MC-Fluid: simplified and optimally quantified

Sanjoy Baruah Arvind Easwaran Zhishan Guo

***Abstract*—The *fluid scheduling* model allows for schedules in which an individual task may be assigned a fraction of a processor at each time instant. These assignments are subject to the constraints that no fraction exceeds one and the sum of all the assigned fractions do not exceed the sum of the computing capacities of all the processors at any instant. An algorithm, MC-Fluid, has recently been proposed for scheduling systems of mixed-criticality implicit-deadline sporadic tasks under the fluid scheduling model. MC-Fluid has been shown to have a speedup bound no worse than $(1+\sqrt{5})/2$ or ≈ 1.618 for scheduling dual-criticality systems.**

We derive here a simplified variant of MC-Fluid called MCF, that has run-time linear in the number of tasks. We prove that this simplified variant has a speedup bound no worse than $4/3$ for dual-criticality systems, and show that this implies that MC-Fluid, too, has a speedup bound no worse than $4/3$. We know from prior results in uniprocessor mixed-criticality scheduling that no algorithm may have a speedup bound smaller than $4/3$, allowing us to conclude that MCF and MC-Fluid are in fact speedup-optimal for dual-criticality scheduling.

I. Introduction[1]

The MC-Fluid scheduling algorithm [4] was designed for scheduling mixed-criticality implicit-deadline sporadic task systems upon identical multiprocessor platforms. Given such a task system, MC-Fluid determines a scheduling strategy under the *fluid scheduling* model. This allows for schedules in which individual tasks may be assigned a fraction ≤ 1 of a processor (rather than an entire processor, or none) at each instant in time, subject to the constraint that the sum of the fractions assigned to all the tasks do not exceed the sum of the computing capacities of all the processors at any instant. MC-Fluid, as described in [4], restricts itself to consideration of dual-criticality systems – there are two criticality levels designated LO and HI. A task τ_i is characterized by the parameters $(\chi_i, C_i^L, C_i^H, T_i)$, where $\chi_i \in \{\text{LO}, \text{HI}\}$ denotes its criticality, C_i^L and C_i^H its LO and HI criticality WCETs, and T_i its period. The objective is to schedule a system τ comprising n such tasks upon m unit-speed processors in a *mixed-criticality correct* (MC-correct) manner, where the notion of MC-correctness is formally specified in Section II (Definition 1). To do so, MC-Fluid seeks to determine *execution rates* θ_i^L and θ_i^H for each task τ_i such that the scheduling algorithm depicted in Figure 1 constitutes an MC-correct scheduling strategy for τ. Such values for the θ_i^L's and θ_i^H's are derived in [4] by solving a convex optimization problem. It is also shown in [4] that this approach has a *speedup factor* no worse than $(1+\sqrt{5})/2$: if a given task system τ can be scheduled in an MC-correct manner by an optimal clairvoyant scheduler upon an m-processor platform, then MC-Fluid will successfully determine an MC-correct scheduling strategy for τ upon an m-processor platform in which each processor is faster by a factor of $(1+\sqrt{5})/2$.

[1]Some familiarity is assumed here on the part of the reader with the mixed-criticality scheduling model introduced by Vestal [7] and reviewed in, e.g. [3]. A brief introduction to this model is provided in the appendix.

- Each τ_i initially executes at a constant rate θ_i^L. That is, at each time-instant it is executing upon θ_i^L fraction of a processor (here, θ_i^L is required to be ≤ 1).
- If a job of any task τ_i does not complete despite having received C_i^L units of execution (equivalently, having executed for a duration (C_i^L/θ_i^L)), then
  - All LO-criticality tasks are immediately discarded, and
  - Each HI-criticality task henceforth executes at a constant rate θ_i^H (θ_i^H, too, must be ≤ 1).

Fig. 1. The run-time scheduling strategy used by Algorithm MC-Fluid

This work. In this paper, we derive an algorithm called MCF for computing the execution rates — i.e., the θ_i^L's and θ_i^H's — that, we believe, is conceptually simpler than the convex optimization technique used in [4]. We show that Algorithm MCF has run-time linear in the number of tasks, and a speedup bound of $4/3$. We also evaluate our algorithm via schedulability experiments, in which we compare it with earlier-proposed algorithms for scheduling mixed-criticality implicit-deadline sporadic task systems upon identical multiprocessor platforms.

Since fluid schedules are not always implementable upon actual computing platforms, another algorithm, named MC-DP-Fair, was also derived in [4] that transforms such a fluid schedule into a schedule in which each task is assigned either zero or one processor at each instant in time. MC-DP-Fair continues to be valid for use in conjunction with Algorithm MCF; hence in the remainder of this paper we will not address the issue of constructing non-fluid schedules any further. Instead, we will assume that the schedule constructed by Algorithm MCF is passed on to MC-DP-Fair to be converted into a non-fluid schedule, just as the schedules constructed by MC-Fluid were in [4].

Organization. The remainder of this paper is organized as follows. We describe the system model, and introduce some terminology and notation, in Section II. We describe Algorithm MCF in Section III, prove its correctness in Section IV, and derive its speedup bound of $4/3$ in Section V. In Section VI we prove that this improved speedup bound of $4/3$ is applicable to MC-Fluid as well, thereby improving upon the bound derived in [4]. We have conducted some

1052-8725/15 $31.00 © 2015 IEEE
DOI 10.1109/RTSS.2015.38

schedulability experiments to compare the performance of MCF with existing multiprocessor mixed-criticality scheduling algorithms; we describe these schedulability experiments in Section VII.

II. System model

A mixed-criticality (MC) implicit-deadline sporadic task system τ consists of a finite specified collection of MC implicit-deadline sporadic tasks, each of which may generate an unbounded number of MC jobs.

MC jobs. As stated in Section I, MC-Fluid [4] restricts itself to consideration of dual-criticality systems and we will, for the most part, do the same. The workload of such a dual-criticality real-time system is assumed to consist of individual jobs, each characterized by a 5-tuple of parameters: $J_i = (a_i, d_i, c_i^L, c_i^H, \chi_i)$, where

- $a_i \in \mathbf{R}^+$ is the release time, and $d_i \in \mathbf{R}^+$ the deadline. We require that $d_i \geq a_i$.
- c_i^L specifies a less conservative estimate, and c_i^H a more conservative estimate, of the worst case execution time (WCET) of job J_i. That is, we assume $c_i^L \leq c_i^H$.
- $\chi_i \in \{\text{LO}, \text{HI}\}$ denotes the criticality of the job.

System *behavior*. The MC job model has the following semantics. Job J_i is released at time a_i, has a deadline at d_i, and needs to execute for some duration γ_i. The value of γ_i is not known beforehand, but only becomes revealed by actually executing the job until it *signals* that it has completed execution. These values of γ_i for a given run of the system defines the kind of *behavior* exhibited by the system during that run. If each J_i signals completion without exceeding c_i^L units of execution, the system is said to have exhibited LO-*criticality behavior*; if any job J_i signals completion after executing for more than c_i^L but no more than c_i^H units of execution, the system is said to have exhibited HI-*criticality behavior*. If any job J_i does not signal completion despite having executed for c_i^H units, the system is said to have exhibited *erroneous behavior*.

MC implicit-deadline sporadic tasks. Analogously to traditional (non-MC) implicit-deadline sporadic tasks, an MC implicit-deadline sporadic task τ_i is characterized by a four-tuple $(\chi_i, C_i^L, C_i^H, T_i)$, with the following interpretation. Task τ_i generates an unbounded sequence of jobs, with successive jobs being released at least T_i time units apart. Each such job has a deadline that is T_i time units after its release. The criticality of each such job is χ_i, and it has LO-criticality and HI-criticality WCET's of C_i^L and C_i^H respectively.

An MC *implicit-deadline sporadic task system* is specified as a finite number of such sporadic tasks.

