum_{\forall \tau_i \in \Gamma_H} u_i^{\texttt{L}}$ and $U_{\texttt{H}}^{\texttt{H}} = \sum_{\forall \tau_i \in \Gamma_H} u_i^{\texttt{H}}$, respectively.

Behavior: An MC sporadic task system shows different behaviors during different runs of the system since different jobs may be released at different time instants and may have different execution times. We assume, similar to [2], that the run-time environment budgets the execution time of the jobs generated by the LO-critical tasks such that any such job will be terminated once it consumes its budgeted amount of execution, regardless of whether it has completed execution or not. The criticality level of a behavior is determined by how much execution is needed by the HI-critical jobs to complete execution in that behavior.

If each HI-critical job of task τ_i signals completion after completing at most $C_i^{\texttt{L}}$ units of execution, then the behavior of the system is defined to be a *LO-critical behavior*. If some job of a HI-critical task τ_i does not signal completion after completing at most $C_i^{\texttt{L}}$ units of execution at time t, then the system is said to *switch* from LO- to HI-critical behavior at time t. If each job of a HI-critical task τ_i signals completion after completing at most $C_i^{\texttt{H}}$ units of execution, then the behavior of the system is defined to be a *HI-critical behavior*. All other behaviors are erroneous.

Correctness: We define an algorithm for scheduling an MC system to be correct if it is able to schedule any system in such a manner that both the following properties are satisfied:

- During all the LO-critical behaviors of the system, each HI-critical job receives enough execution between its release time and deadline to complete, and each LO-critical job either completes or receives at least its LO-critical WCET, between its release time and deadline.
- During all the HI-critical behaviors of the system, each HI-critical job receives enough execution between its release time and deadline to complete, and each LO-critical job of a LO-critical task either completes or receives **at least** its HI-critical WCET, between its release time and deadline.

The proposed MCFQ algorithm first seeks to find execution rates of the tasks to ensure the correctness of the system. As it is evident from the definition of correctness that if a system is correct, then it is ensured that each LO-critical task provides degraded service during the HI-critical behavior and contributes a QoS of $V_i^{\texttt{H}}$ to the overall QoS of the system. Given that a system is correct, we exploit slack of the processors in HI-critical behavior to select some LO-critical tasks so that these tasks are guaranteed to provide full service even during the HI-critical behavior – improving the QoS of the LO-critical task τ_i from $V_i^{\texttt{H}}$ to $V_i^{\texttt{L}}$.

3 An overview of MCFQ Scheduling Algorithm

The MCFQ algorithm is based on fluid-based scheduling [12, 3] in which a task may be assigned a fraction ≤ 1 of a processor, called the *execution rate* of the task, at each time instant. The MCFQ algorithm prior to runtime determines the LO- and HI-critical execution rates, denoted respectively by $\theta_i^{\texttt{L}}$ and $\theta_i^{\texttt{H}}$, for each task $\tau_i \in \Gamma$. The execution rates $\theta_i^{\texttt{L}}$ and $\theta_i^{\texttt{H}}$ for each task τ_i are computed such that the run-time scheduling strategy presented in Figure 1 constitutes a correct scheduling strategy for task set Γ. According to the algorithm in Figure 1, each task

- Each task τ_i initially executes at a constant rate $\theta_i^{\mathtt{L}}$. That is, at each time instant it is executing upon $\theta_i^{\mathtt{L}}$ fraction of a processor.
- If a job of task $\tau_i \in \Gamma_H$ does not complete despite having received $C_i^{\mathtt{L}}$ units of execution (equivalently, having executed for a duration $C_i^{\mathtt{L}}/\theta_i^{\mathtt{L}}$), then each task τ_i executes at a constant rate $\theta_i^{\mathtt{H}}$. That is, at each time instant it is executing upon $\theta_i^{\mathtt{H}}$ fraction of a processor.

Figure 1 Run-time scheduling strategy originally proposed in [2] for uniprocessor is also applicable to multiprocessors.

τ_i is executed with execution rate $\theta_i^{\mathtt{L}}$ during the `LO`-critical behavior of the system. Once the system switches to `HI`-critical behavior, τ_i executes with execution rate $\theta_i^{\mathtt{H}}$.

The system can switch back (not considered in this paper) from `HI`- to `LO`-critical behavior when there is an idle period and no job of any `HI`-critical task awaits for execution as is proposed by Santy et al [18]. Transforming the fluid schedule generated by `MCFQ` algorithm to construct a (non-fluid) schedule for real hardware can be done using the MC-DP-Fair algorithm proposed in [12].

We present the execution-rate assignment strategy of `MCFQ` in Figure 2 in Section 4. It will be proved in subsection 4.1 that if the `MCFQ` algorithm successfully determines the execution rates $\theta_i^{\mathtt{L}}$ and $\theta_i^{\mathtt{H}}$, then the system is correct. Given that an `MC` system is correct using the execution rates determined by the `MCFQ` algorithm in Figure 2, we then consider increasing the `HI`-critical execution rates $\theta_i^{\mathtt{H}}$ of some `LO`-critical tasks to increase the QoS of the system in Section 5. The following lemmas and definitions will be used in Section 4.

Lemma 1, derived in [12], states a necessary and sufficient schedulability condition of a `HI`-critical task τ_k during the `HI`-critical behavior (including the particular scenario when the system switches from `LO`- to `HI`-critical behavior) assuming that the task is schedulable in `LO`-critical behavior. The condition in Eq. (1) is derived regardless of any parameters of the `LO`-critical tasks and is thus also applicable to IMC task systems.

▶ **Lemma 1** (From Lemma 5 in [12]). *Given a `HI`-critical task τ_k satisfying task-schedulability in `LO`-critical behavior, the task can meet its deadline if and only if*

$$u_k^{L}/\theta_k^{L} + (u_k^{H} - u_k^{L})/\theta_k^{H} \;\leq\; 1 \tag{1}$$

Based on Lemma 1, a lower bound on $\theta_k^{\mathtt{L}}$ for each `HI`-critical task $\tau_k \in \Gamma_H$ is given in Lemma 2.

▶ **Lemma 2.** *If the execution rates θ_k^{L} and θ_k^{H} of a `HI`-critical task $\tau_k \in \Gamma_H$ guarantees that all the jobs of τ_k meet their deadlines in all the correct behaviors of the system, then the following holds:*

$$\theta_k^{L} \;\geq\; u_k^{L}/(1 - u_k^{H} + u_k^{L}) \;\geq\; u_k^{L} \tag{2}$$

Proof. Since $\tau_k \in \Gamma_H$ meets its deadline, the following (from Eq. (1) of Lemma 1) holds:

$$\begin{aligned}
& u_k^{\mathtt{L}}/\theta_k^{\mathtt{L}} + (u_k^{\mathtt{H}} - u_k^{\mathtt{L}})/\theta_i^{\mathtt{H}} \;\leq\; 1 \\
\Rightarrow\;& u_k^{\mathtt{L}}/\theta_k^{\mathtt{L}} + (u_k^{\mathtt{H}} - u_k^{\mathtt{L}}) \;\leq\; 1 \qquad\qquad \text{(Since } \theta_k^{\mathtt{H}} \leq 1 \text{ for any execution rate to be valid)} \\
\Leftrightarrow\;& \theta_k^{\mathtt{L}} \;\geq\; u_k^{\mathtt{L}}/(1 - u_k^{\mathtt{H}} + u_k^{\mathtt{L}})
\end{aligned}$$

For a `HI`-critical task τ_k, we have $0 \leq (1 - u_k^{\mathtt{H}} + u_k^{\mathtt{L}}) \leq 1$ because $1 \geq u_k^{\mathtt{H}} \geq u_k^{\mathtt{L}}$. Therefore, $u_i^{\mathtt{L}}/(1 - u_k^{\mathtt{H}} + u_k^{\mathtt{L}}) \geq u_k^{\mathtt{L}}$ and we have $\theta_k^{\mathtt{L}} \;\geq\; u_k^{\mathtt{L}}/(1 - u_k^{\mathtt{H}} + u_k^{\mathtt{L}}) \;\geq\; u_k^{\mathtt{L}}$. ◀

Assumptions: $(\overline{U}^{\mathtt{L}}_{\mathtt{H}} + U^{\mathtt{L}}_{\mathtt{L}}) \le m$, $(U^{\mathtt{H}}_{\mathtt{H}} + U^{\mathtt{H}}_{\mathtt{L}}) \le m$, $max\{u^{\mathtt{H}}_i, u^{\mathtt{L}}_i\} \le 1$ for all $\tau_i \in \Gamma$

1. $\theta^{\mathtt{H}}_i = u^{\mathtt{H}}_i$ and $\theta^{\mathtt{L}}_i = u^{\mathtt{L}}_i$ for all $\tau_i \in \Gamma_L$.

2. For $i = 1$ to h //Tasks in Γ_H are indexed from $1 \dots h$

$$\theta^{\mathtt{L}}_i = min\{u^{\mathtt{H}}_i, \mathcal{F}_{i-1} \cdot \overline{u}^{\mathtt{L}}_i\} \tag{5}$$

$$\theta^{\mathtt{H}}_i = (u^{\mathtt{H}}_i - u^{\mathtt{L}}_i)/(1 - \frac{u^{\mathtt{L}}_i}{\theta^{\mathtt{L}}_i}) \tag{6}$$

3. If

$$\sum_{\tau_k \in \Gamma} \theta^{\mathtt{H}}_k \le m \textbf{ and } \sum_{\tau_k \in \Gamma} \theta^{\mathtt{L}}_k \le m \tag{7}$$

then declare success else declare failure.

Figure 2 Execution Rate Assignment.

Lemma 2 essentially states that if an MC task set is schedulable in all the correct behaviors of the system based on the runtime scheduling strategy in Figure 1, then it is **necessary** that the LO-critical execution rate $\theta^{\mathtt{L}}_k$ for each HI-critical task $\tau_k \in \Gamma_H$ must not be smaller than $\frac{u^{\mathtt{L}}_k}{1-u^{\mathtt{H}}_k+u^{\mathtt{L}}_k}$. We denote by $\overline{u}^{\mathtt{L}}_k$ the lower bound on the LO-critical execution rate of the HI-critical task $\tau_k \in \Gamma_H$:

$$\overline{u}^{\mathtt{L}}_k = u^{\mathtt{L}}_k/(1 - u^{\mathtt{H}}_k + u^{\mathtt{L}}_k)\,. \tag{3}$$

We also define $\overline{U}^{\mathtt{L}}_{\mathtt{H}}$ as follows:

$$\overline{U}^{\mathtt{L}}_{\mathtt{H}} = \sum_{\tau_k \in \Gamma_H} \overline{u}^{\mathtt{L}}_k = \sum_{\tau_k \in \Gamma_H} \frac{u^{\mathtt{L}}_k}{1 - u^{\mathtt{H}}_k + u^{\mathtt{L}}_k}\,. \tag{4}$$

4 Execution Rate Assignment of MCFQ

If $(U^{\mathtt{H}}_{\mathtt{H}} + U^{\mathtt{H}}_{\mathtt{L}}) > m$ or $(U^{\mathtt{L}}_{\mathtt{L}} + \overline{U}^{\mathtt{L}}_{\mathtt{H}}) > m$ or $max\{u^{\mathtt{H}}_k, u^{\mathtt{L}}_k\} > 1$ for any task $\tau_k \in \Gamma$, then no algorithm can schedule such a task set. The MCFQ algorithm considers task sets where $(U^{\mathtt{H}}_{\mathtt{H}} + U^{\mathtt{H}}_{\mathtt{L}}) \le m$ and $(U^{\mathtt{L}}_{\mathtt{L}} + \overline{U}^{\mathtt{L}}_{\mathtt{H}}) \le m$ and $max\{u^{\mathtt{H}}_k, u^{\mathtt{L}}_k\} \le 1$ for each task $\tau_k \in \Gamma$.

