MC-Fluid: rate assignment strategies

Saravanan Ramanathan, Arvind Easwaran
Nanyang Technological University, Singapore
Email: {saravana016, arvinde}@ntu.edu.sg

***Abstract*—In this paper, we consider fluid scheduling of mixed-criticality implicit-deadline sporadic task systems. Fluid scheduling allows tasks to be allocated fractional processing capacity, which, although hard to implement, significantly improves the schedulability performance. For dual-criticality systems, dual-rate fluid scheduling in which each task is assigned two execution rates depending on the system criticality level has been proposed in the past. An optimal rate assignment algorithm for such systems called MC-Fluid that assigns rates with polynomial complexity has also been proposed. Another rate assignment strategy called MCF has recently been proposed with linear run-time complexity. Although MCF results in a lower schedulability ratio when compared to MC-Fluid, it is shown that both the strategies result in a speed-up optimal scheduling algorithm for dual-criticality systems. We propose two new algorithms to assign execution rates called MC-Sort and MC-Slope, both with linearithmic (i.e., *nlogn*) complexity in the number of tasks n. The proposed algorithms have a schedulability ratio that is significantly better than MCF and almost as good as MC-Fluid, but with a reduced run-time complexity when compared to MC-Fluid.**

I. Introduction

Increasing trend in the embedded industry towards platform integration has motivated Mixed-Criticality (MC) systems in the research community. These systems integrate multiple components with varying criticality onto a common hardware. Safety-critical cyber-physical systems such as automotive and avionics fall under this category. Recently, the complexity of such systems has increased owing to increased functionality. Multi-cores have therefore become a natural choice to meet the growing demand of such systems.

In this paper, we consider the problem of multi-core MC scheduling of implicit-deadline sporadic task systems. We consider a type of global scheduling called fluid scheduling in which each task is assigned a fraction of a processing core at each time instant. Assignment criteria are subjected to two constraints: 1. No task is allowed to have an assignment greater than 1 and 2. Sum of assignments of all the tasks should not exceed the total processing capacity of the system. Though fluid scheduling provides better schedulability, it is practically infeasible to implement as heavy overhead is incurred due to frequent context switching. Levin et al. [1] proposed a method to convert fluid schedules into non-fluid schedules without any loss in performance and thereby, making fluid scheduling practically feasible.

Lee et al. [2] proposed a dual-rate fluid algorithm MC-Fluid for scheduling dual-criticality (LO and HI) implicit-deadline sporadic task systems on a multi-core platform. In a dual-rate fluid scheduling, tasks are assigned two execution rates, one in each of LO and HI modes, based on their execution requirements in the two modes. These rates are criticality-dependent as tasks have different execution requirement for different criticality levels, and the criticality level of system changes at run-time. MC-Fluid determines the values of these execution rates by solving a convex optimization problem in polynomial time, and is shown to have an optimal rate assignment strategy [2]. Baruah et al. [3] derived a simplified fluid scheduling algorithm called MCF with an optimal speed-up factor of 4/3 and linear time rate assignment strategy. Although MCF compromises on the schedulability when compared to MC-Fluid, it has lower time complexity for determining the rates and is speed-up optimal.

Extending MC-Fluid to multi-rate model or to multi-criticality systems without compromising on the complexity is quite hard. Formulating and solving an optimization problem for such system can be challenging. In case of MCF, extension to such systems is rather simple, but it will significantly affect the schedulability.

Contributions: We propose two fluid algorithms: MC-Sort and MC-Slope for computing the execution rate of each task at different criticality levels. MC-Sort algorithm sorts all the high-critical tasks based on their high-critical execution requirement, and assigns a larger rate for a task with larger execution requirement. The challenge with the MC-Sort algorithm is that it does not consider the difference in execution requirement of tasks between different criticality levels. It is necessary to allocate higher rate to such tasks since they need to execute more in high-criticality. MC-Slope algorithm, on the other hand, assigns a larger rate to a task with a larger rate of change in the execution requirement between different criticality levels.

To evaluate the performance of our algorithms, experiments are conducted with randomly generated tasks sets. Experiment results in Section IV show that our algorithm performs better than MCF [3] in terms of schedulability ratio and closely follows the optimal fluid algorithm MC-Fluid [2]. It is also shown that both MC-Sort and MC-Slope algorithm have a linearithmic (i.e., *nlogn*) complexity in the number of tasks n. Thus, both the proposed algorithms significantly improve schedulability when compared to MCF, with a marginal loss in time complexity.