Correctness criteria. We define an algorithm for scheduling MC task systems to be *correct* if it is able to schedule any system in such a manner that

- During all LO-criticality behaviors of the system, all jobs receive enough execution between their release time and deadline to be able to signal completion; and
- During all HI-criticality behaviors of the system, all HI-criticality jobs receive enough execution between their release time and deadline to be able to signal completion.

This is formally stated in the following definition:

Definition 1 (MC-correct): A scheduling strategy is MC-correct if it ensures that

- During any run of the system in which it exhibits LO-criticality behavior (i.e., each job of each task completes upon executing for no more than the task's LO-criticality WCET), all jobs complete by their deadlines; and
- During any run of the system in which it exhibits HI-criticality behavior (i.e., each job of each task completes upon executing for no more than the task's HI-criticality WCET), all jobs of all the HI-criticality tasks complete by their deadlines (while jobs of LO-criticality tasks may fail to do so).

■

We now describe some notation that we will be using throughout the reminder of this document. We will let τ denote a collection of n dual-criticality implicit-deadline sporadic tasks that are to be scheduled upon m unit-speed processors. As a general rule, τ with a subscript (as in τ_i) denotes an individual task in τ; however, $\tau_H \subseteq \tau$ ($\tau_L \subseteq \tau$, respectively) denotes all the HI-criticality tasks (all the LO-criticality tasks, resp.) in τ.

The superscripted notation X^H and X^L denote HI-criticality and LO-criticality variants of the quantity X. Hence

- $u_i^L \stackrel{\text{def}}{=} (C_i^L/T_i)$ and $u_i^H \stackrel{\text{def}}{=} (C_i^H/T_i)$ denote the LO-criticality and HI-criticality *utilizations* of task τ_i.
- MC-Fluid seeks to assign values to the *execution-rate* variables $\{\theta_i^L\}_{\tau_i \in \tau} \bigcup \{\theta_i^H\}_{\tau_i \in \tau_H}$. All these variables are to be assigned values in the range $[0, 1]$.
- Various system utilization parameters are defined for τ as follows:

$$U_L^L \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau_L} u_i^L$$
$$U_H^L \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau_H} u_i^L$$
$$U_H^H \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau_H} u_i^H$$

  Hence the LO-criticality total system utilization of task-system τ is $\left(U_L^L + U_H^L\right)$, and its HI-criticality total system utilization is U_H^H.

III. Algorithm MCF

In this section we describe Algorithm MCF, our simplified version of MC-Fluid [4]. Given a dual-criticality implicit-deadline sporadic task system τ to be scheduled upon an m processor platform, MCF, like MC-Fluid, seeks to assign values to the θ_i^L and θ_i^H execution-rate variables such that the run-time algorithm depicted in Figure 1 constitutes an MC-correct scheduling strategy for τ. The manner in which

1) Define ρ as follows:

$$\rho \leftarrow \max\Big\{\Big(\frac{U_L^L + U_H^L}{m}\Big), \Big(\frac{U_H^H}{m}\Big), \max_{\tau_i \in \tau_H}\big\{u_i^H\big\}\Big\} \quad (1)$$

2) **If** $\rho > 1$ **then** declare failure; **else** assign values to the execution-rate variables as follows:

$$\theta_i^H \leftarrow u_i^H/\rho \text{ for all } \tau_i \in \tau_H \quad (2)$$

$$\theta_i^L \leftarrow \begin{cases} \frac{u_i^L \theta_i^H}{\theta_i^H - \left(u_i^H - u_i^L\right)}, & \text{if } \tau_i \in \tau_H \\ u_i^L, & \text{else (i.e., if } \tau_i \in \tau_L) \end{cases} \quad (3)$$

3) **If**

$$\sum_{\tau_i \in \tau} \theta_i^L \le m \quad (4)$$

then declare success **else** declare failure

Fig. 2. Algorithm MCF

Algorithm MCF computes these θ_i^L, θ_i^H values is depicted in Figure 2; the steps are explained below.

Observe that $(U_L^L + U_H^L)$ denotes the total system utilization in LO-criticality behaviors, and U_H^H the total system utilization in HI-criticality behaviors. Hence for τ to be feasible on a platform of m unit-speed processors, it is necessary that $\big(U_L^L + U_H^L\big) \le m$, $U_H^H \le m$ and $u_i^H \le 1$ for each $\tau_i \in \tau_H$. The value assigned to ρ (Expression 1) should therefore be ≤ 1 for any feasible system. Informally speaking, the quantity $(1-\rho)$ can be thought of as representing the "slack" or excess capacity in the system; we seek to exploit this slack by setting the execution rates (the θ_i^L's and θ_i^H's) to be greater than the utilizations (the u_i's).

If ρ is indeed ≤ 1, then the execution rates at HI-criticality (the θ_i^H's) for the HI-criticality tasks are set equal to their HI-criticality utilizations u_i^H scaled by a factor $1/\rho$ (Expression 2). The execution rates at LO-criticality (the θ_i^L's) for each LO-criticality task is set equal to the task utilization (u_i^L), while the θ_i^L for each HI-criticality task is set according to the formula given in Expression 3. The correctness of these assignments will be formally proved in Section IV below.

Finally, the assignment of execution rates is declared a success if the θ_i^L values that are assigned sum to no more than the cumulative computing capacity of the platform.

A. An example

We now illustrate the manner in which Algorithm MCF computes the θ_i^L and θ_i^H parameters via a simple example. An example dual-criticality implicit-deadline sporadic task system that is to be scheduled upon a 2-processor platform was considered in [4]; this task system is reproduced in Table I. For this task system,

$$\begin{aligned} \rho &= \max\Big\{\frac{.3+.4+.1+.5}{2}, \frac{.8+.7+.1}{2}, \max\{.8,.7,.1\}\Big\} \\ &= \max\big\{1.3/2, 1.6/2, .8\big\} \\ &= 0.8 \end{aligned}$$

Therefore tasks τ_1, τ_2 and τ_3, get θ_i^H values assigned as follows:

$$\begin{aligned} \theta_1^H &= \frac{0.8}{0.8} = 1.0 \\ \theta_2^H &= \frac{0.7}{0.8} = 0.875 \\ \text{and } \theta_3^H &= \frac{0.1}{0.8} = 0.125 \end{aligned}$$

The assigned θ_i^L values are as follows:

$$\begin{aligned} \theta_1^L &= \frac{1.0 \times 0.3}{1.0 - (0.8 - 0.3)} = 0.6 \\ \theta_2^L &= \frac{0.875 \times 0.4}{0.875 - (0.7 - 0.4)} = \frac{14}{23} < 0.61 \\ \theta_3^L &= \frac{0.125 \times 0.1}{0.125 - (0.1 - 0.1)} = 0.1 \\ \text{and } \theta_4^L &= 0.5 \end{aligned}$$

Since

$$\sum_{i=1}^{4} \theta_i^L < \big(0.6 + 0.61 + 0.1 + 0.5\big) = 1.81,$$

we conclude that the task system is indeed schedulable by Algorithm MCF.

B. Run-time complexity

It should be evident that Algorithm MCF, as listed in Figure 2, has run-time that is *linear* in the number of tasks in τ. In one straightforward implementation strategy, the scaling factor ρ can be computed in one pass through the task system; the θ_i^H and θ_i^L values in a second pass; and the test to ensure that the θ_i^L values sum to no more than m in a third pass.

	T_i	C_i^L	C_i^H	χ_i	u_i^L	u_i^H
τ_1	10	3	8	HI	0.3	0.8
τ_2	20	8	14	HI	0.4	0.7
τ_3	30	3	3	HI	0.1	0.1
τ_4	40	20	20	LO	0.5	0.5

TABLE I
EXAMPLE TASK SYSTEM

IV. Algorithm MCF: Proof of correctness

In Section III above, we presented Algorithm MCF, a linear-time algorithm for computing execution rates at LO and HI criticalities for a given dual-criticality implicit-deadline sporadic task system that is to be scheduled upon an m-processor platform under the fluid scheduling paradigm. In this section, we prove that this algorithm is correct: if Algorithm MCF computes the execution rates without declaring failure for a given task system τ, then the schedule resulting from using these execution rates in the manner described in Figure 1 does indeed constitute an MC-correct scheduling strategy. Our proof proceeds in several steps.