The execution rate assignment algorithm of MCFQ is given in Figure 2. The LO- and the HI-critical execution rates $\theta^{\mathtt{L}}_i$ and $\theta^{\mathtt{H}}_i$ of **each LO-critical task** is equal to its LO- and HI-critical utilizations $u^{\mathtt{L}}_i$ and $u^{\mathtt{H}}_i$, respectively. In Step 1 of the algorithm in Figure 2, we assign $\theta^{\mathtt{L}}_i = u^{\mathtt{L}}_i$ and $\theta^{\mathtt{H}}_i = u^{\mathtt{H}}_i$ for each LO-critical task $\tau_i \in \Gamma_L$.

The LO- and HI-critical execution rates to each **HI-critical task** is assigned in Step 2 in Figure 2. Let the HI-critical tasks in set Γ_H are indexed[2] from 1 to h. The value of the LO-critical execution rate $\theta^{\mathtt{L}}_i$ of a HI-critical task $\tau_i \in \Gamma_H$ is assigned in Eq. (5) such that $\theta^{\mathtt{L}}_i = min\{u^{\mathtt{H}}_i, \mathcal{F}_{i-1} \cdot \overline{u}^{\mathtt{L}}_i$ based on a *threshold*, denoted by $\mathcal{F}_{i-1}$, which is defined in Eq. (8). Notice that the value $\theta^{\mathtt{L}}_i$ of the i^{th} HI-critical task $\tau_i \in \Gamma_H$ is assigned based on the

[2] The execution rate-assignment algorithm in Figure 2 works for any arbitrary order of considering the HI-critical tasks when assigning their execution rates in Step 2. However, sorting the HI-critical tasks in increasing order of $u^{\mathtt{H}}_i/\overline{u}^{\mathtt{L}}_i$ has schedulability performance very close to the optimal MC-Fluid algorithm (shown in Section 6).

Table 1 An example dual-criticality IMC taskset.

τ_i	T_i	$C_i^{\mathtt{L}}$	$C_i^{\mathtt{H}}$	L_i	$u_i^{\mathtt{L}}$	$u_i^{\mathtt{H}}$	$\overline{u}_i^{\mathtt{L}}$	$\mathcal{F}_{i-1}$	$V_i^{\mathtt{H}}$
τ_1	20	7	13	HI	0.35	0.65	0.5	13/9	-
τ_2	10	2	7	HI	0.2	0.7	0.4	13/8	-
τ_3	40	8	5	LO	0.2	0.125	-	-	0.6
τ_4	60	30	12	LO	0.5	0.2	-	-	0.4

$(i-1)^{th}$ threshold (i.e, $\mathcal{F}_{i-1}$) for $i = 1, \ldots h$. After the value of $\theta_i^{\mathtt{L}}$ is assigned using Eq. (5), the HI-critical execution rate $\theta_i^{\mathtt{H}}$ is assigned in Eq. (6) based on the value of $\theta_i^{\mathtt{L}}$ such that $\theta_i^{\mathtt{H}} = (u_i^{\mathtt{H}} - u_i^{\mathtt{L}})/(1 - u_i^{\mathtt{L}}/\theta_i^{\mathtt{L}})$.

If $\sum_{\tau_k \in \Gamma} \theta_k^{\mathtt{H}} \le m$ and $\sum_{\tau_k \in \Gamma} \theta_k^{\mathtt{L}} \le m$, we declare success; otherwise, we declare failure in Step 3. We will prove in subsection 4.1 that if the algorithm declares success, then the run-time scheduling strategy in Figure 1 constitutes a correct scheduling strategy.

Threshold $\mathcal{F}_i$: The value of threshold $\mathcal{F}_i$ in Eq. (8) for task τ_{i+1} is derived as follows. Initially, $\mathcal{F}_0 = (m - U_{\mathtt{L}}^{\mathtt{L}})/\overline{U}_{\mathtt{H}}^{\mathtt{L}}$. Task $\tau_1 \in \Gamma_H$ in Eq. (5) uses $\mathcal{F}_0$. The value of $\mathcal{F}_i$ for task $\tau_{i+1} \in \Gamma_H$ is recursively defined as follows for $i = 1, \ldots (h-1)$:

$$\mathcal{F}_i = \begin{cases} \frac{m - U_{\mathtt{L}}^{\mathtt{L}}}{\overline{U}_{\mathtt{H}}^{\mathtt{L}}} & \text{if } i = 0 \\ max\{\mathcal{F}_{i-1}, \frac{m - U_{\mathtt{L}}^{\mathtt{L}} - \sum_{k=1}^{i} u_k^{\mathtt{H}}}{(\overline{U}_{\mathtt{H}}^{\mathtt{L}} - \sum_{k=1}^{i} \overline{u}_k^{\mathtt{L}})}\} & \text{if } i > 0 \end{cases} \tag{8}$$

▶ **Example 3.** Consider the task set in Table 1 where $m = 2$. The last column shows the $V_i^{\mathtt{H}}$ values for the two LO-critical tasks τ_3 and τ_4 where $V_3^{\mathtt{H}} = hpy_3 = 0.6 \ne C_3^{\mathtt{H}}/C_3^{\mathtt{L}}$ and $V_4^{\mathtt{H}} = 0.4 = C_4^{\mathtt{H}}/C_4^{\mathtt{L}} = 12/30$. Note that $V_4^{\mathtt{H}}$ is assigned based on the imprecise computation model while $V_3^{\mathtt{H}}$ is not assigned based on the imprecise computation model. The values of $\overline{u}_i^{\mathtt{L}}$ and F_{i-1} that are required to compute the execution rates of the HI-critical tasks are shown in the eighth and ninth columns, respectively.

For the task set in Table 1, we have

$U_{\mathtt{H}}^{\mathtt{H}} = 0.65 + 0.7 = 1.35$ $\quad U_{\mathtt{L}}^{\mathtt{L}} = 0.2 + 0.5 = 0.7$ $\quad U_{\mathtt{L}}^{\mathtt{H}} = 0.125 + 0.2 = 0.325$

$\overline{U}_{\mathtt{H}}^{\mathtt{L}} = 0.5 + 0.4 = 0.9$ $\quad (U_{\mathtt{H}}^{\mathtt{H}} + U_{\mathtt{L}}^{\mathtt{H}}) = 1.675 \le m$ $\quad (U_{\mathtt{L}}^{\mathtt{L}} + \overline{U}_{\mathtt{H}}^{\mathtt{L}}) = 1.6 \le m$

Also note that $(U_{\mathtt{L}}^{\mathtt{L}} + U_{\mathtt{H}}^{\mathtt{H}}) = (1.35 + 0.7) = 2.05 > m$, which implies such a task set cannot allow both the LO-critical tasks to provide full service during the HI-critical behavior. Now we present how the values of $\mathcal{F}_{i-1}$ are computed for $i = 1, 2$ since there are two HI-critical tasks (i.e, $h = 2$).

$\mathcal{F}_0 = \frac{m - U_{\mathtt{L}}^{\mathtt{L}}}{\overline{U}_{\mathtt{H}}^{\mathtt{L}}} = \frac{2-0.7}{0.9} = 13/9$ and $\mathcal{F}_1 = max\{\mathcal{F}_0, \frac{m - U_{\mathtt{L}}^{\mathtt{L}} - u_1^{\mathtt{H}}}{\overline{U}_{\mathtt{H}}^{\mathtt{L}} - \overline{u}_1^{\mathtt{L}}}\} = max\{13/9, \frac{2-0.7-0.65}{0.9-0.5}\} = 13/8$

The LO-critical tasks τ_3 and τ_4 get values of $\theta_i^{\mathtt{L}}$ and $\theta_i^{\mathtt{H}}$ based on Step 1 in Figure 2 as follows: $\theta_3^{\mathtt{L}}$=$u_3^{\mathtt{L}}$=0.2, $\theta_3^{\mathtt{H}}$=$u_3^{\mathtt{H}}$=0.125 and $\theta_4^{\mathtt{L}}$=$u_4^{\mathtt{L}}$=0.5, and $\theta_4^{\mathtt{H}}$=$u_4^{\mathtt{H}}$=0.2. The HI-critical tasks τ_1 and τ_2 get values of $\theta_i^{\mathtt{L}}$ based on Eq. (5) as follows:

$\theta_1^{\mathtt{L}} = min\{u_1^{\mathtt{H}}, \mathcal{F}_0 \cdot \overline{u}_1^{\mathtt{L}}\} = min\{0.65, 13/9 \times 0.5\} = 0.65$

$\theta_2^{\mathtt{L}} = min\{u_2^{\mathtt{H}}, \mathcal{F}_1 \cdot \overline{u}_2^{\mathtt{L}}\} = min\{0.7, 13/8 \times 0.4\} = 0.65$

The HI-critical tasks τ_1 and τ_2 get values of $\theta_i^{\mathtt{H}}$ based on Eq. (6) as follows:

$\theta_1^{\mathtt{H}} = \frac{u_1^{\mathtt{H}} - u_1^{\mathtt{L}}}{1 - \frac{u_1^{\mathtt{L}}}{\theta_1^{\mathtt{L}}}} = \frac{0.65-0.35}{1-\frac{0.35}{0.65}} = 13/20$ $\quad \theta_2^{\mathtt{H}} = \frac{u_2^{\mathtt{H}} - u_2^{\mathtt{L}}}{1 - \frac{u_2^{\mathtt{L}}}{\theta_2^{\mathtt{L}}}} = \frac{0.7-0.2}{1-\frac{0.2}{0.65}} = 13/18$

Since $\sum_{i=1}^{4} \theta_i^{\mathtt{L}} = 0.65 + 0.65 + 0.2 + 0.5 = 2 \le m$ and $\sum_{i=1}^{4} \theta_i^{\mathtt{H}} = 13/20 + 13/18 + 0.125 + 0.2 = 1.6972 \le m$, we conclude that the MCFQ algorithm returns success.

Note that the sum of the LO-critical execution rates is m (i.e., full capacity of the platform) while the sum of the HI-critical execution rates is 1.6972. The slack capacity in HI-critical behavior is $(m - 1.6972) = 0.3027$. ◀

To prove the correctness of the MCFQ algorithm, the following lemmas will be used.