Dual-rate fluid scheduling of MC task systems on a multi-core is not optimal; i.e., there are tasksets that are schedulable by some algorithm but are deemed to be not schedulable by the dual-rate MC-Fluid algorithm. In Section V we present a simple example that is not schedulable by any dual-rate fluid

scheduler whereas, a multi-rate ($>$ 2 execution rates for each task) fluid scheduler successfully schedules it. Thus, exploring multi-rate fluid scheduling algorithms to further improve schedulability is a worthy research direction to consider, even though speed-up optimality has already been achieved.

Related Work: The concept of mixed-criticality systems was introduced by Vestal [4]. Several studies have been done on single-core MC scheduling in recent years; see [5] for review. As the recent trend in the chip industry is towards multi-cores, there have been some studies on multi-core MC scheduling as well. Initial work on multi-core MC scheduling is by Anderson et al. [6] is based on hierarchical scheduling in which they proposed a mix of partitioned and global approaches. A global fixed-priority scheduling algorithm based on response time analysis was proposed by Pathan et al. [7]. Li and Baruah [8] proposed a global scheduling algorithm by combining a multi-core fixed priority algorithm fpEDF and single-core virtual deadline based MC algorithm EDF-VD. Baruah et al. [9] also presented a partitioned scheduling algorithm based on EDF-VD and showed that partitioned scheduling performs better than global scheduling with respect to schedulability ratio. Rodriguez et al. [10] compared the performance of different partitioning heuristics for the partitioned EDF-VD algorithm. Guan et al. [11] extended the work on single-core demand bound function to multi-core and presented two enhancements to improve the overall system schedulability and heavy low-critical task schedulability. Ren et al. [12] proposed a partitioned scheduling algorithm based on compositional scheduling and task grouping that offers strong isolation for high-critical tasks and improved real-time performance for low-critical tasks. In contrast to the above studies, we focus on global fluid scheduling algorithms because they have been shown to have good, theoretically bounded, performance.

II. Background

A. System Model

MC scheduling problem is considered for an implicit-deadline sporadic task system scheduled on m identical cores. In this paper, we restrict ourselves to a dual-criticality system (namely LO and HI) as in [2] [3].

Tasks: We consider a sporadic taskset τ, in which each MC task τ_i is characterized by a tuple (T_i,C_i^L,C_i^H,X_i), where $T_i \in \mathbb{R}^+$ is the minimum release separation time, $C_i^L \in \mathbb{R}^+$ is the LO-criticality Worst-Case Execution Time (WCET), $C_i^H \in \mathbb{R}^+$ is the HI-criticality WCET (HI-WCET); we assume $C_i^L \leq C_i^H$ and $X_i \in \{LO, HI\}$ is the criticality level. We assume an implicit-deadline task model in which each task τ_i has a relative deadline equal to T_i.

Notation: We consider a dual-criticality sporadic taskset τ with n tasks. LO-criticality taskset τ_L and HI-criticality taskset τ_H are defined as $\tau_L \stackrel{\text{def}}{=} \{\tau_i \in \tau \mid Xi = LO\}$ and $\tau_H \stackrel{\text{def}}{=} \{\tau_i \in \tau \mid Xi = HI\}$. LO-criticality and HI-criticality utilization of a task τ_i is defined as $u_i^L \stackrel{\text{def}}{=} C_i^L/T_i$ and $u_i^H \stackrel{\text{def}}{=} C_i^H/T_i$ respectively. System-level utilizations of a taskset τ are defined as: $U_L^L \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau_L} C_i^L/T_i$, $U_H^L \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau_H} C_i^L/T_i$ and $U_H^H \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau_H} C_i^H/T_i$.

MC Behaviour: If each task $\tau_i \in \tau$ signals completion before exceeding its LO-WCET, then the system is said to be in LO-criticality behaviour or LO-mode. If any HI-task $\tau_i \in \tau_H$ signals completion after executing beyond its LO-WCET and before exceeding its HI-WCET, then the system is said to be in HI-criticality behaviour or HI-mode. Mode change represents the change in criticality level of the system from LO to HI. System initially starts in LO-mode, and switches to HI-mode at the earliest time instant when any HI-task executes beyond its LO-WCET without signalling completion. After mode switch all LO-tasks are discarded by the system. If at any point a LO-task executes beyond its LO-WCET in LO-mode or a HI-task executes beyond its HI-WCET in HI-mode, then the system behaviour is said to be erroneous.

MC Schedulability: A taskset τ is said to be MC schedulable by a scheduling algorithm if,

- In LO-mode, each instance of each task in τ is able to complete LO-WCET execution within its deadline, and
- In HI-mode, each instance of each HI-task in τ is able to complete HI-WCET execution within its deadline.