1) First, we show that each execution rate is assigned a valid value in Figure 2: a non-negative real number no greater than one.
2) Next, we separately prove that the sum of the LO-criticality and the HI-criticality execution rates assigned in Figure 2 to all the tasks do not exceed the capacity of the platform.
3) Third, we show that the θ_i^L and θ_i^H values assigned in Figure 2 are no smaller than the corresponding u_i^L's and u_i^H's — this ensures the correctness of the "steady-state" behavior of the system at either criticality level.
4) Finally, we examine the system in the event that some job does not signal completion despite having executed for up to its LO-criticality WCET (which indicates that the system is exhibiting a HI-criticality behavior rather than a LO-criticality one); we show that the system behavior is correct during the transition as well.

Putting the pieces together, we conclude (Theorem 1) that Algorithm MCF is indeed correct.

§**1: Assigned execution rates are all $\leq$ 1.** Observe that $\rho \geq u_i^H$ for all $\tau_i \in \tau$. It follows that $\theta_i^H \stackrel{\text{def}}{=} (u_i^H/\rho)$ is always ≤ 1, as required. With regards to the θ_i^L's, the value assigned to θ_i^L for each $\tau_i \in \tau_L$ is equal to u_i^L (and hence ≤ 1). We show, in Lemma 1 below, that $\theta_i^L \leq \theta_i^H$ for each $\tau_i \in \tau_H$ (i.e., the execution rate guaranteed to each HI-criticality task does not *de*crease upon identification of HI-criticality behavior). It follows thereby that the θ_i^L variables are also assigned values ≤ 1.

Lemma 1: For each $\tau_i \in \tau_H$

$$\theta_i^H \geq \theta_i^L \tag{5}$$

Proof: By Equation 3, θ_i^L for each $\tau_i \in \tau_H$ is assigned a value $\frac{u_i^L \theta_i^H}{\theta_i^H - \left(u_i^H - u_i^L\right)}$. This is $\leq \theta_i^H$ if

$$\begin{aligned} & \frac{u_i^L}{\theta_i^H - \left(u_i^H - u_i^L\right)} \leq 1 \\ \Leftrightarrow \quad & u_i^L \leq \theta_i^H - \left(u_i^H - u_i^L\right) \\ \Leftrightarrow \quad & u_i^H \leq \theta_i^H \end{aligned}$$

which follows from the requirement that ρ be ≤ 1 (else, we would have declared failure). ■

§**2: Capacity constraints are met.** Condition 4 ensures that the assignment of values to the θ_i^L variables does not exceed the capacity of the m-processor platform; Lemma 2 below shows that neither does the assignment of values to the θ_i^H variables.

Lemma 2:

$$\sum_{\tau_i \in \tau_H} \theta_i^H \leq m \tag{6}$$

Proof: It follows from Equation 1 that

$$\begin{aligned} & \rho \geq \frac{U_H^H}{m} \\ \Leftrightarrow \quad & \frac{U_H^H}{\rho} \leq m \end{aligned} \tag{7}$$

We use this inequality to conclude that

$$\left(\sum_{\tau_i \in \tau_H} \theta_i^H\right) = \left(\sum_{\tau_i \in \tau_H} \frac{u_i^H}{\rho}\right) = \left(\frac{1}{\rho} \sum_{\tau_i \in \tau_H} u_i^H\right) = \left(\frac{U_H^H}{\rho}\right) \leq m$$

and Condition 6 is shown to hold. ■

§**3: Assigned execution rates are (eventually) adequate.** Since $\rho \leq 1$, it must be the case that

$$\theta_i^H \stackrel{\text{def}}{=} \left(\frac{u_i^H}{\rho}\right) \geq u_i^H \tag{8}$$

That is, the execution rate assigned to each task in "steady state" following a transition to HI-criticality behaviors is adequate. Lemma 3 below asserts that the execution rate assigned to each task in LO-criticality behaviors is also adequate.

Lemma 3: For each $\tau_i \in \tau$

$$\theta_i^L \geq u_i^L \tag{9}$$

Proof: This is clearly true for each $\tau_i \in \tau_L$, since $\theta_i^L = u_i^L$ for all such τ_i. To see that it is also true for each $\tau_i \in \tau_H$, observe that for each such τ_i,

$$\begin{aligned} \theta_i^L \quad &= \quad u_i^L \times \frac{\theta_i^H}{\theta_i^H - \left(u_i^H - u_i^L\right)} \\ &\geq \quad u_i^L \quad \text{(Since } (u_i^H - u_i^L) \geq 0) \end{aligned}$$

■

§**4: Correct transition.** Finally, we show that the θ-values computed by Algorithm MCF ensure MC-correctness in HI-criticality behaviors, by analyzing the point in time during run-time at which it is detected that some job has executed beyond its LO-criticality WCET.

Lemma 4: Let t_o denote the first time-instant at which some job does not signal completion despite having executed for its LO-criticality WCET. Any HI-criticality job that is active (i.e., that has been released but has not completed execution) at time-instant t_o receives an amount of execution no smaller than its HI-criticality WCET prior to its deadline.

Proof: Suppose that a job of HI-critiality task τ_i is active at time-instant t_o. Let us suppose that it had arrived at time-instant $(t_o - w)$, were w is a positive number $\leq T_i$; its deadline is then at time-instant $(t_o - w + T_i)$. Over the interval $[t_o -$

$w, t_o)$, this job will have received an amount of execution equal to $\theta_i^L \times w$; since the job is still active, it must be the case that

$$\begin{aligned} & \theta_i^L \times w \leq C_i^L \\ \Leftrightarrow \quad & w \leq \frac{C_i^L}{\theta_i^L} \end{aligned} \tag{10}$$

From the instant t_o to its deadline — i.e., over the interval $[t_o, t_o - w + T_i)$, of duration $(T_i - w)$ — the job of τ_i will execute at a rate θ_i^H. Hence for this job to meet its deadline, it is sufficient that

$$\begin{aligned} & w\theta_i^L + (T_i - w)\theta_i^H \geq C_i^H \\ \Leftrightarrow \quad & T_i\theta_i^H - w(\theta_i^H - \theta_i^L) \geq C_i^H \\ \Leftarrow \quad & T_i\theta_i^H - \frac{C_i^L}{\theta_i^L}(\theta_i^H - \theta_i^L) \geq C_i^H \text{ (By Inequality 10)} \\ \Leftrightarrow \quad & \theta_i^H - \frac{u_i^L}{\theta_i^L}(\theta_i^H - \theta_i^L) \geq u_i^H \\ \Leftrightarrow \quad & \theta_i^H - \frac{u_i^L\theta_i^H}{\theta_i^L} + u_i^L \geq u_i^H \\ \Leftrightarrow \quad & \theta_i^H \geq (u_i^H - u_i^L) + \frac{u_i^L\theta_i^H}{\theta_i^L} \\ \Leftrightarrow \quad & 1 \geq \frac{u_i^H - u_i^L}{\theta_i^H} + \frac{u_i^L}{\theta_i^L} \end{aligned} \tag{11}$$

By Equation 3, for each $\tau_i \in \tau_H$ we have

$$\begin{aligned} & \theta_i^L = \frac{u_i^L\theta_i^H}{\theta_i^H - \left(u_i^H - u_i^L\right)} \\ \Leftrightarrow \quad & \frac{\theta_i^H - \left(u_i^H - u_i^L\right)}{\theta_i^H} = \frac{u_i^L}{\theta_i^L} \\ \Leftrightarrow \quad & 1 - \left(\frac{\left(u_i^H - u_i^L\right)}{\theta_i^H}\right) = \frac{u_i^L}{\theta_i^L} \\ \Leftrightarrow \quad & \frac{u_i^L}{\theta_i^L} + \frac{u_i^H - u_i^L}{\theta_i^H} = 1 \end{aligned}$$

thereby establishing Condition 11 and completing the proof of the lemma. ■

We are now ready to prove the correctness of Algorithm MCF.