▶ **Lemma 4.** *Consider the tasks in Γ_H are indexed from 1 to h. If Γ is feasible, then*

$$1 \leq \mathcal{F}_0 \leq \mathcal{F}_1 \leq \ldots \leq \mathcal{F}_{h-1} \,. \tag{9}$$

Proof. We prove this lemma using induction on $i = 0, 1, \ldots (h-1)$. Since Γ is feasible, it is necessary that $(U_{\mathtt{L}}^{\mathtt{L}} + \overline{U}_{\mathtt{H}}^{\mathtt{L}}) \leq m$. In other words, $\mathcal{F}_0 = \frac{m - U_{\mathtt{L}}^{\mathtt{L}}}{\overline{U}_{\mathtt{H}}^{\mathtt{L}}} \geq 1$. Now assume that $1 \leq \mathcal{F}_0 \leq \mathcal{F}_1 \ldots \leq \mathcal{F}_{i-1}$ for some i where $i < (h-1)$. From Eq. (8), we have

$$\mathcal{F}_i = max\{\mathcal{F}_{i-1}, \frac{m - U_{\mathtt{L}}^{\mathtt{L}} \; - \sum_{k=1}^{i} u_k^{\mathtt{H}}}{(\overline{U}_{\mathtt{H}}^{\mathtt{L}} - \sum_{k=1}^{i} \overline{u}_k^{\mathtt{L}})}\} \geq \mathcal{F}_{i-1} \,.$$

Therefore, we have $1 \leq \mathcal{F}_0 \leq \mathcal{F}_1 \ldots \leq \mathcal{F}_{i-1} \leq \mathcal{F}_i$. Consequently, Eq. (9) holds. ◀

▶ **Lemma 5.** *Consider a LO-critical task $\tau_i \in \Gamma_L$. We have $u_i^L \leq \theta_i^L \leq 1$ and $u_i^H \leq \theta_i^H \leq 1$.*

Proof. For each LO-critical task $\tau_i \in \Gamma_L$, we have $\theta_i^{\mathtt{L}} = u_i^{\mathtt{L}}$ and $\theta_i^{\mathtt{H}} = u_i^{\mathtt{H}}$ according to Step 1 in Figure 2. Since $u_i^{\mathtt{H}} \leq 1$ and $u_i^{\mathtt{L}} \leq 1$ (necessary conditions for schedulability), we also have that $u_i^{\mathtt{L}} \leq \theta_i^{\mathtt{L}} \leq 1$ and $u_i^{\mathtt{H}} \leq \theta_i^{\mathtt{H}} \leq 1$ for all $\tau_i \in \Gamma_L$. ◀

▶ **Lemma 6.** *Consider a HI-critical task $\tau_i \in \Gamma_H$. We have We have $u_i^L \leq \theta_i^L \leq 1$ and $u_i^H \leq \theta_i^H \leq 1$.*

Proof. From Eq. (5), we have $\theta_i^{\mathtt{L}} = min\{u_i^{\mathtt{H}}, \mathcal{F}_{i-1} \cdot \overline{u}_i^{\mathtt{L}}\}$ for $\tau_i \in \Gamma_H$. We will prove this lemma considering two cases: case (i) $\theta_i^{\mathtt{L}} = u_i^{\mathtt{H}}$, and case (ii) $\theta_i^{\mathtt{L}} = \mathcal{F}_{i-1} \cdot \overline{u}_i^{\mathtt{L}}$.

Case (i): $\boldsymbol{\theta_i^{\mathtt{L}} = u_i^{\mathtt{H}}}$. In such case, from Eq. (6) we have

$$\theta_i^{\mathtt{H}} = (u_i^{\mathtt{H}} - u_i^{\mathtt{L}})/(1 - \frac{u_i^{\mathtt{L}}}{\theta_i^{\mathtt{L}}}) = (u_i^{\mathtt{H}} - u_i^{\mathtt{L}})/(1 - \frac{u_i^{\mathtt{L}}}{u_i^{\mathtt{H}}}) = u_i^{\mathtt{H}} \,. \tag{10}$$

Therefore, $\theta_i^{\mathtt{L}} = \theta_i^{\mathtt{H}} = u_i^{\mathtt{H}}$. Since $1 \geq u_i^{\mathtt{H}} \geq u_i^{\mathtt{L}}$ for a HI-critical task $\tau_i \in \Gamma_H$, we have $1 \geq \theta_i^{\mathtt{L}} \geq u_i^{\mathtt{L}}$ and $1 \geq \theta_i^{\mathtt{H}} \geq u_i^{\mathtt{H}}$.

Case (ii): $\boldsymbol{\theta_i^{\mathtt{L}} = \mathcal{F}_{i-1} \cdot \overline{u}_i^{\mathtt{L}}}$. Since $\theta_i^{\mathtt{L}} = min\{u_i^{\mathtt{H}}, \mathcal{F}_{i-1} \cdot \overline{u}_i^{\mathtt{L}}\}$ in Eq. (5) and $\theta_i^{\mathtt{L}} = \overline{u}_i^{\mathtt{L}} \cdot \mathcal{F}_{i-1}$ for this case, it follows that $\theta_i^{\mathtt{L}} = \mathcal{F}_{i-1} \cdot \overline{u}_i^{\mathtt{L}} \leq u_i^{\mathtt{H}} \leq 1$. Because $\mathcal{F}_{i-1} \geq 1$ according to Eq. (9), we have $\theta_i^{\mathtt{L}} = \mathcal{F}_{i-1} \cdot \overline{u}_i^{\mathtt{L}} \geq \overline{u}_i^{\mathtt{L}}$. Consequently, the following holds

$$1 \geq u_i^{\mathtt{H}} \geq \theta_i^{\mathtt{L}} = (\overline{u}_i^{\mathtt{L}} \cdot \mathcal{F}_{i-1}) \geq \overline{u}_i^{\mathtt{L}} \,. \tag{11}$$

Since $0 < (1 - u_i^{\mathtt{H}} + u_i^{\mathtt{L}}) \leq 1$ for a HI-critical task $\tau_i \in \Gamma_H$, from Eq. (3) we have $\overline{u}_i^{\mathtt{L}} = u_i^{\mathtt{L}}/(1 - u_i^{\mathtt{H}} + u_i^{\mathtt{L}}) \geq u_i^{\mathtt{L}}$. Then from Eq. (11) and based on the fact that $\overline{u}_i^{\mathtt{L}} \geq u_i^{\mathtt{L}}$, it follows that $1 \geq \theta_i^{\mathtt{L}} \geq u_i^{\mathtt{L}}$ for this case. Now we will show that $1 \geq \theta_i^{\mathtt{H}} \geq u_i^{\mathtt{H}}$ holds. From Eq. (6) we have

$$
\begin{aligned}
& \theta_i^{\mathtt{H}} = (u_i^{\mathtt{H}} - u_i^{\mathtt{L}})/(1 - \frac{u_i^{\mathtt{L}}}{\theta_i^{\mathtt{L}}}) \\
& \text{(From Eq. (11), } u_i^{\mathtt{H}} \geq \theta_i^{\mathtt{L}} \geq \overline{u}_i^{\mathtt{L}} \text{ for this case)} \\
\Rightarrow\; & (u_i^{\mathtt{H}} - u_i^{\mathtt{L}})/(1 - \frac{u_i^{\mathtt{L}}}{u_i^{\mathtt{H}}}) \;\leq\; \theta_i^{\mathtt{H}} \;\leq\; (u_i^{\mathtt{H}} - u_i^{\mathtt{L}})/(1 - \frac{u_i^{\mathtt{L}}}{\overline{u}_i^{\mathtt{L}}}) \\
\Leftrightarrow\; & u_i^{\mathtt{H}} \;\leq\; \theta_i^{\mathtt{H}} \;\leq\; (u_i^{\mathtt{H}} - u_i^{\mathtt{L}})/(1 - \frac{u_i^{\mathtt{L}}}{\overline{u}_i^{\mathtt{L}}}) \\
& \text{(From Eq. (3), we have } \overline{u}_i^{\mathtt{L}} = u_i^{\mathtt{L}}/(1 - u_i^{\mathtt{H}} + u_i^{\mathtt{L}})) \\
\Leftrightarrow\; & u_i^{\mathtt{H}} \;\leq\; \theta_i^{\mathtt{H}} \;\leq\; (u_i^{\mathtt{H}} - u_i^{\mathtt{L}})/(1 - \frac{u_i^{\mathtt{L}}}{u_i^{\mathtt{L}}/(1 - u_i^{\mathtt{H}} + u_i^{\mathtt{L}})}) \\
\Leftrightarrow\; & u_i^{\mathtt{H}} \;\leq\; \theta_i^{\mathtt{H}} \;\leq\; (u_i^{\mathtt{H}} - u_i^{\mathtt{L}})/(1 - \frac{1}{1/(1 - u_i^{\mathtt{H}} + u_i^{\mathtt{L}})}) \\
\Leftrightarrow\; & u_i^{\mathtt{H}} \;\leq\; \theta_i^{\mathtt{H}} \;\leq\; 1\,.
\end{aligned}
$$

◀

4.1 Algorithm MCFQ: Proof of Correctness

In this subsection, Theorem 8 proves that the MCFQ algorithm presented in Figure 2 is correct. Before presenting Theorem 8, we show in Lemma 7 that the execution rates $\theta_i^{\mathtt{L}}$ and $\theta_i^{\mathtt{H}}$ that are computed by MCFQ in Figure 2 ensure the correctness in HI-critical behavior for both LO- and HI-critical tasks by analyzing the special case in which during runtime it is detected that some job of a HI-critical task has executed beyond its LO-critical execution time (i.e., criticality of the system is switched).

▶ **Lemma 7.** *Assume that the system is schedulable in stable LO-critical behavior. Let t_o denote the first time instant at which some job of a HI-critical task does not signal completion despite having executed for its LO-critical WCET. Any LO or any HI critical task that is active (has been released but not completed execution equal to its HI-critical execution) at time instant t_o receives an amount of execution no smaller than its HI-critical WCET prior to its deadline.*

Proof. We will show that this lemma holds for both LO-critical tasks and HI-critical tasks.

LO-critical task. Suppose a job of a LO-critical task τ_i is active at time t_o. Recall from Step 1 of the algorithm in Figure 2 that the LO- and HI-critical execution rates are set as $\theta_i^{\mathtt{L}} = u_i^{\mathtt{L}}$ and $\theta_i^{\mathtt{H}} = u_i^{\mathtt{H}}$, where $u_i^{\mathtt{H}} \leq u_i^{\mathtt{L}}$ for each LO-critical task τ_i. Therefore, it is guaranteed that each job of τ_i will receive *at least* execution rate $u_i^{\mathtt{H}}$ from its release to its deadline. Consequently, the LO-critical active job at time t_o is guaranteed to complete its $C_i^{\mathtt{H}}$ units of execution by its deadline.

HI-critical task. Suppose a job of a HI-critical task τ_i is active at time t_o. Lemma 1 (originally proved as Lemma 5 in [12]) states that if $u_i^{\mathtt{L}}/\theta_i^{\mathtt{L}} + (u_i^{\mathtt{H}} - u_i^{\mathtt{L}})/\theta_i^{\mathtt{H}} \leq 1$ and the HI-critical task τ_i is schedulable in (stable) LO-critical behavior, then an active job of task τ_i at time t_o also meets its deadline. The MCFQ algorithm in Eq. (6) assigns the HI-critical execution rate $\theta_i^{\mathtt{H}}$ of task τ_i based on the value of $\theta_i^{\mathtt{L}}$ such that $\theta_i^{\mathtt{H}} = (u_i^{\mathtt{H}} - u_i^{\mathtt{L}})/(1 - u_i^{\mathtt{L}}/\theta_i^{\mathtt{L}})$ which implies $u_i^{\mathtt{L}}/\theta_i^{\mathtt{L}} + (u_i^{\mathtt{H}} - u_i^{\mathtt{L}})/\theta_i^{\mathtt{H}} \leq 1$. Therefore, any active job of the HI-critical task τ_i at time t_o meets its deadline under MCFQ. ◀

▶ **Theorem 8.** *If the condition in Eq.* (7) *of the* MCFQ *algorithm in Figure 2 is satisfied, then the execution rates assigned to the tasks constitute an* MC*-correct scheduling strategy.*

Proof. Lemma 5 and Lemma 6 together show that the values of $\theta_i^{\mathtt{L}}$ and $\theta_i^{\mathtt{H}}$ for each task $\tau_i \in \Gamma$ are larger than or equal to $u_i^{\mathtt{L}}$ and $u_i^{\mathtt{H}}$, respectively. In addition, Eq. (7) ensures that the individual sum of the LO- and HI-critical execution rates for all tasks is not larger than the capacity of the platform. Therefore, the system is correct for both stable LO- and HI-critical behaviors (i.e., when the system is not switching its criticality). Finally, Lemma 7 shows the correctness of the system upon transition from LO- to HI-critical behavior. ◀

The MCFQ algorithm has a speedup bound of 4/3 which is optimal for IMC task set. The proof of the speedup bound is given in Theorem 11 in Appendix A.