B. Dual-Rate Fluid Scheduling

Dual-rate fluid scheduling algorithm was designed to schedule dual-criticality implicit-deadline sporadic task systems on an identical multi-core platform. Dual-rate fluid scheduling can be summarized as follows:

- In LO-mode, each task $\tau_i \in \tau$ executes at a constant rate θ_i^L (where $\theta_i^L \in (0, 1]$).
- After mode switch, all tasks in τ_L are discarded immediately and each HI-task starts executing at a constant rate θ_i^H (where $\theta_i^H \in [\theta_i^L, 1]$).

MC-Fluid uses the criticality-dependent execution rates θ_i^L and θ_i^H to schedule each task τ_i [2]. MC-Fluid formulates an optimization problem to determine the execution rates where, θ_i^H is the solution to the optimization problem,

$$\begin{aligned} minimize \quad & \sum_{\tau_i \in \tau_H} \frac{u_i^L(u_i^H - u_i^L)}{\theta_i^H - u_i^H + u_i^L} && \leq m \\ subject \;\; to \quad & \sum_{\tau_i \in \tau_H} \theta_i^H - m && \leq 0 \\ & \forall \tau_i \in \tau_H, \quad -\theta_i^H + u_i^H && \leq 0 \\ & \forall \tau_i \in \tau_H, \quad \theta_i^H - 1 && \leq 0 \end{aligned}$$

The θ_i^H values are determined by solving the convex optimization problem. The speed-up factor of MC-Fluid is shown to be 4/3, which is optimal among all multi-core MC scheduling algorithms [3].

MCF is a simplified variant of MC-Fluid that has linear run-time complexity in the number of tasks n [3]. MCF, like MC-Fluid, tries to compute the execution rates θ_i^L and θ_i^H to meet the MC schedulability condition. It can be summarized as follows:

- Compute ρ where,

$$\rho \leftarrow max\left\{(\frac{U_L^L + U_H^L}{m}), (\frac{U_H^H}{m}), \underset{\tau_i \in \tau_H}{max}\{u_i^H\}\right\}$$

- **If** $\rho \leq 1$ compute θ_i^H and θ_i^L **else** declare failure;

$$\theta_i^H \leftarrow \frac{u_i^H}{\rho}, \text{for all } \tau_i \in \tau_H$$

$$\theta_i^L \leftarrow \begin{cases} \frac{u_i^L.\theta_i^H}{\theta_i^H - u_i^H + u_i^L}, & \text{if } \tau_i \in \tau_H, \\ u_i^L, & \text{otherwise} \end{cases}$$

- **If** $\sum_{\tau_i \in \tau} \theta_i^L \leq m$ declare success **else** declare failure

The key difference between MCF and MC-Fluid is that MCF uses a simple rate assignment strategy with linear run-time complexity as opposed to the convex optimization framework of MC-Fluid. Although MCF has lower schedulability performance when compared to MC-Fluid in experiments, it is also shown to have an optimal speed-up bound of 4/3.

A taskset τ is said to be MC schedulable under dual-rate fluid scheduling iff

$$\begin{aligned} \forall \tau_i \in \tau, \quad & \theta_i^L \geq u_i^L, \\ \forall \tau_i \in \tau_H, \quad & \frac{u_i^L}{\theta_i^L} + \frac{u_i^H - u_i^L}{\theta_i^H} \leq 1, \\ & \sum_{\tau_i \in \tau} \theta_i^L \leq m, \\ & \sum_{\tau_i \in \tau_H} \theta_i^H \leq m. \end{aligned}$$

III. Proposed Rate Assignment Strategies

In this section, we present two new algorithms, namely MC-Sort and MC-Slope, for computing the execution rates θ_i^L and θ_i^H for each task τ_i. MC-Sort and MC-Slope algorithm can be summarized as follows:

- Compute θ_i^H for all $\tau_i \in \tau_H$ using MC-Sort/MC-Slope HI-rate assignment,
- **If** $\sum_{\tau_i \in \tau_H} \theta_i^H \leq m$ compute θ_i^L **else** declare failure;

$$\theta_i^L \leftarrow \begin{cases} \frac{u_i^L.\theta_i^H}{\theta_i^H - u_i^H + u_i^L}, & \text{if } \tau_i \in \tau_H, \\ u_i^L, & \text{otherwise} \end{cases}$$

- **If** $\sum_{\tau_i \in \tau} \theta_i^L \leq m$ declare success **else** declare failure.

A. MC-Sort Algorithm

MC-Sort rate assignment algorithm sorts all the HI-tasks based on their HI-WCET values, and assigns the maximum possible rate in HI-mode (Θ_i^H) for each HI-task in that order. The detailed steps of this strategy for assigning the HI-rates are given in Algorithm 1.