Theorem 1: Values assigned to the θ_i^H and θ_i^L variables according to Equations 2-3 that satisfy Condition 4 constitute an MC-correct fluid scheduling strategy.

Proof: Lemma 3 and Condition 4 together suffice to establish correctness in any LO-criticality behavior. Similarly, Lemma 1, in conjunction with Inequality 8 above establishes correctness in "steady state" following transition to HI-criticality behavior. And finally, Lemma 4 establishes that MC-correctness is also preserved upon a transition from LO-criticality to HI-criticality behavior. ■

V. Algorithm MCF: A Speedup Bound

It has previously been shown [4] that Algorithm MC-Fluid has a speedup bound no worse than $(1 + \sqrt{5})/2$; i.e., if a given dual-criticality implicit-deadline 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 MC-Fluid upon a platform in which each processor is faster by a factor $(1+\sqrt{5})/2$ or approximately 1.618. We will prove here a better (i.e., lower) speedup bound of $4/3$ for Algorithm MCF.

Our approach towards developing this speedup bound of $4/3$ is as follows. Observe that for a task system to be feasible upon m speed-s processors, it is necessary that $(U_L^L + U_H^L)$ and U_H^H for the system both be $\leq m \times s$, and that in addition $u_i^H \leq s$ for each task. It therefore follows that the scaling factor ρ that is computed by Algorithm MCF (Expression 1 in Figure 2) for such a system is $\leq s$. We will show below, in Lemma 6, that if $\rho \leq 3/4$ and the θ_i^H, θ_i^L values are computed as specified in Expressions 2–3 of Figure 2, then the θ_i^L's so computed are guaranteed to sum to $\leq m$ and therefore satisfy Condition 4 of Figure 2 (which in turn means that the system is correctly scheduled by MCF on m unit-speed processors). The speedup bound follows, by observing that $4/3$ is the multiplicative inverse of $3/4$.

First, a technical lemma.

Lemma 5: Let c denote any positive constant. The function

$$f(x) \stackrel{\text{def}}{=} \frac{x(c-x)}{\frac{c}{3} + x}$$

is $\leq \frac{c}{3}$ for all values of $x \in [0, c]$.

Proof: This lemma is easily proved rigorously using standard techniques from the calculus: taking the derivative of $f(x)$ with respect to x, we see that the only value of $x \in [0, c]$ where this derivative equals zero is $x \leftarrow c/3$. We therefore conclude that $f(x)$ takes on its maximum value over $[0, c]$ for some $x \in \{0, c/3, c\}$. Explicit computation of $f(x)$ at each of these values reveals that the value is maximized at $x = c/3$, where it takes on the value $c/3$. (We skip the details of the derivation here; for a visual depiction of $f(x)$, it is plotted as a function of x in Figure 3.) ■

Lemma 6: If $\rho \leq 3/4$ and θ_i^H, θ_i^L values are as computed by Algorithm MCF (in the manner specified in Expressions 2–3 of Figure 2), then the θ_i^L values so computed satisfy Condition 4.

Proof: Let us first rewrite Condition 4 to an equivalent form expressed in Condition 12 below.

$$\begin{aligned} & \sum_{\tau_i \in \tau} \theta_i^L \leq m \\ \Leftrightarrow \quad & \sum_{\tau_i \in \tau_L} \theta_i^L + \sum_{\tau_i \in \tau_H} \theta_i^L \leq m \\ \Leftrightarrow \quad & U_L^L + \sum_{\tau_i \in \tau_H} \frac{u_i^L\theta_i^H}{\theta_i^H - \left(u_i^H - u_i^L\right)} \leq m \end{aligned}$$

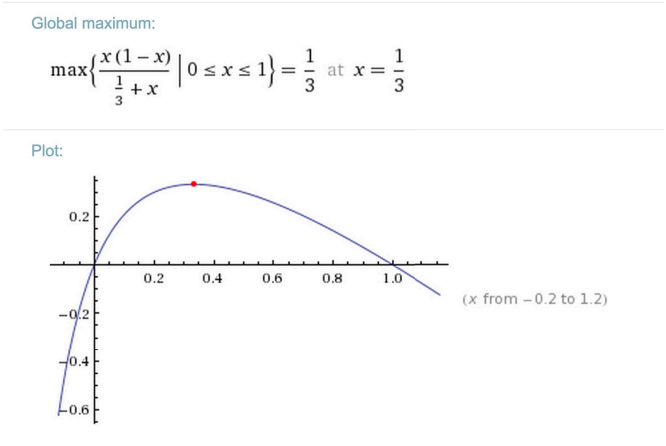


Fig. 3. Plot of $f(x)$ for $c = 1$ (made with the WolframAlpha ® computational knowledge engine: https://www.wolframalpha.com/)

$$\Leftrightarrow \quad U_L^L + \sum_{\tau_i \in \tau_H} u_i^L \Big(1 + \frac{u_i^H - u_i^L}{\theta_i^H - \big(u_i^H - u_i^L\big)}\Big) \le m$$

$$\Leftrightarrow \quad U_L^L + \sum_{\tau_i \in \tau_H} u_i^L + \sum_{\tau_i \in \tau_H} \frac{u_i^H - u_i^L}{\theta_i^H - \big(u_i^H - u_i^L\big)} \le m$$

$$\Leftrightarrow \quad U_L^L + U_H^L + \sum_{\tau_i \in \tau_H} \frac{u_i^L \big(u_i^H - u_i^L\big)}{\theta_i^H - \big(u_i^H - u_i^L\big)} \le m \qquad (12)$$

We will show, in the remainder of this proof, that if $\rho \le 3/4$ then Condition 12 is satisfied; this will serve to establish the correctness of Lemma 6.

Let us assume henceforth that $\rho \le 3/4$. From the definition of ρ (Expression 1), it follows that

$$U_L^L + U_H^L \le \frac{3}{4} m \qquad (13)$$

$$U_H^H \le \frac{3}{4} m \qquad (14)$$

$$\forall \tau_i \in \tau_H \quad u_i^H \le \frac{3}{4} \qquad (15)$$

Additionally, since $\theta_i^H \leftarrow u_i^H / \rho$, it must hold that

$$\forall \tau_i \in \tau_H \quad \theta_i^H \ge \frac{4}{3} u_i^H \qquad (16)$$

Let us use Inequalities 13–16 to further simplify Condition 12.