5 QoS Oriented Scheduling

When the MCFQ algorithm in Figure 2 declares "success", then the system is correct according to Theorem 8 and each LO-critical task has enough execution budget to provide degraded service during the HI-critical behavior of the system. In other words, each job of the LO-critical task τ_i executes for at most $C_i^{\mathtt{H}}$ time units during the HI-critical behavior and contributes a QoS value of $V_i^{\mathtt{H}}$ to the overall QoS of the system, where $V_i^{\mathtt{H}} \leq V_i^{\mathtt{L}}$ since $C_i^{\mathtt{H}} \leq C_i^{\mathtt{L}}$.

The question we investigate in this section is the following: If the system designer is not happy with the degraded service of the LO-critical tasks in HI-critical behavior, how can we make them happier? To answer this question we investigate the possibility of allowing some/all of the LO-critical tasks to provide full service during the HI-critical behavior of the system while ensuring correctness. If $(U_{\mathtt{H}}^{\mathtt{H}} + U_{\mathtt{L}}^{\mathtt{L}}) \leq m$, then each LO- and HI-critical task can be assigned $\theta_i^{\mathtt{L}} = \theta_i^{\mathtt{H}} = \max\{u_i^{\mathtt{L}}, u_i^{\mathtt{H}}\}$ and 100% QoS is achieved in all possible behaviors. Our proposed QoS-oriented scheduling in this section is applicable to task sets even when $(U_{\mathtt{H}}^{\mathtt{H}} + U_{\mathtt{L}}^{\mathtt{L}}) > m$.

If the algorithm in Figure 2 returns success, then slack in utilization during the HI-critical behavior is defined as

$$\mathcal{S} = (m - \sum_{\tau_i \in \Gamma} \theta_i^{\mathtt{H}})\,. \tag{12}$$

Recall that MCFQ algorithm in Step 1 sets $\theta_i^{\mathtt{H}} = u_i^{\mathtt{H}}$ where $u_i^{\mathtt{H}} \leq u_i^{\mathtt{L}}$ for each LO-critical task $\tau_i \in \Gamma_L$. We will try to distribute the slack $\mathcal{S}$ to guarantee execution budget for some *selected* LO-critical tasks such that these tasks provide full service (i.e., execute $C_i^{\mathtt{L}}$ time units) also during the HI-critical behavior. In other words, the execution rates assigned to the variables $\theta_i^{\mathtt{H}}$ for the selected LO-critical tasks will be set to $u_i^{\mathtt{L}}$ rather than $u_i^{\mathtt{H}}$ so that τ_i provides full service during the HI-critical behavior. If some LO-critical tasks provides full service – due to their execution rates $\theta_i^{\mathtt{H}}$ being upgraded – such that the total increase in HI-critical execution rates in comparison to that is computed by the MCFQ algorithm is not larger than slack $\mathcal{S}$, then the correctness of the system is not compromised. This is because the execution rate of *no* HI-critical task is modified and the sum of HI-critical execution rates is still $\leq m$.

Consider the Example 3 in Section 4. The sum of the HI-critical execution rates is 1.6972 and the slack is 0.3027 where $m = 2$. The LO-critical task τ_3 is assigned execution rate $\theta_3^{\mathtt{H}} = u_3^{\mathtt{H}} = 0.125$ by algorithm MCFQ and provides degraded service during the HI-critical behavior. By increasing $\theta_3^{\mathtt{H}}$ from 0.125 to $u_3^{\mathtt{L}} = 0.2$, we can guarantee that τ_3 provides full service in all behaviors. In such case, the total increase in HI-critical execution rates is $(0.2 - 0.125) = 0.075$. Since we have a slack of 0.3027, the increase in execution rate of $\theta_3^{\mathtt{H}}$ by

0.075 will not violate the correctness. In such case, the QoS of the system is increased by $V_3^{\texttt{L}} - V_3^{\texttt{H}} = 1 - 0.6 = 0.4$ where $V_3^{\texttt{L}} = 1.0$.

Similarly, the LO-critical task τ_4's execution rate $\theta_4^{\texttt{H}} = u_4^{\texttt{H}} = 0.2$ can also be increased to $\theta_4^{\texttt{H}} = u_4^{\texttt{L}} = 0.5$. In such case, the HI-critical execution rate is increased by $(0.5 - 0.2) = 0.3$ which is less than the slack 0.3027 and the correctness of the system is not violated. In such case, the QoS of the system will be increased by $V_4^{\texttt{L}} - V_4^{\texttt{H}} = 1 - 0.4 = 0.6$. However, we cannot increase both $\theta_3^{\texttt{H}}$ and $\theta_4^{\texttt{H}}$ respectively to 0.2 and 0.5 because the total HI-critical execution rate will be increased by $(0.075 + 0.3) = 0.375$, which is larger than the slack 0.3027 and the system is not guaranteed to remain correct. To maximize the increase in overall system's QoS while ensuing correctness, at most one LO-critical task can be selected: task τ_4 is selected since it increases the QoS by 0.6 which is larger than that of task τ_3.

Given a correct system, we formulate an ILP to determine which LO-critical tasks are to be selected to maximize the increase in overall system's QoS while ensuring correctness. Let $x_i \in \{0, 1\}$ denote a decision variable whether the LO-critical task τ_i can be guaranteed to provide full service or not. The solution of the ILP determines the values of x_i for each LO-critical task. If $x_i = 1$, then the increase in HI-critical execution rate of the LO-critical task τ_i is $x_i \cdot (u_i^{\texttt{L}} - u_i^{\texttt{H}})$ and the increase in QoS is $x_i \cdot (V_i^{\texttt{L}} - V_i^{\texttt{H}})$.

The purpose of the ILP is to find the value of x_i for each LO-critical task τ_i such that (i) the total *increase* in QoS is maximized (i.e., $\sum_{\tau_i \in \Gamma_L} x_i \cdot (V_i^{\texttt{L}} - V_i^{\texttt{H}})$ is maximized), and (ii) the total *increase* in the HI-critical execution rates for all the LO-critical tasks is not larger than the slack $\mathcal{S}$ to ensure correctness (i.e., $\sum_{\tau_i \in \Gamma_L} x_i \cdot (u_i^{\texttt{L}} - u_i^{\texttt{H}}) \leq \mathcal{S}$). The LO-critical task τ_i for which $x_i = 1$ provides full service in all possible criticality behaviors. The value of the decision variable x_i for each $\tau_i \in \Gamma_L$ is determined using the following ILP:

$$\begin{aligned} &\underset{x_i}{\text{maximize}} && \sum_{\tau_i \in \Gamma_L} x_i \cdot (V_i^{\texttt{L}} - V_i^{\texttt{H}}) \\ &\text{subject to} && \sum_{\tau_i \in \Gamma_L} x_i \cdot (u_i^{\texttt{L}} - u_i^{\texttt{H}}) \leq \mathcal{S} \\ & && \text{and } x_i = 0 \text{ or } x_i = 1 \end{aligned} \tag{13}$$

Given the values of x_i for all the LO-critical tasks in Γ_L, the total increase in QoS of the system is normalized by the number of LO-critical tasks. The normalized QoS, denoted by V_{task}^{norm}, is given in Eq. (14). We set $V_{task}^{norm} = 0$ if $|\Gamma_L| = 0$. Note that $0 \leq V_{task}^{norm} \leq 1$.

$$V_{task}^{norm} = \frac{\sum_{\tau_i \in \Gamma_L} x_i \cdot (V_i^{\texttt{L}} - V_i^{\texttt{H}})}{|\Gamma_L|} \tag{14}$$

6 Empirical Results

This section presents the results on the effectiveness of MCFQ algorithm both in terms of schedulability and improving the system-level QoS using randomly generated implicit-deadline sporadic IMC task sets. The proposed MCFQ algorithm is compared against the MC-Fluid [12] and MCF [3] algorithms. However, the MC-Fluid and MCF algorithms do not consider IMC task models (i.e., all the LO-critical tasks are aborted once the system switches to HI-critical behavior). Baruah et al. [2] extended the MC-Fluid algorithm for uniprocessor considering IMC task model. We extended the MC-Fluid and MCF algorithms for multiprocessors based on a similar approach in [2] for IMC tasks (details of this extension are in Appendix B). Before we present our results, we present the task set generation algorithm.

6.1 Task Set Generation Algorithm

Random implicit-deadline sporadic MC tasksets are generated using an approach similar to those in [12, 3, 13]. Let $U_B = max\{(U_{\mathtt{H}}^{\mathtt{H}} + U_{\mathtt{L}}^{\mathtt{H}})/m, (U_{\mathtt{H}}^{\mathtt{L}} + U_{\mathtt{L}}^{\mathtt{L}})/m\}$ denotes the upper bound on normalized total system utilization in both LO- and HI-critical behaviors. The number of tasks in a randomly generated task set is controlled using an upper bound on the individual task's utilization (u_{max}). The proportion of HI-critical tasks is controlled using probability (p_h). The ratio of HI- and LO-critical utilizations of each task τ_i is controlled using a parameter (R_{max}) such that $1 \leq u_i^{\mathtt{L}} / u_i^{\mathtt{H}} \leq R_{max}$ for a LO-critical task and $1 \leq u_i^{\mathtt{H}} / u_i^{\mathtt{L}} \leq R_{max}$ for a HI-critical task. The following set of values are considered in our experiments for the task set parameters:

- Number of processors: $m \in \{2, 4, 8, 16\}$.
- Normalized utilization bound: $U_B \in \{0.1, 0.15, \ldots 1.0\}$.
- Probability of tasks to be of HI-critical: $p_h \in \{0.0, 0.1, 0.2, \ldots 1.0\}$.
- Maximum individual task utilization: $u_{max} \in \{0.1, 0.2, 0.3, \ldots 1.0\}$.
- Maximum ratio of individual task's utilizations: $R_{max} \in \{1.0, 1.4, 1.8, \ldots 4.0\}$.

We consider 75,240 different combinations of the above parameters to generate the tasksets. For each combination, we generate 1000 task sets where each task set is generated as follows where each parameter is selected from an uniform distribution.

- A real number P_i is drawn from the range $[0, 1]$. If $P_i < p_h$, then $L_i = \mathtt{HI}$; otherwise $L_i = \mathtt{LO}$.
- Task period T_i is drawn from the range $[10, 1000]$.
- Task utilization u_i is drawn from the range $[0.02, u_{max}]$.
- A real number R_i is drawn from the range $[1, R_{max}]$.
- If $L_i = \mathtt{LO}$, then $u_i^{\mathtt{L}} = u_i$ and $u_i^{\mathtt{H}} = (u_i/R_i)$. Otherwise, $u_i^{\mathtt{H}} = u_i$ and $u_i^{\mathtt{L}} = (u_i/R_i)$. The value of $C_i^{\mathtt{L}} = \lceil u_i^{\mathtt{L}} \cdot T_i \rceil$ and $C_i^{\mathtt{H}} = \lceil u_i^{\mathtt{H}} \cdot T_i \rceil$.
- If $L_i = \mathtt{LO}$, then $V_i^{\mathtt{L}} = 1.0$ and $V_i^{\mathtt{H}} = C_i^{\mathtt{H}}/C_i^{\mathtt{L}}$, i.e., the QoS value is set by the system designer based on imprecise computation model [15, 16].
- Repeat the above steps as long as $max\{(U_{\mathtt{H}}^{\mathtt{H}} + U_{\mathtt{L}}^{\mathtt{H}})/m, (U_{\mathtt{H}}^{\mathtt{L}} + U_{\mathtt{L}}^{\mathtt{L}})/m\} \leq U_B$. Once the condition is violated, discard the task that was generated the last.
- If the resulting task set satisfies the condition $max\{(U_{\mathtt{H}}^{\mathtt{H}} + U_{\mathtt{L}}^{\mathtt{H}})/m, (U_{\mathtt{H}}^{\mathtt{L}} + U_{\mathtt{L}}^{\mathtt{L}})/m\} > U_B - 0.05$, then accept the task set and stop the procedure. Otherwise, discard the taskset and the repeat the above steps.