The run-time complexity of the MC-Sort algorithm is linearithmic in the number of tasks in τ. Initial assignment of θ_i^H can be done in a single pass through all the HI-tasks in the system. Sorting θ_i^H can be done in $O(nlogn)$ time, where n is the total number of tasks in the system. Final assignment

Algorithm 1 MC-Sort HI-rate assignment

Input: τ_H, m and for each θ_i^H assign an initial value of $\frac{u_i^H}{max\left\{(\frac{U_H^H}{m}), u_i^H\right\}}$

Output: θ_i^H for each $\tau_i \in \tau_H$

1: Sort τ_H in decreasing order of u_i^H
2: **for** j := 1 to length(τ_H) **do**
3: **if** $(m - \sum_{\tau_i \in \tau_H} \theta_i^H) > 0$ and $u_i^H \neq u_i^L$ **then**
4: **if** $(m - \sum_{\tau_i \in \tau_H} \theta_i^H) \geq (1 - \theta_i^H)$ **then**
5: Update θ_i^H to 1.0
6: **else if** $(m - \sum_{\tau_i \in \tau_H} \theta_i^H) < (1 - \theta_i^H)$ **then**
7: Update θ_i^H to $\theta_i^H + (m - \sum_{\tau_i \in \tau_H} \theta_i^H)$
8: **else**
9: Break
10: **end if**
11: **end if**
12: **end for**

of θ_i^H and θ_i^L can be done in linear time, and therefore the overall run-time complexity of MC-Sort is $O(nlogn)$.

If algorithm MC-Sort successfully determines the rates, then the schedule resulting from using these rates will result in a correct MC-scheduling strategy. From Line 5, it is evident that the individual tasks' execution rate never exceeds 1. The condition in Line 3 ensures that the total execution rate of all tasks do not exceed the system capacity. The initial assignment for θ_i^H, and θ_i^L computation are the same as in MCF. For correctness proof please refer to theorem 1 in [3].

B. MC-Slope Algorithm

Another rate assignment algorithm with linearithmic run-time complexity called MC-Slope is presented in this section. MC-Slope assigns execution rates based on the rate of change of task τ_is component ($O(\theta_i^H)$) in the objective function of the optimization problem in [2]. The rate of change of objective function $O(\theta_i^H)$, $R(\theta_i^H)$, is defined as follows.

$$\begin{aligned} O(\theta_i^H) &= \frac{u_i^L(u_i^H - u_i^L)}{\theta_i^H - u_i^H + u_i^L} \\ R(\theta_i^H) &= \frac{d^2 O(\theta_i^H)}{d\theta_i^{H2}} \\ &= \frac{2.u_i^L(u_i^H - u_i^L)}{(\theta_i^H - u_i^H + u_i^L)^3} \end{aligned} \tag{1}$$

Algorithm 2 below then gives the HI-rate assignment strategy of MC-Slope. Line 1 in Algorithm 2 sorts all the HI-criticality tasks in increasing order of $R(u_i^H)$. It considers one HI-task (τ_j) at a time from this sorted list (Line 2). For each HI-task (τ_i) such that $i > j$, it assigns rate θ_i^H such that $R(\theta_i^H)$ is decreased to match $R(\theta_j^H)$. This can be computed using Equation (1) by considering θ_i^H as the unknown quantity with a fixed value for θ_i^H. If the assigned θ_i^H rate is greater than 1, θ_i^H will be updated to 1. Line 5 checks if the assigned rates are feasible, and if not, then the above process is repeated

Algorithm 2 MC-Slope HI-rate assignment

Input: τ_H, m and for each θ_i^H assign an initial value of u_i^H
Output: θ_i^H for each $\tau_i \in \tau_H$

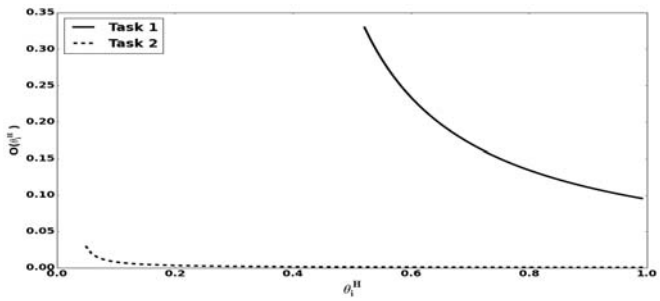
1: Sort τ_H in increasing order of $R(\theta_i^H)$ at $\theta_i^H = u_i^H$
2: **for** j := 1 to length(τ_H) **do**
3: Compute θ_i^H for each $\tau_i \in \tau_H$ s.t. $i > j$ $R(\theta_i^H) = R(\theta_j^H)$
4: Update θ_i^H to 1.0 if $\theta_i^H > 1$ for each $\tau_i \in \tau_H$
5: **if** $\sum_{\tau_i \in \tau_H} \theta_i^H \leq m$ **then**
6: Break
7: **else**
8: Continue
9: **end if**
10: **end for**
11: Slack = m - $\sum_{\tau_i \in \tau_H} \theta_i^H$
12: Sum_$O(\theta_i^H)$ = $\sum_{\tau_i \in \tau_H, \theta_i^H \neq 1} O(\theta_i^H)$
13: **for** i := length(τ_H) to 1 **do**
14: **if** $(\theta_i^H \neq 1)$ and (Slack > 0) **then**
15: Update θ_i^H to $\theta_i^H + \frac{Slack * O(\theta_i^H)}{Sum\_O(\theta_i^H)}$
16: **if** $\theta_i^H \geq 1$ **then**
17: Update θ_i^H to 1.0
18: **end if**
19: **end if**
20: **end for**