$$U_L^L + U_H^L + \sum_{\tau_i \in \tau_H} \frac{u_i^L \big(u_i^H - u_i^L\big)}{\theta_i^H - \big(u_i^H - u_i^L\big)} \le m$$

$$\Leftarrow \quad \frac{3}{4} m + \sum_{\tau_i \in \tau_H} \frac{u_i^L \big(u_i^H - u_i^L\big)}{\theta_i^H - \big(u_i^H - u_i^L\big)} \le m \text{ (By Ineq. 13)}$$

$$\Leftarrow \quad \frac{3}{4} m + \sum_{\tau_i \in \tau_H} \frac{u_i^L \big(u_i^H - u_i^L\big)}{\frac{4}{3} u_i^H - \big(u_i^H - u_i^L\big)} \le m \text{ (By Ineq. 16)}$$

$$\Leftrightarrow \quad \frac{3}{4} m + \sum_{\tau_i \in \tau_H} \frac{u_i^L \big(u_i^H - u_i^L\big)}{\frac{u_i^H}{3} + u_i^L} \le m$$

$$\Leftrightarrow \quad \sum_{\tau_i \in \tau_H} \frac{u_i^L \big(u_i^H - u_i^L\big)}{\frac{u_i^H}{3} + u_i^L} \le \frac{m}{4}$$

$$\Leftarrow \quad \Big(\sum_{\tau_i \in \tau_H} \frac{u_i^H}{3} \le \frac{m}{4}\Big) \text{ (By Lemma 5)}$$

$$\Leftrightarrow \quad \Big(\frac{1}{3} U_H^H \le \frac{m}{4}\Big)$$

$$\Leftarrow \quad \Big(\frac{1}{3} \times \frac{3}{4} m \le \frac{m}{4}\Big) \text{ (By Inequality 14)}$$

$$\Leftrightarrow \quad \Big(\frac{m}{4} \le \frac{m}{4}\Big)$$

and Lemma 6 is thereby proved. ■

It has previously been shown [1, Theorem 5] that no non-clairvoyant algorithm for scheduling dual-criticality implicit-deadline sporadic task systems can have a speedup factor smaller than $4/3$ even on *uni*processors (i.e., for $m = 1$); below, we reproduce the example from [1] that bears witness to this fact.

Consider the example task system $\tau = \{\tau_1, \tau_2\}$, with the following parameters, where ϵ is an arbitrary small positive number.

τ_i	χ_i	C_i^L	C_i^H	T_i
τ_1	HI	$1+\epsilon$	$1+\epsilon$	2
τ_2	LO	$1+\epsilon$	3	4

This system is successfully scheduled by a clairvoyant scheduler upon a single processor ($m = 1$): EDF would meet all deadlines in LO-criticality behaviors (since $U_L^L + U_H^L \le 1$), while only jobs of τ_2 would get to execute in HI-criticality behaviors (and $U_H^H \le 1$).

To see that τ cannot be scheduled correctly on a unit-speed processor by any online scheduler, suppose both tasks generate jobs simultaneously. It need not be revealed prior to one of the jobs receiving $(1 + \epsilon)$ units of execution, whether the behavior is going to be a LO-criticality or a HI-criticality one. We consider two cases.

1) τ_1's job receives $(1 + \epsilon)$ units of execution before τ_2's job does. In this case, the behavior is revealed to be a HI-criticality one. But now there is not enough time remaining for τ_2's job to complete by its deadline at time instant 4.
2) τ_2's job receives $(1+\epsilon)$ units of execution before τ_1's job does. In this case, the behavior is revealed to be a LO-criticality one in that τ_2's job signals that it has completed execution. But then, there is not enough time remaining for τ_1's job to complete by its deadline at time 2.

We have thus shown that no non-clairvoyant algorithm can schedule τ in an MC-correct manner. Based on the observation that

$$\max\Big\{U_L^L + U_L^H, U_H^H, \max_{\tau_i \in \tau_H} \big\{u_i^H\big\}\Big\}$$

exceeds $3/4$ by an arbitrarily small amount, we conclude that no non-clairvoyant algorithm for scheduling dual-criticality implicit-deadline sporadic task systems can have a speedup factor smaller than $4/3$.

The speedup optimality of Algorithm MCF for dual-criticality implicit-deadline task systems follows by combining the analysis in the above example with Lemma 6. A speedup bound of $4/3$ holds for Algorithm MCF and, by the argument above, a smaller speedup is not possible for *any* non-clairvoyant algorithm. This is formally expressed in the following theorem:

Theorem 2: Algorithm MCF is speedup-optimal for scheduling dual-criticality implicit-deadline task systems: it has a speedup factor of $4/3$, and no non-clairvoyant algorithm may have a speedup factor lower than $4/3$.

VI. MC-FLUID: AN IMPROVED SPEEDUP BOUND

It was shown in [4] that Algorithm MC-Fluid has a speedup bound no worse than $(1+\sqrt{5})/2$ (≈ 1.618) for dual-criticality implicit-deadline sporadic task systems. We will now improve this result, and show that MC-Fluid, like MCF, has a speedup bound no worse than $4/3$; this, in conjunction with the example above showing that no on-line algorithm may have a speedup bound smaller than $4/3$, serves to establish the speedup-optimality of MC-Fluid.

As we had stated in the introduction, MC-Fluid computes the θ_i^L and θ_i^H execution rates for the tasks, and then uses the run-time dispatcher listed in Figure 1 to schedule the system during run-time. MC-Fluid first computes the θ_i^H values for all $\tau_i \in \tau_H$ by solving a convex optimization problem, and uses these θ_i^H values to assigns values to the θ_i^L variables exactly as Algorithm MCF does (i.e., according to Equation 3, which is reproduced below):

$$\theta_i^L \leftarrow \begin{cases} \frac{u_i^L \theta_i^H}{\theta_i^H - \left(u_i^H - u_i^L\right)}, & \text{if } \tau_i \in \tau_H \\ u_i^L, & \text{else (i.e., if } \tau_i \in \tau_L) \end{cases}$$

It is shown [4, Theorem 2] that the convex optimization problem solved by MC-Fluid essentially computes θ_i^H values to satisfy the following inequalities:

$$\forall i : \tau_i \in \tau_H : \; u_i^H \leq \theta_i^H \quad (17)$$

$$U_L^L + U_H^L + \sum_{\tau_i \in \tau_H} \frac{u_i^L \left(u_i^H - u_i^L\right)}{\theta_i^H - \left(u_i^H - u_i^L\right)} \leq m \quad (18)$$

$$\sum_{\tau_i \in \tau_H} \theta_i^H \leq m \quad (19)$$

Lemma 7 below establishes a strong relationship between Algorithm MCF and these inequalities: that for any task system that is successfully scheduled by Algorithm MCF, there is a solution to these inequalities (and the system is therefore also successfully scheduled by Algorithm MC-Fluid).

Lemma 7: If Algorithm MCF (Figure 2) computes θ_i^H and θ_i^L values for a given dual-criticality implicit-deadline sporadic task system τ without declaring failure, then the θ_i^H values so computed satisfy Inequalities 17–19 (and τ is therefore successfully scheduled by MC-Fluid as well).

Proof: Let us suppose that Algorithm MCF (Figure 2) computes θ_i^H and θ_i^L values for a given dual-criticality implicit-deadline sporadic task system τ without declaring failure. Since ρ, as computed by Expression 1 of Figure 2, must be ≤ 1, it follows that $\theta_i^H \stackrel{\text{def}}{=} u_i^H/\rho$ is $\leq u_i^H$ and Inequality 17 is satisfied for all $\tau_i \in \tau_H$.

Since $\rho \geq U_H^H/m$, it follows that

$$\begin{aligned} \rho &\geq \frac{U_H^H}{m} \\ \Leftrightarrow \quad \rho &\geq \frac{\sum_{\tau_i \in \tau_H} u_i^H}{m} \\ \Leftrightarrow \quad m &\geq \frac{\sum_{\tau_i \in \tau_H} u_i^H}{\rho} \\ \Leftrightarrow \quad m &\geq \sum_{\tau_i \in \tau_H} \frac{u_i^H}{\rho} \\ \Leftrightarrow \quad m &\geq \sum_{\tau_i \in \tau_H} \theta_i^H \end{aligned}$$

and Inequality 19 is also satisfied.