6.2 Results: Schedulability Tests

We compare the effectiveness of the MCFQ algorithm in terms of schedulability of randomly generated task sets with the (extended) MC-Fluid and MCF algorithms applicable to IMC task sets. For a specific scheduling algorithm, and m, U_B, p_h, R_{max} values, let the *acceptance ratio* denote the fraction of task sets out of 1000 task sets that are deemed schedulable by the algorithm.

The HI-critical tasks are considered in increasing order of $u_i^{\mathtt{H}}/\overline{u}_i^{\mathtt{L}}$ when assigning the execution rates based on the MCFQ algorithm. Figure 3 presents the acceptance ratio for each scheduling algorithm against various values of m and U_B with $p_h = 0.5$, $u_{max} = 0.9$ and $R_{max} = 2$. All the algorithms have acceptance ratio 100% when $U_B < 0.70$ and we plot results for $U_B > 0.7$.

We have the following observations. The MCFQ algorithm outperforms the MCF algorithm for task sets with large utilization. The performance of the MCFQ algorithm is very close to

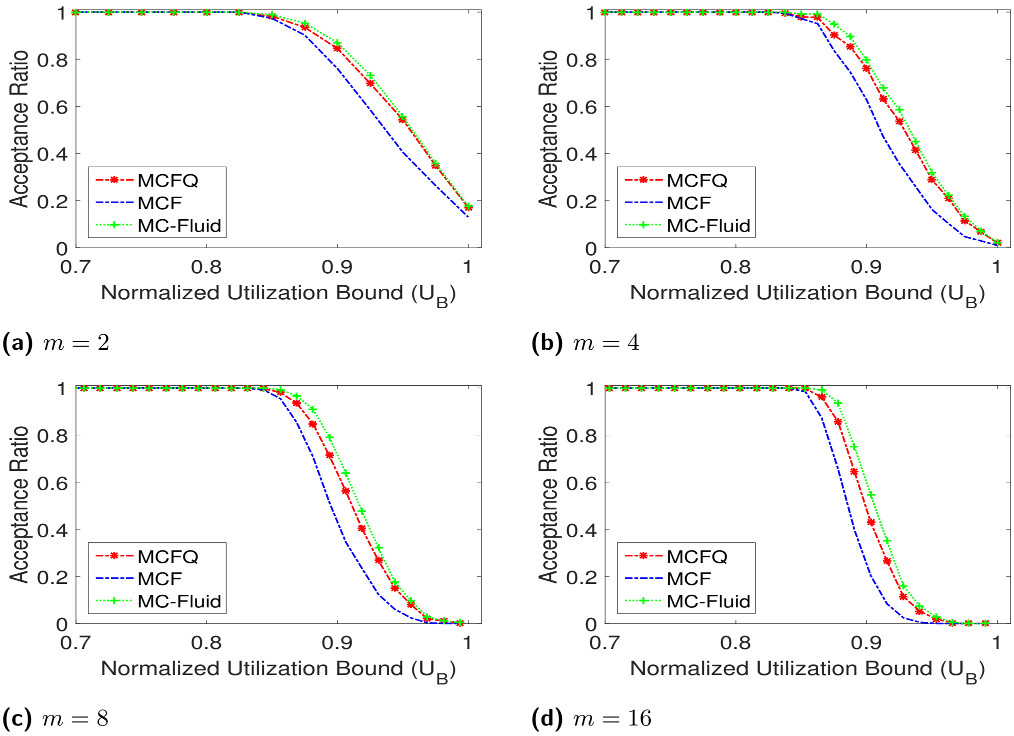


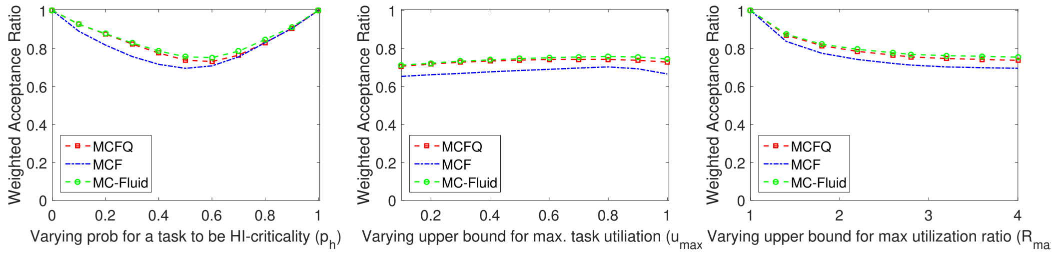
Figure 3 Comparison of acceptance ratios for different number of processors.

the optimal MC-Fluid algorithm. For larger U_B, the acceptance ratio of MCF drops sharply while the performance of MCFQ remains very close to the optimal MC-Fluid algorithm.

For comparison of the acceptance ratios of different algorithms for varying values of p_h, u_{max} and R_{max}, we also computed the *weighted acceptance ratios* in Figure 4. This metric denotes the fraction of schedulable task sets weighted by the normalized utilization bound U_B. If $AR(U_B)$ denotes the acceptance ratio of a scheduling algorithm for normalized utilization bound U_B for some given values of p_h, u_{max}, R_{max} and m, then the weighted acceptance ratio for a set S of U_B values is given as $W(S) = \left(\sum_{U_B \in S}(AR(U_B) \times U_B)\right) / \sum_{U_B \in S} U_B$.

In Figure 4a, we plot the weighted acceptance ratio of the algorithms for different values of p_h for $u_{max} = 0.9$ and $R_{max} = 2$. The performance of the algorithms is better when the value of p_h is either very small or very large since at these extremes the task sets behaves more like non-MC task systems and the effect of switching the criticality behavior has less impact on schedulability. In Figure 4b, we plot the weighted acceptance ratio of the algorithms for different values of u_{max} for $p_h = 0.5$ and $R_{max} = 2$. The performance of the algorithms is independent of the variation in u_{max} (which is also observed in [3] for non-IMC task sets).

In Figure 4c, we plot the weighted acceptance ratio of the algorithms for different values of R_{max} for $p_h = 0.5$ and $u_{max} = 0.9$. The performance of the algorithms decreases with increasing values of R_{max}. When R_{max} increases, the total HI-critical utilization of the HI-critical tasks also increases and the total HI-critical utilization of the LO-critical tasks decreases. As it is already shown in [3] for non-IMC tasks (i.e., LO-critical tasks are dropped at criticality switch), the weighted acceptance ratio decreases with larger R_{max}. For IMC task sets in which the LO-critical tasks execute in HI-critical behavior with degraded service, it is even more difficult to schedule such task sets as R_{max} increases.



(a) Varying p_h **(b)** varying u_{max} **(c)** Varying R_{max}

Figure 4 Comparison of weighted acceptance ratios for different number of processors.

Considering the plots in Figure 4a–4c, it is evident that the performance of `MCFQ` algorithm is much better than the MCF algorithm and is very close to the optimal MC-Fluid algorithm for varying values of u_{max}, p_h, and R_{max} for IMC task sets.

6.3 Results: System's QoS

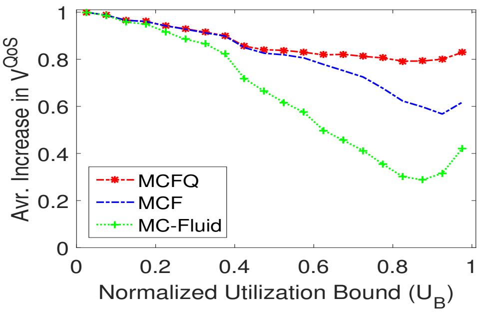
In this section, we present the effectiveness of `MCFQ` algorithm in increasing the overall QoS value of the system (defined as V_{task}^{norm} in Eq. (14)) in comparison to MC-Fluid and MCF algorithms.

If a task set is not schedulable using a particular algorithm, then such a task set is not subjected to QoS evaluation. This is because the system designer's first concern is ensuring `MC`-correctness (i.e., schedulability). If more than one algorithm guarantee the `MC`-correctness of a given task system, then the system designer's second concern is which algorithm maximizes the QoS of the system. Therefore, we consider only those task sets that are schedulable using all the three algorithms for QoS evaluation based on the approach presented in Section 5. For each such randomly-generated task set and each particular algorithm, we determine V_{task}^{norm} by solving the ILP given in Eq. (13) using Matlab's `intlinprog` function. If $\mathcal{K}$ out of 1000 tasksets for a particular configuration are deemed to be schedulable using all three algorithms, then the average V_{task}^{norm}, denoted by V^{QoS}, of these $\mathcal{K}$ task sets is computed for each algorithm.

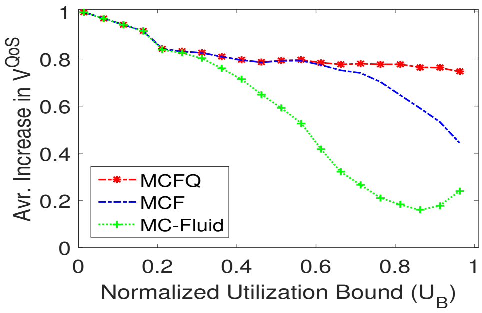
Figure 5 presents the average increase in overall QoS of the system (i.e., value of V^{QoS}) for each scheduling algorithm against various values of m and U_B with $p_h = 0.5$, $u_{max} = 0.9$ and $R_{max} = 2$. The `MCFQ` algorithm outperforms *both* MC-Fluid and MCF. This is because the amount of slack available during the `HI`-critical behavior, based on the execution rates determined by algorithm `MCFQ` in Figure 2, is much larger than that of both MC-Fluid and MCF algorithms. Such slack allows more `LO`-critical tasks to provide full service. When the utilization of the system is very large, the `MCFQ` algorithm provides much higher QoS than both MC-Fluid and MCF.

Figure 6 presents the average number of `LO`-critical tasks (among all the `LO`-critical tasks of all the $\mathcal{K}$ schedulable task sets at each utilization point) that provide full service during the `HI`-critical behavior of the system for various values of m and U_B with $p_h = 0.5$, $u_{max} = 0.9$, and $R_{max} = 2$.

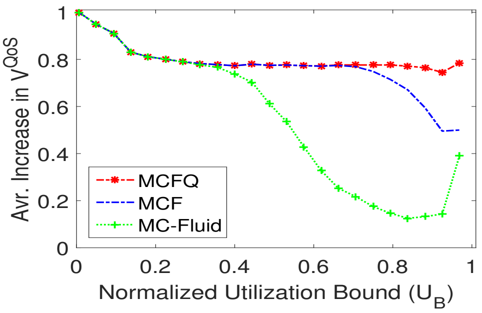
It is evident from Figure 6 that when the utilization of the system is low, then all the algorithms allow almost all the `LO`-critical tasks to provide full service in all behaviors. Earlier approaches do not consider such QoS improvement for systems with $(U_{\mathtt{H}}^{\mathtt{H}} + U_{\mathtt{L}}^{\mathtt{L}}) > m$, i.e., the `LO`-critical tasks are either aborted (non-IMC task model) or only guaranteed to provide degraded (IMC task model) service.