for the next HI-task τ_j in the sorted list. In the worst-case, the for loop exits without any modification to the initial assignment of θ_i^H, which is always feasible. Line 11 computes the remaining slack in the system after the above assignment. Lines 13-18 allocates this remaining slack to all the HI-tasks proportionately, based on their updated $O(\theta_i^H)$ values.

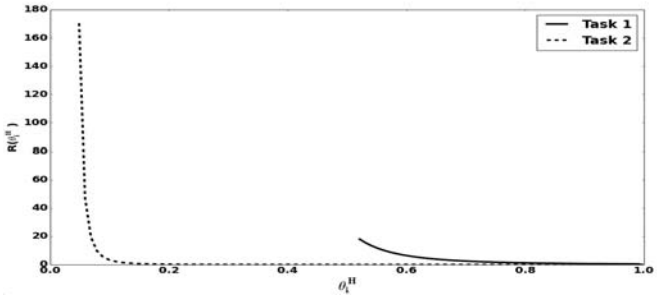
Objective function $O(\theta_i^H)$ and the rate of change of objective function $R(\theta_i^H)$ of a sample taskset is plotted against θ_i^H in Figure 1. Task 1 has a larger $O(\theta_i^H)$ value compared to task 2, whereas, task 2 has larger $R(\theta_i^H)$. MC-Slope assigns θ_i^H rate to task 2 first, until its $R(\theta_i^H)$ value becomes equal to that of task 1. If slack remains after their $R(\theta_i^H)$ becomes equal, it proportionately allocates the slack based on their updated $O(\theta_i^H)$ values. The rationale behind MC-Slope strategy is that the algorithm tries to minimize the total objective function by allocating a larger portion of the execution rate to tasks with faster decreasing $R(\theta_i^H)$.

The schedule resulting from using the execution rates computed by the MC-Slope algorithm constitutes a correct MC-scheduling strategy. Lines 4 and 17 in Algorithm 2 ensure execution rates do not exceed 1. Lines 5 and 14 guarantees that the total execution rate of tasks does not exceed the system capacity. The θ_i^L computation is the same as in other fluid algorithms. MC-Slope algorithm declares success if the total LO-rate of the tasks do not exceed the processing capacity.

MC-Slope algorithm has a run-time complexity of $O(nlogn)$ in the number of tasks in τ. Sorting $R(\theta_i^H)$ in MC-Slope algorithm can be done in $O(nlogn)$ time complexity. Selecting a task with least $R(\theta_i^H)$ that satisfies the condition



(a) Objective Function - O(θ_i^H)



(b) Rate of change of objective function - R(θ_i^H)

Fig. 1: MC-Slope Algorithm

in Line 5 can be done using binary search. Thus, the outer FOR loop of Line 2 runs at most $O(logn)$ times. Computing θ_i^H in Line 3 consumes linear time in the number of tasks n. Proportionately allocating the remaining slack to the tasks can also be done in linear time. The overall complexity of the MC-Slope algorithm is thus $O(nlogn)$.

IV. Experiments and Results

Experiments with randomly generated tasksets are conducted to compare the performance of the proposed algorithms with the existing fluid algorithms MCF [3] and MC-Fluid [2]. In this section, we first present the experiment setup and then discuss the results.