It remains to show that Inequality 18 is satisfied as well. Observe that

$$\begin{aligned} & U_L^L + U_H^L + \sum_{\tau_i \in \tau_H} \frac{u_i^L \left(u_i^H - u_i^L\right)}{\theta_i^H - \left(u_i^H - u_i^L\right)} \\ &= U_L^L + \sum_{\tau_i \in \tau_H} \left(u_i^L + \frac{u_i^L \left(u_i^H - u_i^L\right)}{\theta_i^H - \left(u_i^H - u_i^L\right)} \right) \\ &= U_L^L + \sum_{\tau_i \in \tau_H} \left(\frac{u_i^L \theta_i^H - u_i^L \left(u_i^H - u_i^L\right) + u_i^L \left(u_i^H - u_i^L\right)}{\theta_i^H - \left(u_i^H - u_i^L\right)} \right) \\ &= U_L^L + \sum_{\tau_i \in \tau_H} \left(\frac{u_i^L \theta_i^H}{\theta_i^H - \left(u_i^H - u_i^L\right)} \right) \\ &= U_L^L + \sum_{\tau_i \in \tau_H} \theta_i^L \\ &= \sum_{\tau_i \in \tau_L} \theta_i^L + \sum_{\tau_i \in \tau_H} \theta_i^L \\ &= \sum_{\tau_i \in \tau} \theta_i^L \end{aligned}$$

which is indeed $\leq m$, according to Inequality 4. ■

Thus, Lemma 7 above shows that any task system that is successfully scheduled by Algorithm MCF is also successfuly scheduled by MC-Fluid. Earlier (Section V), we had seen that Algorithm MCF successfully schedules any system upon m unit-speed processors, that is successfully scheduled upon m speed-$3/4$ processors by an optimal clairvoyant scheduler. This fact, in conjunction with the earlier result that no on-line algorithm may have a smaller speedup factor, immediately yields:

Theorem 3: The speedup factor of MC-Fluid is $4/3$.

VII. EXPERIMENTS

In this section we use experiments to evaluate the schedulability of Algorithm MCF, and compare it with existing multiprocessor algorithms for mixed-criticality systems. (We thank Saravanan Ramanathan and Jaewoo Lee for sharing access to their simulation framework.) These algorithms include MC-Fluid [4], global fpEDF [5], global fixed-priority [6]

and partitioned EDF [2], which we respectively denote as *MC-Fluid, GLO-EDF, GLO-FP and PART-EDF*. In terms of the speed-up factor, Section V shows that both MCF and MC-Fluid have an optimal speed-up of $4/3$. This speed-up is significantly better than the best known speed-up bounds for GLO-EDF $(1 + \sqrt{5})$ and PART-EDF $(8/3)$ [2]. In terms of schedulability, the experiments presented below show that MCF clearly outperforms GLO-EDF, GLO-FP and PART-EDF under all the scenarios that were considered. It has been shown in [4] that MC-Fluid is an optimal execution-rate assignment algorithm; i.e., if a set of θ_i^L's and θ_i^H's exist for a specified dual-criticality implicit-deadline sporadic task system that constitutes an MC-correct fluid scheduling strategy, then MC-Fluid is guaranteed to find at least one such assignment. Algorithm MCF, on the other hand, provides no such guarantee, although it is speedup-optimal. That is, there exist task systems for which MC-Fluid can find execution-rates that ensure MC-correctness, whereas MCF fails to do so. MCF in fact trades this optimality in rate assignment for simplicity; it uses a strategy with run-time that is linear in the number of tasks, whereas MC-Fluid uses a convex optimization framework with run-time that is quadratic in the number of tasks. The schedulability experiments below show that the drop in schedulability for Algorithm MCF when compared to MC-Fluid is relatively small when compared to the other algorithms.

A. Taskset generation procedure

We generate random tasksets using a procedure similar to the one adopted in previous studies (e.g., [5], [4]). Let $U_B = \max\{(U_L^L + U_H^L)/m, U_H^H/m\}$ denote the upper bound for normalized total system utilization in both LO- and HI-criticality behaviors. Note that $U_B \leq 1$ is a necessary condition for the feasibility of any dual-criticality implicit-deadline sporadic taskset on an m-processor platform. Besides U_B and m, the other important parameters in generating a dual-criticality taskset include the total number of tasks and the proportion of HI-criticality tasks. We control the former using a bound on the maximum individual task utilization (u_{max}), while for the latter we use a probability measure (P_H). The following sets of values are considered in our experiments for these taskset parameters.

- Number of processors: $m \in \{2, 4, 8, 16\}$.
- Normalized utilization bound: $U_B \in \{0.1, 0.15, 0.2, \ldots, 1.0\}$.
- Probability for a task to be HI-criticality: $P_H \in \{0.0, 0.2, \ldots, 1.0\}$.
- Maximum individual task utilization: $u_{max} \in \{0.1, 0.2, \ldots, 1.0\}$.

Thus, in all we consider $8,360$ different combinations of the above taskset parameters. For each combination of values, we generate $10,000$ different tasksets and evaluate their schedulability. Each taskset is generated using the procedure described in Figure 4, wherein each parameter is drawn at random from an uniform distribution. This procedure ensures that the normalized system utilization of the generated taskset is between $U_B - 0.05$ and U_B. This is reasonable because in our experiments we consider values of U_B that are incremented in steps of 0.05.

[2]There is no known speed-up bound for the GLO-FP algorithm.

1) Task period T_i is an integer drawn from the range $[20, 300]$.
2) The ratio $R_i = u_i^H/u_i^L$ is a real number drawn from the range $[1, 4]$.
3) A real number P_i is drawn from the range $[0, 1]$. If $P_i < P_H$, then $\chi_i =$ HI. Otherwise, $\chi_i =$ LO.
4) Task utilization u_i is drawn from the range $[0.02, u_{max}]$. If $\chi_i =$ LO, then $u_i^L = u_i$. Otherwise, $u_i^H = u_i$ and $u_i^L = u_i^H/R_i$. In both the cases, C_i^L is set to $\lceil u_i^L \times T_i \rceil$ and C_i^H is set to $\lceil u_i^H \times T_i \rceil$.
5) Repeat the above steps as long as $\max\{(U_L^L + U_H^L)/m, U_H^H/m\} \leq U_B$. Once this condition is violated, discard the task that was generated last.
6) If the resulting taskset satisfies the condition $\max\{(U_L^L + U_H^L)/m, U_H^H/m\} > U_B - 0.05$, then accept the taskset and exit the procedure. Otherwise, discard the taskset and repeat the above steps.

Fig. 4. Procedure for generating a single taskset

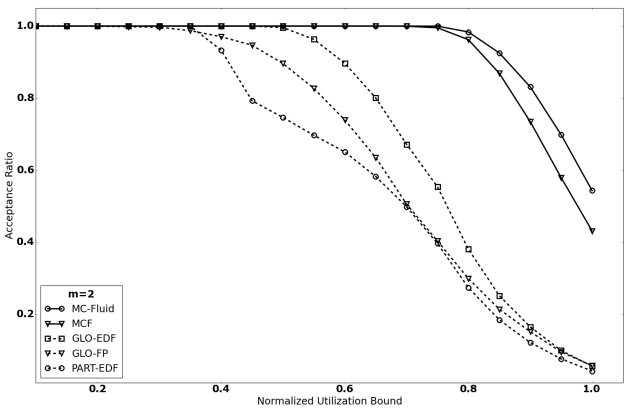
For each taskset, we evaluate its schedulability under the five different scheduling algorithms mentioned above. We regard the taskset as schedulable under MCF (likewise MC-Fluid) if an execution-rate assignment that ensures MC-correctness is found by the respective algorithms. For the other algorithms, we determine taskset schedulability using tests presented in the respective cited references.