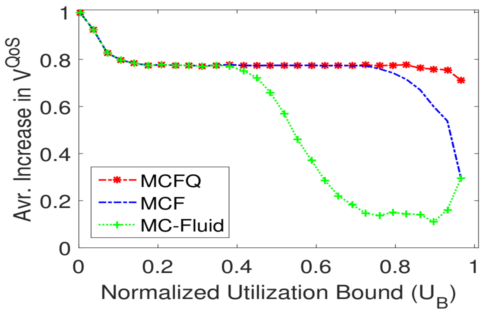
(a) m=2



(b) m=4

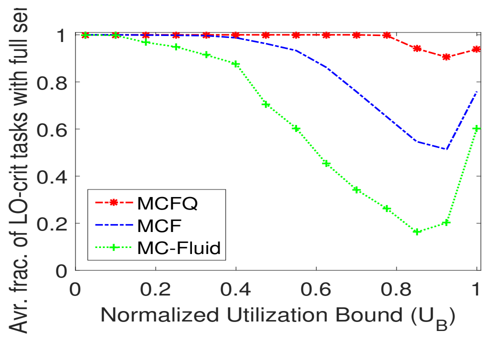


(c) m=8

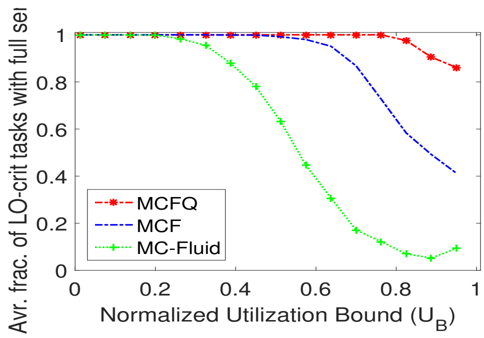


(d) m=16

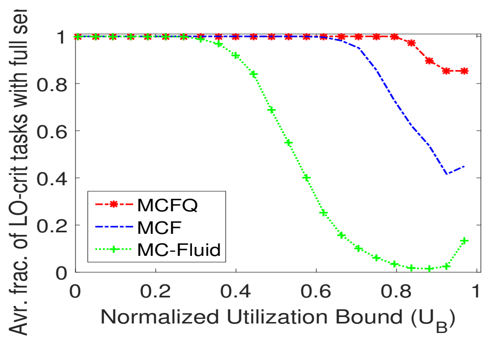
Figure 5 Average increase in QoS of the system (V^{QoS})



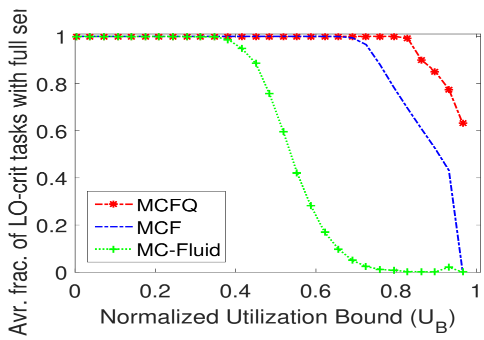
(a) m=2



(b) m=4



(c) m=8



(d) m=16

Figure 6 Average fraction of LO-critical tasks providing full service in both criticality behaviors

The MCFQ algorithm allows almost 100% of *all* the LO-critical tasks to provide full service up to very high utilization ($U_B \approx 0.8$) in comparison to both MC-Fluid and MCF algorithms for different number of processors. The MCFQ algorithm allows much larger fraction of the LO-critical tasks to provide full service in comparison to both MC-Fluid and MCF when the utilization is higher than 0.8.

7 Conclusion

This paper proposes the MCFQ algorithm based on the fluid scheduling model and determines the execution rates for a set of implicit-deadline IMC sporadic tasks considering multiprocessor platform. The recently proposed IMC task model is extended with two QoS values for each LO-critical task. The system designer can assign these QoS values and determine the QoS of the overall system.

The design of the execution rate assignment algorithm of MCFQ ensures that the system is fully utilized during the LO-critical behavior so that the system may have some slack capacity during the HI-critical behavior. The slack, if available, is distributed to the LO-critical tasks such that some of these tasks continue to provide full service after the system switches to HI-critical behavior. The LO-critical tasks that provide full service improve the QoS of the system – making the system designers happier.

References

1. 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 *Proc. of ECRTS*, 2012. doi:10.1109/ECRTS.2012.42.
2. S. Baruah, A. Burns, and Z. Guo. Scheduling mixed-criticality systems to guarantee some service under all non-erroneous behaviors. In *Proc. of ECRTS*, 2016. doi:10.1109/ECRTS.2016.12.
3. S. Baruah, A. Eswaran, and Z. Guo. MC-Fluid: Simplified and Optimally Quantified. In *Proc. of RTSS*, 2015. doi:10.1109/RTSS.2015.38.
4. S. Baruah, Haohan Li, and L. Stougie. Towards the Design of Certifiable Mixed-criticality Systems. In *Proc. of RTAS*, 2010. doi:10.1109/RTAS.2010.10.
5. S. Baruah and S. Vestal. Schedulability Analysis of Sporadic Tasks with Multiple Criticality Specifications. In *Proc. of ECRTS*, 2008. doi:10.1109/ECRTS.2008.26.
6. Sanjoy Baruah, Alan Burns, and Robert Davis. Response-time analysis for mixed criticality systems. In *Proc. of RTSS*, 2011. doi:10.1109/RTSS.2011.12.
7. A. Burns and S. Baruah. Towards a more practical model for mixed criticality systems. In *Proc. of WMC, RTSS*, 2013. http://www-users.cs.york.ac.uk/~robdavis/wmc2013/paper3.pdf.
8. A. Burns and R. Davis. Mixed-criticality systems: A review. In *(available online), Eighth Edition*, July, 2016. http://www-users.cs.york.ac.uk/~burns/review.pdf.
9. Oliver Gettings, Sophie Quinton, and Robert I. Davis. Mixed criticality systems with weakly-hard constraints. In *Proc. of RTNS*, 2015. doi:10.1145/2834848.2834850.
10. Nan Guan, Pontus Ekberg, Martin Stigge, and Wang Yi. Effective and Efficient Scheduling of Certifiable Mixed-Criticality Sporadic Task Systems. In *Proc. of RTSS*, 2011. doi:10.1109/RTSS.2011.10.
11. Mathieu Jan, Lilia Zaourar, and Maurice Pitel. Maximizing the execution rate of low-criticality tasks in mixed criticality systems. In *Proc. of WMC, RTSS*, 2013. http://www-users.cs.york.ac.uk/~robdavis/wmc2013/paper6.pdf.

12 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 *Proc. of RTSS*, 2014. doi:10.1109/RTSS.2014.32.

13 Haohan Li and Sanjoy Baruah. Global mixed-criticality scheduling on multiprocessors. In *Proc of ECRTS*, 2012. doi:10.1109/ECRTS.2012.41.

14 Di Liu, Jelena Spasic, Gang Chen, Nan Guan, Songran Liu, Todor Stefanov, and Wang Yi. EDF-VD Scheduling of Mixed-Criticality Systems with Degraded Quality Guarantees. In *Proc. of RTSS*, 2016. doi:10.1109/RTSS.2016.013.

15 Jane W. S. Liu, Kwei-Jay Lin, Wei-Kuan Shih, Albert Chuang-shi Yu, Jen-Yao Chung, and Wei Zhao. Algorithms for scheduling imprecise computations. *Computer*, 24(5):58–68, May 1991. doi:10.1007/978-1-4615-3956-8_8.

16 J. W. Liu, W.-K. Shih, K.-J. Lin, R. Bettati, and J.-Y. Chung. Imprecise computations. *Proceedings of the IEEE*, 82(1):83–94, 1994. doi:10.1109/5.259428.

17 Risat Mahmud Pathan. Fault-tolerant and real-time scheduling for mixed-criticality systems. *Real-Time Systems*, 50(4):509–547, 2014. doi:10.1007/s11241-014-9202-z.

18 F. Santy, L. George, P. Thierry, and J. Goossens. Relaxing Mixed-Criticality Scheduling Strictness for Task Sets Scheduled with FP. In *Proc. of ECRTS*, 2012. doi:10.1109/ECRTS.2012.39.

19 H. Su, N. Guan, and D. Zhu. Service guarantee exploration for mixed-criticality systems. In *Proc. of RTCSA*, 2014. doi:10.1109/RTCSA.2014.6910499.

20 H. Su and D. Zhu. An elastic mixed-criticality task model and its scheduling algorithm. In *Proc. of DATE*, 2013. doi:10.7873/DATE.2013.043.

21 S. Vestal. Preemptive Scheduling of Multi-criticality Systems with Varying Degrees of Execution Time Assurance. In *Proc. of RTSS*, pages 239–243, 2007. doi:10.1109/RTSS.2007.47.

A Speed Up Bound

The MCFQ algorithm has a speed-up bound of 4/3: if a given dual-criticality implicit-deadline IMC sporadic task system can be scheduled upon a particular multiprocessor platform in an MC-correct manner by any algorithm (including an optimal, clairvoyant, one), then it can be scheduled by MCFQ upon a platform in which each processor is faster by a factor 4/3. It is already shown (in Theorem 5 in [1]) that no non-clairvoyant algorithm for scheduling dual-criticality implicit-deadline non-IMC sporadic task systems can have a speedup factor smaller than 4/3 even on uniprocessor (i.e., for $m = 1$). Therefore, the speed-up bound of 4/3 for MCFQ is optimal since IMC task model is a generalization of non-IMC task model.

To prove the speed-up of 4/3 in Theorem 11, we use Lemma 9 and Lemma 10. Lemma 9 shows that the sum of the LO-critical execution rates that are determined by the MCFQ algorithm in Figure 2 does not exceeds the capacity of the platform.

▶ **Lemma 9.** *The MCFQ algorithm in Figure 2 ensures that* $\sum_{\tau_i \in \Gamma} \theta_i^{\mathtt{L}} \leq m$.

Proof. Each LO-critical task $\tau_i \in \Gamma_L$ is assigned LO-critical execution rate $\theta_i^{\mathtt{L}} = u_i^{\mathtt{L}}$ in Step 1 of the algorithm in Figure 2. Therefore, $\sum_{\tau_i \in \Gamma_L} \theta_i^{\mathtt{L}} = \sum_{\tau_i \in \Gamma_L} u_i^{\mathtt{L}} = U_{\mathtt{L}}^{\mathtt{L}}$. Since $\Gamma = \Gamma_H \cup \Gamma_L$, this lemma is proved by showing that $\sum_{\tau_i \in \Gamma_H} \theta_i^{\mathtt{L}} \leq m - U_{\mathtt{L}}^{\mathtt{L}}$.