A. Experiment Setup

Task set generation: Our experiments are carried out for a dual-criticality implicit-deadline task systems scheduled on an m-core platform. We use the same approach for generating random tasksets as in earlier studies [3]. The task parameters used in our experiments are described as follows:

1) $U^B \in [0.1, 0.15, ..., 1.0]$ denotes the normalized system utilization in both LO and HI modes.
2) $P^H \in [0.1, 0.2, ..., 1.0]$ denotes the probability of a task to be HI-task.
3) Minimum and maximum individual task utilization u_{min} ($= 0.02$) and u_{max} ($= 0.90$).
4) $m \in \{2, 4, 8\}$ denotes the total number of cores.
5) T_i, the period of task τ_i is drawn uniformly at random from $[20, 300]$.
6) P_i is drawn uniformly at random from $[0, 1]$. If $P_i < P^H$, then X_i = HI else X_i = LO.
7) Task utilization u_i is drawn from the range [u_{min},u_{max}]. If X_i = HI, then $u_i^H = u_i$ else $u_i^L = u_i$. If X_i = HI,

$u_i^L = u_i^H/R$, where R is an integer drawn uniformly at random from the range $[1, 4]$.

8) Execution requirements C_i^L and C_i^H are derived as $\lceil u_i^L * T_i \rceil$ and $\lceil u_i^H * T_i \rceil$ respectively.

The steps $5-8$ are repeated to generate tasks until the system utilization condition $\max\{\frac{U_L^L+U_H^L}{m}, \frac{U_H^H}{m}\} \leq U^B$ is met. Once the condition is violated, the last generated task is discarded. If the resulting taskset has a normalized utilization between U^B - 0.05 and U^B, then the taskset is accepted, else the taskset is discarded and the procedure is repeated again. We evaluate the performance of four algorithms MC-Fluid, MC-Slope, MC-Sort and MCF for each successfully generated taskset.

B. Results

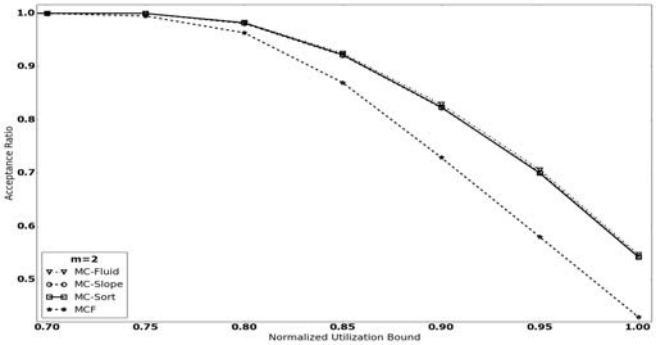
Figures 2a-2c show the acceptance ratios of the four algorithms i.e., fraction of schedulable tasksets, versus normalized average utilization U^B over varying $m \in \{2, 4, 8\}$ and fixed P^H $(= 0.5)$. Each data point in the figure corresponds to $10,000$ tasksets. The results show that both MC-Sort and MC-Slope outperform MCF, and the difference in schedulability is marginal when compared to MC-Fluid.

Figure 3a shows the effect of varying P^H values over the weighted acceptance ratio. Weighted acceptance ratio is defined as $WAR(\mathbb{S}) = \frac{\sum_{U^B \in \mathbb{S}} (AR(U^B) X U^B)}{\sum_{U^B \in \mathbb{S}} U^B}$ where, $\mathbb{S}$ is the set of U^B values and AR(U^B) is the acceptance ratio for a specific value of the normalized utilization U^B. All the algorithms perform well at the extreme probability values as tasksets contain only either LO or HI tasks. It can be seen that both MC-Sort and MC-Slope outperforms MCF algorithm for all the P^H values. The performance of MC-Slope is marginally better than that of MC-Sort for P^H values greater than 0.6. This is as expected because MC-Slope uses the rate of change in the objective function to determine HI-rates, whereas MC-Sort simply uses the execution requirement in HI-mode.

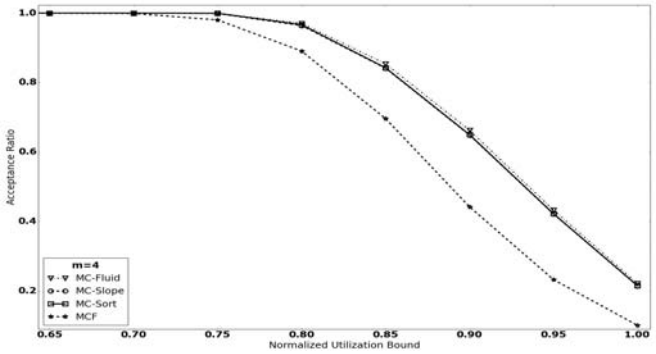
Figure 3b shows the results of weighted acceptance ratio for varying range of u_i^H/u_i^L $(\in [1, 1.5], [1.5, 2.0], [2.0, 2.5], [2.5, 3.0], [3.0, 3.5], [3.5, 4.0])$ with fixed P^H $(= 0.5)$ and u_{max} $(= 0.90)$ values. Only the upper bound of the ranges is presented in the plot. It can be seen that as the ratio u_i^H/u_i^L becomes larger there is a drop in the performance of all the algorithms. However, unlike MCF, MC-Slope and MC-Sort continue to perform exceedingly well in comparison to the optimal MC-Fluid.