B. Results

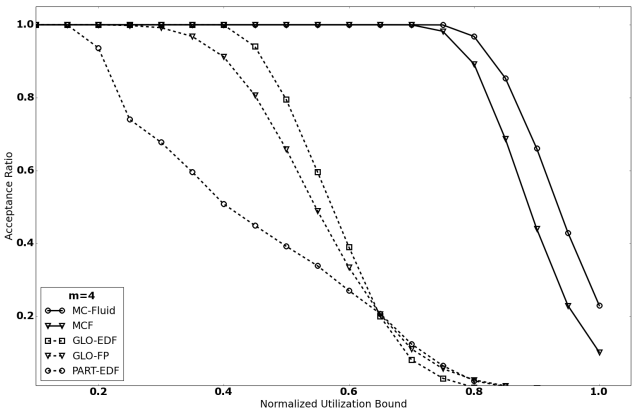
In this section we present two sets of results from the experiments. For a specific scheduling algorithm and m, U_B, P_H and u_{max} values, let *acceptance ratio* denote the fraction out of $10,000$ generated tasksets that are deemed to be schedulable by the algorithm. This metric provides a comparison across the different algorithms when the taskset parameters are fixed. In Figure 5 we plot the acceptance ratio for each scheduling algorithm against varying values of m and U_B, with P_H and u_{max} fixed at 0.5 and 0.9 respectively. To evaluate the algorithms for different values of P_H and u_{max}, we also plot the *weighted acceptance ratios* in Figure 6. This metric denotes the fraction of schedulable tasksets weighted by the normalized utilization bound U_B. That is, for a given value of P_H and u_{max}, if $AR(U_B)$ denotes the acceptance ratio of a scheduling algorithm for normalized utilization bound U_B, then the weighted acceptance ratio for a set S of U_B values is given as

$$W(S) = \frac{\sum_{U_B \in S}(AR(U_B) \times U_B)}{\sum_{U_B \in S} U_B}. \quad (20)$$

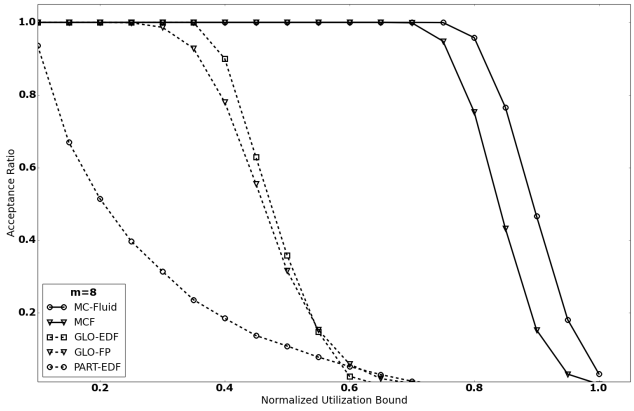
Note that in computing $AR(U_B)$ tasksets scheduled on



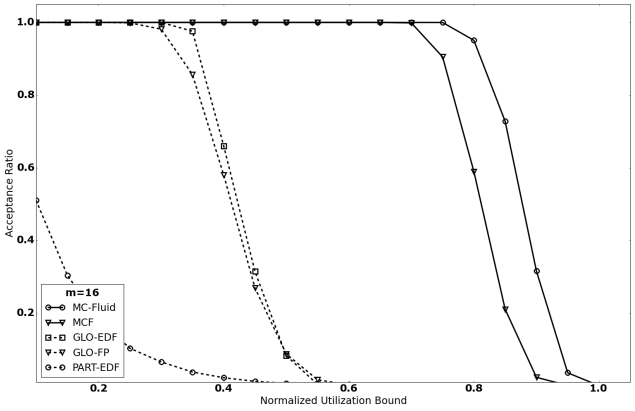
(a) $m = 2$



(b) $m = 4$

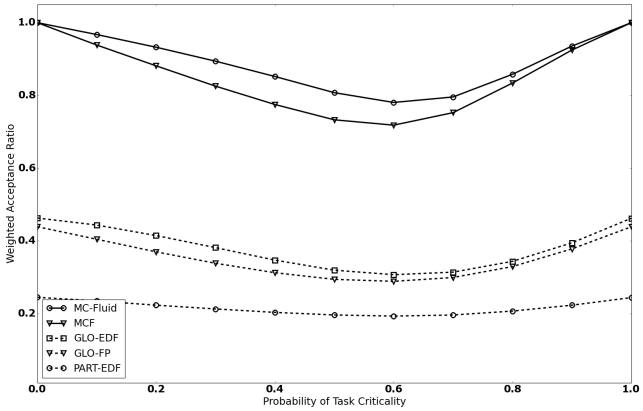


(c) $m = 8$

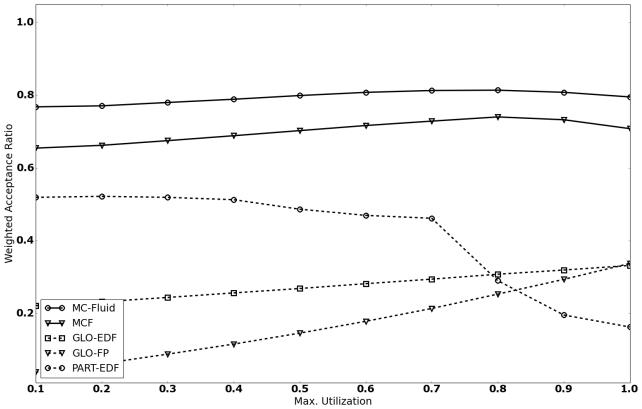


(d) $m = 16$

Fig. 5. Comparison of acceptance ratios for different number of processors



(a) Varying probability for a task to be HI-criticality (P_H)



(b) Varying upper bound for individual task utilization (u_{max})

Fig. 6. Comparison of weighted acceptance ratios

different number of processors can be considered, and further $W(S)$ gives more importance to acceptance ratios of tasksets that are harder to schedule, i.e., tasksets with larger values of U_B. Hence it is an effective metric to combine the impact of parameters m and U_B on acceptance ratios.

As shown in Figure 5, MCF outperforms GLO-EDF, GLO-FP and PART-EDF by a considerable margin, for all the taskset parameter combinations considered in these experiments. This performance gap continues to widen for increasing m. Also, both MCF and MC-Fluid have almost 100% schedulability when $U_B \leq 0.75$ for all the cases. This is expected because both of them have an optimal speed-up bound of $4/3(= 1/0.75)$. For larger values of U_B, the performance of MCF drops below that of MC-Fluid, and this gap widens with increasing m and U_B. This is also expected because, as discussed above, MC-Fluid is an optimal rate assignment strategy, whereas MCF trades this optimality for a simpler and more time-efficient strategy.

In Figure 6(a) we compare the weighted acceptance ratio of the algorithms for varying values of probability P_H, with u_{max} fixed at 0.9. The performance of all the algorithms is better at the extremes when P_H is either small or large. This is reasonable because at these extremes the generated tasksets are comprised of either only LO-criticality or only HI-criticality tasks. Further, it can be seen that both MCF and MC-Fluid continue to outperform all the other algorithms irrespective of the value of P_H.

In Figure 6(b) we compare the weighted acceptance ratio of the algorithms for varying values of maximum utilization u_{max}, with P_H fixed at 0.5. The performance of MCF and MC-Fluid are more or less independent of u_{max}, that of PART-EDF decreases significantly for large values of u_{max}, while that of GLO-EDF and GLO-FP increases gradually with increasing u_{max}. PART-EDF is expected to perform poorly for tasksets with large individual task utilization, because partitioning is harder for such tasksets. On the other hand, the increase in performance of GLO-EDF and GLO-FP is mainly a side-effect of the reduced pessimism in schedulability tests due to fewer number of tasks. This reduced number of tasks also explains why the gap in the performance between MCF and MC-Fluid marginally decreases with increasing u_{max}. MCF, which assigns rates using the same proportional value of ρ for all the HI-criticality tasks, is more sensitive to the number of tasks in the taskset than MC-Fluid that assigns independent rates.