Let the tasks in $\Gamma_H = \{\tau_1, \tau_2, \ldots \tau_h\}$ are indexed (in an arbitrary order) such that there are $h = |\Gamma_H|$ tasks in set Γ_H. Since $\theta_i^{\mathtt{L}} = min\{u_i^{\mathtt{H}}, \mathcal{F}_{i-1} \cdot \overline{u}_i^{\mathtt{L}}\}$ in Eq. (5) for $\tau_i \in \Gamma_H$, we have

$$\theta_i^{\mathtt{L}} = min\{u_i^{\mathtt{H}}, \mathcal{F}_{i-1} \cdot \overline{u}_i^{\mathtt{L}}\} \leq u_i^{\mathtt{H}} \tag{15}$$

$$\theta_i^{\mathtt{L}} = min\{u_i^{\mathtt{H}}, \mathcal{F}_{i-1} \cdot \overline{u}_i^{\mathtt{L}}\} \leq \mathcal{F}_{i-1} \cdot \overline{u}_i^{\mathtt{L}} \tag{16}$$

Recall from Eq. (9) that $1 \le \mathcal{F}_0 \le \mathcal{F}_1 \le \ldots \le \mathcal{F}_{h-1}$. We prove this lemma by considering two cases: case (i) $\mathcal{F}_0 = \mathcal{F}_1 = \ldots = \mathcal{F}_{h-1}$, and case (ii) $\mathcal{F}_{k-1} < \mathcal{F}_k$ for some k, $1 \le k \le h-1$.

Case (i): From Eq. (8), we have $\mathcal{F}_0 \cdot \sum_{i=1}^{h} \overline{u}_i^{\mathtt{L}} = \mathcal{F}_0 \cdot \overline{U}_{\mathtt{H}}^{\mathtt{L}} = (m - U_{\mathtt{L}}^{\mathtt{L}})$. From Eq. (16), we have

$$\sum_{\tau_i \in \Gamma_H} \theta_i^{\mathtt{L}} = \sum_{i=1}^{h} \theta_i^{\mathtt{L}} \le \sum_{i=1}^{h} \mathcal{F}_{i-1} \cdot \overline{u}_i^{\mathtt{L}}$$

(For this case $\mathcal{F}_i = \mathcal{F}_0$ for $i = 0, 1 \ldots (h-1)$)

$$\Leftrightarrow \sum_{\tau_i \in \Gamma_H} \theta_i^{\mathtt{L}} = \sum_{i=1}^{h} \theta_i^{\mathtt{L}} \le \sum_{i=1}^{h} \mathcal{F}_{i-1} \cdot \overline{u}_i^{\mathtt{L}} = \sum_{i=1}^{h} \mathcal{F}_0 \cdot \overline{u}_i^{\mathtt{L}} = m - U_{\mathtt{L}}^{\mathtt{L}}$$

Case (ii): Let q is the largest index in the range $[1, 2, \ldots (h-1)]$ such that $\mathcal{F}_{q-1} < \mathcal{F}_q$ where $1 \le q \le h-1$. Such a q must exist for this case. Since q is largest index, we have based on Eq. (9)

$$\mathcal{F}_{q-1} < \mathcal{F}_q = \mathcal{F}_{q+1} = \ldots = \mathcal{F}_{h-1} \tag{17}$$

Since $\mathcal{F}_q = max\{\mathcal{F}_{q-1}, \frac{m - U_{\mathtt{L}}^{\mathtt{L}} - \sum_{i=1}^{q} u_i^{\mathtt{H}}}{\overline{U}_{\mathtt{H}}^{\mathtt{L}} - \sum_{i=1}^{q} \overline{u}_i^{\mathtt{L}}}\}$ according to Eq. (8) and $\mathcal{F}_{q-1} < \mathcal{F}_q$ for this case, we have

$$\mathcal{F}_q = \frac{m - U_{\mathtt{L}}^{\mathtt{L}} - \sum_{i=1}^{q} u_i^{\mathtt{H}}}{\overline{U}_{\mathtt{H}}^{\mathtt{L}} - \sum_{i=1}^{q} \overline{u}_i^{\mathtt{L}}} > \mathcal{F}_{q-1} \tag{18}$$

To prove this lemma we show that

$$\sum_{\tau_i \in \Gamma_H} \theta_i^{\mathtt{L}} = \sum_{i=1}^{h} \theta_i^{\mathtt{L}} = \sum_{i=1}^{q} \theta_i^{\mathtt{L}} + \sum_{i=q+1}^{h} \theta_i^{\mathtt{L}} \le m - U_{\mathtt{L}}^{\mathtt{L}}$$

(From Eq. (15) and Eq. (16))

$$\Leftarrow \sum_{i=1}^{q} u_i^{\mathtt{H}} + \sum_{i=q+1}^{h} \mathcal{F}_{i-1} \cdot \overline{u}_i^{\mathtt{L}} \le m - U_{\mathtt{L}}^{\mathtt{L}}$$

(From Eq. (17), $\mathcal{F}_q = \mathcal{F}_{i-1}$ for $i = q+1, q+2, \ldots h$))

$$\Leftrightarrow \sum_{i=1}^{q} u_i^{\mathtt{H}} + \mathcal{F}_q \cdot \sum_{i=q+1}^{h} \overline{u}_i^{\mathtt{L}} \le m - U_{\mathtt{L}}^{\mathtt{L}}$$

$$\Leftrightarrow \sum_{i=1}^{q} u_i^{\mathtt{H}} + \mathcal{F}_q \cdot (\overline{U}_{\mathtt{H}}^{\mathtt{L}} - \sum_{i=1}^{q} \overline{u}_i^{\mathtt{L}}) \le m - U_{\mathtt{L}}^{\mathtt{L}}$$

(From Eq. (18))

$$\Leftrightarrow \sum_{i=1}^{q} u_i^{\mathtt{H}} + \frac{m - U_{\mathtt{L}}^{\mathtt{L}} - \sum_{i=1}^{q} u_i^{\mathtt{H}}}{(\overline{U}_{\mathtt{H}}^{\mathtt{L}} - \sum_{i=1}^{q} \overline{u}_i^{\mathtt{L}})} \cdot (\overline{U}_{\mathtt{H}}^{\mathtt{L}} - \sum_{i=1}^{q} \overline{u}_i^{\mathtt{L}}) \le m - U_{\mathtt{L}}^{\mathtt{L}}$$

$$\Leftrightarrow m - U_{\mathtt{L}}^{\mathtt{L}} \le m - U_{\mathtt{L}}^{\mathtt{L}}$$

Therefore, the sum of the `LO`-critical execution rates of all the tasks is not larger than m. ◀

▶ **Lemma 10.** *Consider the following function $f(x)$ where*

$$f(x) = x \cdot ((2s-1)m - x)$$

for $0 \le x \le (2s-1) \cdot m$. The maximum value of function $f(x)$ is $\frac{(2sm-m)^2}{4}$.

Proof. The first derivative of $f(x)$ with respect to x is $f'(x) = ((2s-1)m - 2x)$. The derivative is zero for $x = (2s-1)m/2$. Considering the two end-points of the interval $[0, (2s-1)m]$, function $f(x)$ reaches its maximum for some $x \in \{0, (2s-1)m/2, (2s-1)m\}$. We have $f(x) = 0$ when either $x = 0$ or $x = (2s-1)m$. We have $f(x) = \frac{(2sm-m)^2}{4}$ when $x = (2s-1)m/2$. Therefore, the maximum of $f(x)$ within $[0, (2s-1)m]$ is $\frac{(2sm-m)^2}{4}$. ◀

In addition to Lemma 9–10, we need Eq. (19) to show that MCFQ has a speed-up bound 4/3.

$$\theta_i^{\mathtt{H}} = \frac{(u_i^{\mathtt{H}} - u_i^{\mathtt{L}})}{(1 - \frac{u_i^{\mathtt{L}}}{\theta_i^{\mathtt{L}}})} = u_i^{\mathtt{H}} + \frac{u_i^{\mathtt{H}} - u_i^{\mathtt{L}} - u_i^{\mathtt{H}}(1 - \frac{u_i^{\mathtt{L}}}{\theta_i^{\mathtt{L}}})}{1 - \frac{u_i^{\mathtt{L}}}{\theta_i^{\mathtt{L}}}} = u_i^{\mathtt{H}} + u_i^{\mathtt{L}} \cdot \frac{u_i^{\mathtt{H}} - \theta_i^{\mathtt{L}}}{\theta_i^{\mathtt{L}} - u_i^{\mathtt{L}}} \tag{19}$$

▶ **Theorem 11.** *The speedup bound of MCFQ is 4/3.*

Proof. Consider an IMC task set that is feasible using an algorithm (including an optimal, clairvoyant, one) on m speed-s processors where $s \leq 3/4$. We will now show that the MCFQ algorithm in Figure 2 also declares success (i.e., the condition in Eq. (7) is satisfied) for this task set for m speed-1 processors. Since $1/(3/4) = 4/3$, the MCFQ algorithm has speedup bound 4/3.

If a task system is feasible upon m speed-s processors, it is necessary that the following three conditions hold:

$$(U_{\mathtt{H}}^{\mathtt{H}} + U_{\mathtt{L}}^{\mathtt{H}}) \leq m \cdot s \tag{20}$$

$$(U_{\mathtt{L}}^{\mathtt{L}} + \overline{U}_{\mathtt{H}}^{\mathtt{L}}) \leq m \cdot s \tag{21}$$

$$\forall_i : \quad max\{u_i^{\mathtt{L}}, u_i^{\mathtt{H}}\} \leq s \tag{22}$$

We assume that $U_{\mathtt{H}}^{\mathtt{H}} + U_{\mathtt{L}}^{\mathtt{L}} > m$ since a task set with $U_{\mathtt{H}}^{\mathtt{H}} + U_{\mathtt{L}}^{\mathtt{L}} \leq m$ is trivially schedulable by setting $\theta_i^{\mathtt{L}} = \theta_i^{\mathtt{H}} = \max\{u_i^{\mathtt{L}}, u_i^{\mathtt{H}}\}$ for each task $\tau_i \in \Gamma$. From Eq. (20) and Eq. (21), we have

$$\begin{aligned} & 0 \leq (U_{\mathtt{L}}^{\mathtt{L}} + U_{\mathtt{H}}^{\mathtt{H}}) + (U_{\mathtt{L}}^{\mathtt{H}} + \overline{U}_{\mathtt{H}}^{\mathtt{L}}) \leq 2 \cdot m \cdot s \\ \Rightarrow\ & 0 \leq (U_{\mathtt{L}}^{\mathtt{L}} + U_{\mathtt{H}}^{\mathtt{H}}) \leq 2 \cdot m \cdot s - \overline{U}_{\mathtt{H}}^{\mathtt{L}} \qquad (23) \\ & \text{(Since we assume task sets where } U_{\mathtt{L}}^{\mathtt{L}} + U_{\mathtt{H}}^{\mathtt{H}} > m) \\ \Rightarrow\ & 0 \leq \overline{U}_{\mathtt{H}}^{\mathtt{L}} \leq 2 \cdot m \cdot s - m = (2s-1) \cdot m \qquad (24) \end{aligned}$$

Since Eq. (20)–(22) hold, the assumptions for applying the MCFQ algorithm in Figure 2 are true. Therefore, it follows from Lemma 9 that $\sum_{\tau_i \in \Gamma} \theta_i^{\mathtt{L}} \leq m$ for MCFQ algorithm. To prove this theorem, we now show that $\sum_{\tau_i \in \Gamma} \theta_i^{\mathtt{H}} \leq m$, which implies that the condition in Step 3 in Figure 2 will be true..