V. Discussions and Future Work

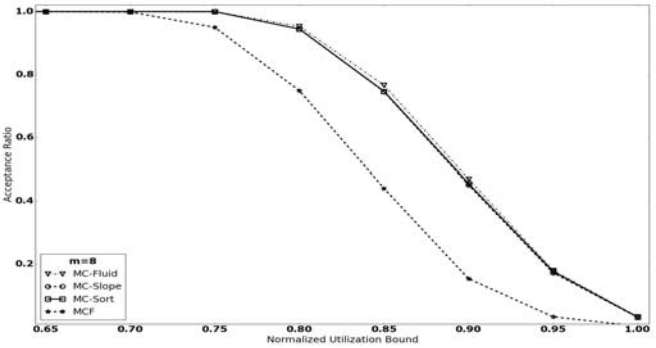
MC-Fluid is shown to be an optimal rate assignment strategy for a dual-rate dual-criticality task systems [2]. That is, if there exists an assignment of θ_i^H and θ_i^L for all the tasks that satisfies MC schedulability condition, then MC-Fluid is guaranteed to find such an assignment. As fluid schedules are not implementable on actual computing platforms due to their fractional allocations, MC-DP-Fair translates the fluid schedules to non-fluid ones without any loss in performance [2]. MCF on the other hand, presents a sub-optimal rate assignment strategy with linear complexity. Both these strategies have been shown to result in optimal speed-up bounds.



(a) m = 2



(b) m=4



(c) m=8

Fig. 2: Comparison of acceptance ratio

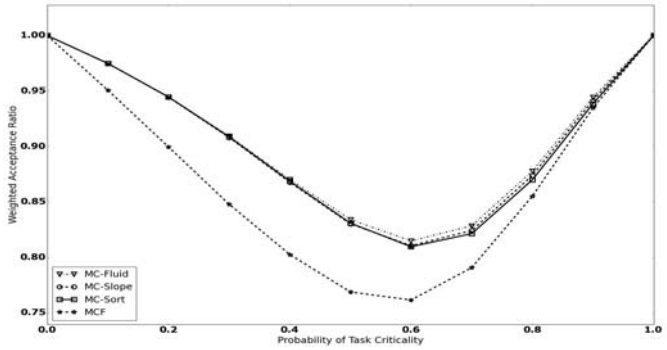
As a part of the future work, we plan to derive the speed-up bounds for the two proposed algorithms, and also identify a linearithmic complexity algorithm with an optimal rate assignment strategy.

Dual-rate fluid scheduling is not optimal among all fluid scheduling algorithms. In particular, algorithms with more than two rate assignments per task can further improve schedulability. Consider a dual-core system with the periodic taskset τ and its execution rate as shown in Table I. Taskset in the table is not MC-schedulable with the dual-rate assignment θ_i^L and θ_i^H given by the MC-Fluid algorithm. We can check that $\sum_{\tau_i \in \tau} \theta_i^L$ is greater than 2.

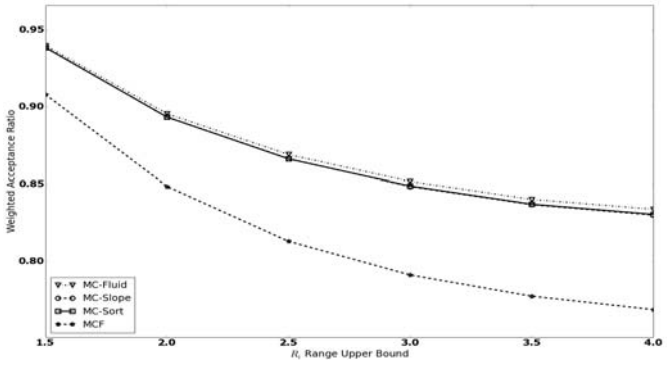
Each MC task is characterized by a sequence of job releases. A job of a task τ_i is said to be a carry-over job, if it is released

TABLE I: Example taskset and its rate assignment

Tasks	C_i^L	C_i^H	T_i	u_i^L	u_i^H	MC-Fluid		Proposed multi-rate model			
						θ_i^L	θ_i^H	δ_i^L	δ_i^{H*}	δ_i^C	δ_i^H
τ_1	1.5	4	5	0.3	0.8	0.641	0.939	0.641	0.939	-	0.8
τ_2	2.8	4.9	7	0.4	0.7	0.700	0.700	0.700	0.700	0.700	0.7
τ_3	3.5	10.5	35	0.1	0.3	0.224	0.360	0.209	0.360	0.499	0.3
τ_4	15.75	-	35	0.45	-	0.450	-	0.450	-	-	-
$\sum$						2.015	1.999	2.00	1.999	1.199	1.80



(a) Varying probability of a task to be HI-Criticality



(b) Varying u_i^H/u_i^L ranges

Fig. 3: Comparison of weighted acceptance ratio

before the mode switch and has not completed its execution until the mode switch.