The performance of MCF and MC-Fluid primarily depends on U_B (as demonstrated in Figure 5), and the ratio of HI- to LO-criticality utilization R_i ($\stackrel{\text{def}}{=} u_i^H/u_i^L$). The larger this ratio, the more challenging it is to find a rate assignment that can ensure MC-correctness, especially during a behavior switch. To confirm this dependency we conducted experiments that measure the impact from varying range values for R_i. We considered the range values $\{[1, 1.5], [1, 2], [1, 2.5], [1, 3], [1, 3.5], [1, 4]\}$, with P_H and u_{max} fixed at 0.5 and 0.9 respectively. Figure 7 presents a plot of the weighted acceptance ratios for varying range values

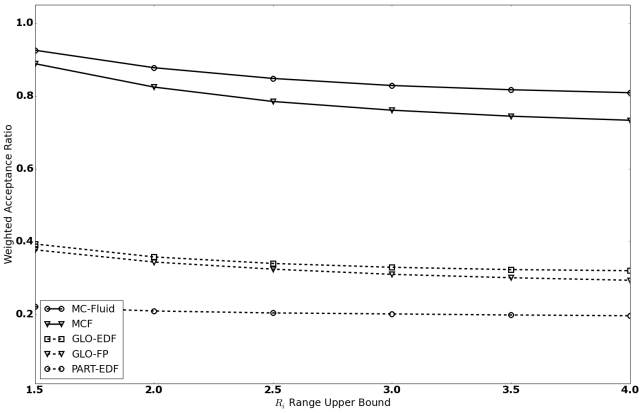


Fig. 7. Comparison of weighted acceptance ratios for varying u_i^H/u_i^L ranges

(since the lower bound for the ranges is always 1, the x-axis in Figure 7 denotes only the upper bound for the ranges). The performance of both MC-Fluid and MCF decreases with increasing range upper bound, thus confirming our hypothesis.

VIII. Summary

We have derived and proved the correctness of Algorithm MCF, an algorithm for the fluid scheduling of dual-criticality implicit-deadline sporadic task systems upon identical multiprocessor platforms. We have shown that Algorithm MCF may be implemented very efficiently to have a run-time that is linear in the number of tasks in the system, and that it has a (tight) speedup bound equal to $4/3$. Our speedup bound result improves upon the prior state of the art (a speedup bound of $(1 + \sqrt{5})/2$, or ≈ 1.618, for the Algorithm MC-Fluid in [4]); in addition, we show that the $4/3$ bound applies to Algorithm MC-Fluid as well.

Our speedup bound results establish that Algorithm MCF is the "best" algorithm to use for constructing fluid schedules for dual-criticality implicit-deadline sporadic task systems upon multiprocessor platforms, for systems satisfying the property that the ρ parameter (defined as in Equation 1) is $\leq 3/4$. Additionally, our schedulability experiments indicate that while Algorithm MC-Fluid is slightly superior to Algorithm MCF for task systems with $\rho > 3/4$ (in the sense that there are such task systems schedulable by MC-Fluid but not by MCF), the difference is indeed very slight. Based upon the superior run-time computational complexity of MCF as well as its intuitive simplicity, we therefore recommend the following strategy for the multiprocessor scheduling of dual-criticality implicit-deadline sporadic task systems:

1) First, apply Algorithm MCF to construct a schedule.
2) If Algorithm MCF fails but $\rho \leq 1$, then apply Algorithm MC-Fluid.

As mentioned in the introduction, the outputs of both MCF and MC-Fluid are execution rates for a fluid schedule; the process of converting the resulting fluid schedule into a non-fluid one may be accomplished by Algorithm MC-DP-Fair [4].

ACKNOWLEDGEMENTS

This research was supported in part by NSF grants CNS 1115284, CNS 1218693, CNS 1409175, and CPS 1446631, AFOSR grant FA9550-14-1-0161, ARO grant W911NF-14-1-0499, and a grant from General Motors Corp.

REFERENCES

[1] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, S. van der Ster, and L. Stougie. The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems. In *Proceedings of the 2012 24th Euromicro Conference on Real-Time Systems*, ECRTS '12, Pisa (Italy), 2012. IEEE Computer Society.

[2] S. Baruah, B. Chattopadhyay, H. Li, and I. Shin. Mixed-criticality scheduling on multiprocessors. *Real-Time Systems*, 50(1):142–177, 2014.

[3] A. Burns and R. Davis. Mixed-criticality systems: A review. Available at `http://www-users.cs.york.ac.uk/~burns/review.pdf`, 2013.

[4] 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 *Real-Time Systems Symposium (RTSS), 2014 IEEE*, pages 41–52, Dec 2014.

[5] H. Li and S. Baruah. Global mixed-criticality scheduling on multiprocessors. In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 166–175, July 2012.

[6] R. M. Pathan. Schedulability analysis of mixed-criticality systems on multiprocessors. In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 309–320, July 2012.

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

APPENDIX

THE VESTAL MODEL: MOTIVATION

In mixed criticality (MC) systems, functionalities of different degrees of importance (or *criticalities*) are implemented upon a common platform. It may be the case that more critical functionalities are required to have their correctness validated to a higher level of assurance than less critical functionalities. For example, consider the worst-case execution (WCET) parameters of pieces of code; these parameters are widely used prior to run-time for validating timing properties of a system. For validating the timing correctness of critical functionalities it is desirable to use WCET parameters that are obtained using extremely conservative tools (for example, ones based on static code-analysis), while less critical functionalities are often validated using (less conservative) measurement-based WCET tools. Based on this fact, Vestal [7, page 239] observed that "the more confidence one needs in a task execution time bound [...] the larger and more conservative that bound tends to become." He proposed that each piece of code therefore be characterized by *multiple* WCET parameters, which are obtained by analyzing the (same) piece of code using different WCET-analysis methodologies. The system is then subject to multiple independent analyses, each using a different set of WCET estimates and seeking to validate different correctness properties. We illustrate the essence of Vestal's idea via a simple (contrived) example.

Example 1: Consider a real-time workload comprising two jobs J_1 and J_2 that is to be implemented on a single preemptive processor, with job J_1 being more critical than J_2. Both jobs have a release time at time 0; J_1's deadline is at time 5 and J_2's at time 10. Let us suppose that the WCET of J_1, determined by a more conservative WCET tool, is equal to 5, while the WCET of J_2, determined using a less conservative WCET tool, is equal to 6. Since the sum of these WCETs exceeds the duration between the jobs' arrival and the latest deadline, conventional scheduling techniques cannot schedule both jobs to guarantee completion by their deadlines. However, Vestal observed in [7] that

- with regards to validating the more critical functionality only (e.g., from the perspective of a certification process), it may be *irrelevant* whether the less critical job J_2 completes on time or not; and
- the value of 5 that is assigned to J_1's WCET parameter may be deemed *too pessimistic* for validating less critical functionalities.

Let us suppose that the WCET of J_1 is estimated once again, this time using the less pessimistic WCET-determination tool; J_1's WCET is determined by this tool to be equal to 3 (rather than 5). If we were now to schedule the jobs by assigning J_1 greater priority than J_2,

- In validating the more critical functionalities, we would determine that J_1 completes by time-instant 5 and hence meets its deadline.
- The validation process responsible for less critical functionalities determines that J_1 completes by time-instant 3, and J_2 by time-instant 9. Thus they both complete by their deadlines.

We thus see that the system is deemed as being correct in both analyses, despite our initial observation that the sum of the relevant WCETs (5 for J_1; 6 for J_2) exceeds the duration between the jobs' common release time and the latest deadline. ■

Example 1 above encapsulates the idea behind Vestal's mixed-criticality model, that

- certain aspects of a system's behavior cannot be known precisely prior to run-time, and must therefore be *estimated* for the purposes of system analysis prior to run-time;
- for mixed-criticality systems, it may make sense to construct multiple models of the entire system, these different models being made under more or less conservative assumptions (so that we can have greater or lesser levels of assurance that the models do indeed bound the actual run-time behavior of the system); and
- it suffices to validate the correctness of the entire system under the less [least] conservative model, but of only the more critical parts of the system under the more conservative model[s].

This idea — that the same system, represented using more and less conservative models, may be demonstrated to satisfy different correctness criteria for different functionalities — has been widely explored since first proposed by Vestal [7]; see, e.g., [3] for a review.