From Step 1 in Figure 2, we have $\sum_{\tau_i \in \Gamma_L} \theta_i^{\mathtt{H}} = \sum_{\tau_i \in \Gamma_L} u_i^{\mathtt{H}} = U_{\mathtt{L}}^{\mathtt{H}}$. Since $\Gamma = \Gamma_H \cup \Gamma_L$, we only need to show that $\sum_{\tau_i \in \Gamma_H} \theta_i^{\mathtt{H}} \leq m - U_{\mathtt{L}}^{\mathtt{H}}$ to prove this theorem. From Eq. (6), the value $\theta_i^{\mathtt{H}}$ for $\tau_i \in \Gamma_H$ is

$$\theta_i^{\mathtt{H}} = (u_i^{\mathtt{H}} - u_i^{\mathtt{L}})/(1 - \frac{u_i^{\mathtt{L}}}{\theta_i^{\mathtt{L}}}).$$

It is evident from the above equation that $\theta_i^{\mathtt{H}}$ increases as $\theta_i^{\mathtt{L}}$ decreases. Recall from Eq. (5) that the maximum value of $\theta_i^{\mathtt{L}}$ is $u_i^{\mathtt{H}}$ for which the value of $\theta_i^{\mathtt{H}}$ is *minimized.* According to Eq. (10), the minimum value of $\theta_i^{\mathtt{H}}$ is $u_i^{\mathtt{H}}$ given that $\theta_i^{\mathtt{L}}$ is also equal to $u_i^{\mathtt{H}}$.

For any feasible task set, the value of $\theta_i^{\mathtt{H}}$ must be at least equal to $u_i^{\mathtt{H}}$ for each HI-critical task τ_i in order to ensure that the system is correct during stable HI-critical behavior. The value of $\theta_i^{\mathtt{H}}$ is larger than $u_i^{\mathtt{H}}$ when $\theta_i^{\mathtt{L}}$ is smaller than $u_i^{\mathtt{H}}$. Therefore, the sum of the HI-critical execution rates of the HI-critical tasks is maximized when each of the LO-critical execution rate $\theta_i^{\mathtt{L}}$ for $\tau_i \in \Gamma_H$ is smaller than $u_i^{\mathtt{H}}$. According to Eq. (5), the value of $\theta_i^{\mathtt{L}}$ is smaller than $u_i^{\mathtt{H}}$ only if $u_i^{\mathtt{H}} \geq \mathcal{F}_{i-1} \cdot \overline{u}_i^{\mathtt{L}}$. Therefore, the sum of $\theta_i^{\mathtt{H}}$ for all the HI-critical tasks is maximized when $\theta_i^{\mathtt{L}} = \mathcal{F}_{i-1} \cdot \overline{u}_i^{\mathtt{L}} \leq u_i^{\mathtt{H}}$ for all $\tau_i \in \Gamma_H$. To prove this theorem we only need to consider the worst-case where $\theta_i^{\mathtt{L}} = \mathcal{F}_{i-1} \cdot \overline{u}_i^{\mathtt{L}}$ for each task $\tau_i \in \Gamma_H$ because the sum of the HI-critical execution rates of the HI-critical tasks is maximized under this worst-case.

We will show that if $s \leq 4/3$, then the following holds even under the worst-case assumption that $\theta_i^{\mathtt{L}} = \mathcal{F}_{i-1} \cdot \overline{u}_i^{\mathtt{L}} \leq u_i^{\mathtt{H}}$ for all $\tau_i \in \Gamma_H$:

$$\sum_{\tau_i \in \Gamma_H} \theta_i^{\mathtt{H}} \leq m - U_{\mathtt{L}}^{\mathtt{H}}$$

(Since $\theta_i^{\mathtt{H}} = u_i^{\mathtt{H}} + u_i^{\mathtt{L}} \cdot \dfrac{u_i^{\mathtt{H}} - \theta_i^{\mathtt{L}}}{\theta_i^{\mathtt{L}} - u_i^{\mathtt{L}}}$ from Eq. (19))

$$\Leftrightarrow \sum_{\tau_i \in \Gamma_H} \left(u_i^{\mathtt{H}} + u_i^{\mathtt{L}} \cdot \frac{u_i^{\mathtt{H}} - \theta_i^{\mathtt{L}}}{\theta_i^{\mathtt{L}} - u_i^{\mathtt{L}}}\right) \leq m - U_{\mathtt{L}}^{\mathtt{H}}$$

$$\Leftrightarrow U_{\mathtt{H}}^{\mathtt{H}} + \sum_{\tau_i \in \Gamma_H} u_i^{\mathtt{L}} \cdot \frac{u_i^{\mathtt{H}} - \theta_i^{\mathtt{L}}}{\theta_i^{\mathtt{L}} - u_i^{\mathtt{L}}} \leq m - U_{\mathtt{L}}^{\mathtt{H}}$$

(Since $\theta_i^{\mathtt{L}} = \mathcal{F}_{i-1} \cdot \overline{u}_i^{\mathtt{L}}$ under the worst-case and $\mathcal{F}_{i-1} \geq \mathcal{F}_0$ from Eq. (9) and $\overline{u}_i^{\mathtt{L}} \geq u_i^{\mathtt{L}}$ from Eq. (2), we have $\theta_i^{\mathtt{L}} = \mathcal{F}_{i-1} \cdot \overline{u}_i^{\mathtt{L}} \geq \mathcal{F}_0 \cdot \overline{u}_i^{\mathtt{L}} \geq \mathcal{F}_0 \cdot u_i^{\mathtt{L}}$)

$$\Leftarrow U_{\mathtt{H}}^{\mathtt{H}} + \sum_{\tau_i \in \Gamma_H} u_i^{\mathtt{L}} \cdot \frac{u_i^{\mathtt{H}} - \mathcal{F}_0 \cdot \overline{u}_i^{\mathtt{L}}}{\mathcal{F}_0 \cdot u_i^{\mathtt{L}} - u_i^{\mathtt{L}}} \leq m - U_{\mathtt{L}}^{\mathtt{H}}$$

$$\Leftrightarrow \frac{\sum_{\tau_i \in \Gamma_H} u_i^{\mathtt{H}} - \mathcal{F}_0 \cdot \sum_{\tau_i \in \Gamma_H} \overline{u}_i^{\mathtt{L}}}{\mathcal{F}_0 - 1} \leq m - U_{\mathtt{L}}^{\mathtt{H}} - U_{\mathtt{H}}^{\mathtt{H}}$$

$$\Leftrightarrow \frac{U_{\mathtt{H}}^{\mathtt{H}} - \mathcal{F}_0 \cdot \overline{U}_{\mathtt{H}}^{\mathtt{L}}}{\mathcal{F}_0 - 1} \leq m - U_{\mathtt{L}}^{\mathtt{H}} - U_{\mathtt{H}}^{\mathtt{H}}$$

(Since $\mathcal{F}_0 = \dfrac{m - U_{\mathtt{L}}^{\mathtt{L}}}{\overline{U}_{\mathtt{H}}^{\mathtt{L}}}$ from Eq. (8))

$$\Leftrightarrow \frac{\overline{U}_{\mathtt{H}}^{\mathtt{L}} \cdot U_{\mathtt{H}}^{\mathtt{H}} - (m - U_{\mathtt{L}}^{\mathtt{L}}) \cdot \overline{U}_{\mathtt{H}}^{\mathtt{L}}}{m - U_{\mathtt{L}}^{\mathtt{L}} - \overline{U}_{\mathtt{H}}^{\mathtt{L}}} \leq m - U_{\mathtt{L}}^{\mathtt{H}} - U_{\mathtt{H}}^{\mathtt{H}}$$

$$\Leftrightarrow \frac{\overline{U}_{\mathtt{H}}^{\mathtt{L}} \cdot (U_{\mathtt{H}}^{\mathtt{H}} + U_{\mathtt{L}}^{\mathtt{L}} - m)}{m - U_{\mathtt{L}}^{\mathtt{L}} - \overline{U}_{\mathtt{H}}^{\mathtt{L}}} \leq m - U_{\mathtt{L}}^{\mathtt{H}} - U_{\mathtt{H}}^{\mathtt{H}}$$

(Since $ms \geq (U_{\mathtt{H}}^{\mathtt{H}} + U_{\mathtt{L}}^{\mathtt{H}})$ and $ms \geq (U_{\mathtt{L}}^{\mathtt{L}} + \overline{U}_{\mathtt{H}}^{\mathtt{L}})$ from Eq. (20)–(21))

$$\Leftarrow \overline{U}_{\mathtt{H}}^{\mathtt{L}} \cdot (U_{\mathtt{H}}^{\mathtt{H}} + U_{\mathtt{L}}^{\mathtt{L}} - m) \leq (m - ms)^2$$

(Since Eq. (23) holds, we have $(U_{\mathtt{L}}^{\mathtt{L}} + U_{\mathtt{H}}^{\mathtt{H}}) \leq 2 \cdot m \cdot s - \overline{U}_{\mathtt{H}}^{\mathtt{L}}$)

$$\Leftarrow \overline{U}_{\mathtt{H}}^{\mathtt{L}} \cdot (2 \cdot m \cdot s - \overline{U}_{\mathtt{H}}^{\mathtt{L}} - m) \leq (m - ms)^2$$

$$\Leftrightarrow \overline{U}_{\mathtt{H}}^{\mathtt{L}} \cdot ((2s-1)m - \overline{U}_{\mathtt{H}}^{\mathtt{L}}) \leq (m - ms)^2$$

(Since $\overline{U}_{\mathtt{H}}^{\mathtt{L}} \cdot ((2s-1)m - \overline{U}_{\mathtt{H}}^{\mathtt{L}})) \leq ((2s-1)m/2)^2$ from Lemma 10)

$$\Leftarrow ((2s-1)m/2)^2 \leq (m - ms)^2$$

$$\Leftarrow (2s-1)m/2) \leq (m - ms)$$

$$\Leftrightarrow (2s-1)/2) \leq (1 - s)$$

$$\Leftrightarrow (2s-1) \leq (2 - 2s) \quad \Leftrightarrow \quad 4s \leq 3 \quad \Leftrightarrow \quad s \leq 3/4$$

Therefore, if $s \leq 3/4$, the sum of the HI-critical execution rates is not larger than m and MCFQ algorithm in Figure 2 returns success. ◀

B Extension of MC-Fluid and MCF for IMC tasks

The IMC task set Γ is transformed to a non-IMC task set $\overline{\Gamma}$ in which all the original parameters of each of the tasks in Γ remains the same in $\overline{\Gamma}$ except that the LO- and HI-critical execution time $C_i^{\mathtt{L}}$ and $C_i^{\mathtt{H}}$ of each LO-critical task $\tau_i \in \overline{\Gamma}$ is reduced by $C_i^{\mathtt{H}}$. In other words, each LO-critical task in $\overline{\Gamma}$ has $\overline{C_i^{\mathtt{L}}} = C_i^{\mathtt{L}} - C_i^{\mathtt{H}}$ and $\overline{C_i^{\mathtt{H}}} = 0$. We use the MC-Fluid/MCF algorithm to find the execution rates $\overline{\theta}_i^{\mathtt{L}}$ and $\overline{\theta}_i^{\mathtt{H}}$ considering a multiprocessor platform with total capacity $\overline{m} = (m - U_{\mathtt{L}}^{\mathtt{H}})$ using the non-IMC task set $\overline{\Gamma}$.

The execution rates $\theta_i^{\mathtt{L}}$ and $\theta_i^{\mathtt{H}}$ for the original IMC tasks in set Γ are set as follows: if τ_i is a LO-critical task, then $\theta_i^{\mathtt{L}} = \overline{\theta}_i^{\mathtt{L}} + u_i^{\mathtt{H}}$ and $\theta_i^{\mathtt{H}} = \overline{\theta}_i^{\mathtt{H}} + u_i^{\mathtt{H}}$; otherwise, ($\tau_i$ is a HI-critical task) $\theta_i^{\mathtt{L}} = \overline{\theta}_i^{\mathtt{L}}$ and $\theta_i^{\mathtt{H}} = \overline{\theta}_i^{\mathtt{H}}$. It can be proved (based on the same proof technique in [2]) that this extension is correct for scheduling IMC task sets.