We plan to improve the fluid scheduling framework by considering multiple rates for each task. The jobs of a HI-task τ_i released before mode switch execute with δ_i^L in LO-mode and the jobs released after mode switch execute with δ_i^H in HI-mode. Here, $\delta_i^H = u_i^H$ as no jobs require more than its HI-utilization for completion. The carry-over jobs of HI-tasks execute with δ_i^{H*} from mode switch until the earliest time instant at which a carry-over job of any HI-task complete its execution, and then execute with δ_i^C ($\geq \delta_i^{H*}$) until its completion. Figure 4 represents the proposed multi-rate model in which each task executes with more than two rates.

By assigning multiple rates to the HI-tasks, the taskset in Table I is shown to be MC-schedulable. Let us assume task τ_1 initiates the mode switch as shown in Figure 4. It is sufficient for jobs of τ_1 released after mode switch to execute with $u_1^H(=0.8)$. The difference in δ_1^{H*} and $u_1^H(=0.939-0.8)$ can be used for executing other HI-tasks; in this case, for task τ_3. By allowing additional execution rate for task τ_3 in HI-mode, we can bring down its LO-mode rate. Now we can check that

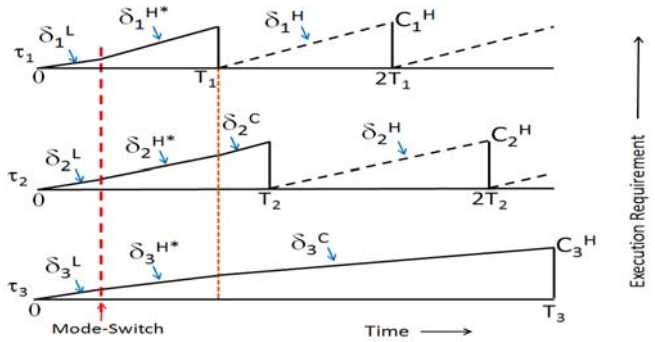


Fig. 4: Proposed multi-rate model

$\sum_{\tau_i \in \tau} \delta_i^L$ is less than or equal 2. Therefore, by assigning more than two rates to each task it is possible to schedule tasksets that are deemed to be not MC-schedulable by dual-rate fluid scheduling algorithms.

REFERENCES

[1] G. Levin, S. Funk, C. Sadowski, I. Pye, and S. Brandt, "Dp-fair: A simple model for understanding optimal multiprocessor scheduling," in *Real-Time Systems (ECRTS), 22nd Euromicro Conference on*, July 2010.

[2] 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), 35th IEEE International*, Dec 2014.

[3] S. Baruah, A. Easwaran, and Z. Guo, "MC-Fluid: simplified and optimally quantified," in *Real-Time Systems Symposium (RTSS), 36th IEEE International*, Dec 2015.

[4] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *Real-Time Systems Symposium, 28th IEEE International*, Dec 2007.

[5] A. Burns and R. I. Davis. (2013) Mixed Criticality Systems - A Review. http://www-users.cs.york.ac.uk/burns/review.pdf.

[6] J. H. Anderson, S. K. Baruah, and B. B. Brandenburg, "Multicore operating-system support for mixed criticality," in *Workshop on Mixed Criticality: Roadmap to Evolving UAV Certification, 2009*, Apr 2009.

[7] R. Pathan, "Schedulability analysis of mixed-criticality systems on multiprocessors," in *Real-Time Systems (ECRTS), 24th Euromicro Conference on*, July 2012.

[8] H. Li and S. Baruah, "Outstanding paper award: Global mixed-criticality scheduling on multiprocessors," in *Real-Time Systems (ECRTS), 24th Euromicro Conference on*, July 2012.

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

[10] P. Rodriguez, L. George, Y. Abdeddaim, and J. Goossens, "Multi-criteria evaluation of partitioned edf-vd for mixed-criticality systems upon identical processors," in *Workshop on Mixed Criticality Systems (WMC), 2013*, December.

[11] C. Gu, N. Guan, Q. Deng, and W. Yi, "Partitioned mixed-criticality scheduling on multiprocessor platforms," in *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, March 2014.

[12] J. Ren and L. T. X. Phan, "Mixed-criticality scheduling on multi-processors using task grouping," in *Real-Time Systems (ECRTS), 27th Euromicro Conference on*, July 2015.