Evaluation of Mixed-Criticality Scheduling Algorithms using a Fair Taskset Generator

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

***Abstract*—The problem of scheduling mixed-criticality (MC) task systems is known to be NP-Hard, and as a consequence the performance of MC scheduling algorithms is frequently assessed using experimental evaluations based on randomly generated tasksets. It is therefore important to have a thorough understanding of all the parameters that impact the algorithms and a taskset generation procedure that is fair with respect to those parameters. Although there are a few popular taskset generators, there is no evaluation of the fairness properties of those generators. In fact, there is no existing study on identifying all the parameters that are relevant in the evaluation of MC scheduling algorithms. We address this shortcoming in this paper, and present a set of essential fairness properties for MC taskset generators. We also develop a new taskset generator and show that it satisfies those fairness properties. Finally, we evaluate the performance of multi-core MC scheduling algorithms using the generator, and provide new insights on the performance of those algorithms with respect to several taskset parameters.**

I. Introduction

Mixed-Criticality (MC) scheduling has received a lot of attention in the real-time literature ever since Vestal proposed the MC task model [1]. This is mainly because of the practical relevance of MC systems in safety-critical industries such as avionics and automotive. There have been several studies, both on single- as well as multi-cores, focusing on the design of scheduling algorithms for the MC task model; see [2] for review. One way to evaluate algorithm performance is analytical, wherein metrics such as speed-up bound [3] are derived. For MC systems it has been shown that the scheduling problem is NP-Hard [4]. Consequently, the only known analytical performance results are in terms of speed-up bounds.

Another mechanism to evaluate algorithm performance is experimental, wherein a *taskset generator* is used to generate a variety of MC task systems, and the algorithms are evaluated by testing their schedulability on these task systems. The driving principle behind such experimental evaluation is that as long as the set of generated task systems is "fair", meaning not biased in terms of the parameters used to define the task system, the resulting comparisons provide a fair assessment on the relative performance of the algorithms. Although, unlike speed-up bounds, these evaluations do not provide any analytical guarantees, they are being increasingly used in the evaluation of MC algorithms [5], [6], [7], [8]. This trend is because of two factors: 1) For many algorithms such as those based on heuristics or non-trivial schedulability tests (e.g., those derived from demand bound functions), it is extremely hard, if not impossible, to derive these speed-up bounds. 2) For algorithms with known speed-up bounds such as EDF-VD, either the bounds are not very tight as in constrained-deadline task systems, or the bounds are not representative of the performance of the algorithm in practice. For example, although EDF-VD has an optimal speed-up bound of $4/3$ for dual-criticality implicit-deadline task systems [9], its performance is shown to be relatively poor in experimental evaluations [5]. Thus, in the absence of tight analytical results on the performance of MC algorithms, it is important to design taskset generators that enable a fair experimental evaluation.

There have been few studies on taskset generators for MC systems ([5], [6], [7], [8], [10], [11]). Although they present different taskset generation algorithms, there is no work which methodically considers all the parameters that impact the performance of MC algorithms. As a consequence, there is neither any clear understanding of what constitutes a fair MC taskset generator, nor is there any discussion in these studies on the fairness properties of the generators themselves. Note that, when compared to non-MC systems, a much larger number of parameters affect the performance of MC algorithms. This is because tasks in MC systems have additional parameters such as resource utilization values at different confidence levels, and further these parameters are known to have a significant impact on algorithm performance. In this paper, we address this challenging problem by first presenting the principles that govern the fairness of a MC taskset generator. We then present a novel taskset generation algorithm for MC systems and show that it satisfies these principles. Similar to the UUnifast algorithm for single-core non-MC systems [12] and MRandFixedSum algorithm for multi-core non-MC systems [13], we believe that the taskset generation algorithm presented here can be used to experimentally evaluate MC algorithms in a fair manner. Thus, the contributions of this paper can be summarized as follows:

- We present the fairness properties (Section III-A) for any MC taskset generator based on all the parameters that affect the performance of MC scheduling algorithms. We identify some new parameters that influence schedulability, which were not considered in the existing generators.
- We propose a MC taskset generator that generates tasksets satisfying the above fairness properties (Section III-B).
- We present extensive experimental evaluation for multi-core MC scheduling algorithms with the proposed taskset generator (Section IV).

II. System Model

In this section we define our system model. We restrict our model to a dual-criticality system (namely LO and HI).

Tasks: We consider a sporadic taskset τ, in which each MC task τ_i is characterized by a tuple $(T_i,\chi_i,\vec{C}_i,D_i)$, where

- $T_i \in \mathbb{R}^+$ is the minimum release separation time,
- $\chi_i \in \{LO, HI\}$ is the criticality level,
- $\vec{C}_i \in \mathbb{R}^+$ is the vector of Worst-Case Execution Time (WCET) values - one for each criticality level. C_i^L and C_i^H are the LO- and HI-criticality WCET values respectively; we assume $C_i^L \leq C_i^H$ and,
- $D_i \in \mathbb{R}^+$ is the relative deadline; for implicit deadlines D_i = T_i and for constrained deadlines $D_i \leq T_i$.

Taskset: We consider a dual-criticality sporadic taskset τ with n tasks, where a task τ_i represents an infinite number of job releases. LO- and HI-criticality utilization of a task τ_i is defined as $u_i^L \overset{\text{def}}{=} C_i^L/T_i$ and $u_i^H \overset{\text{def}}{=} C_i^H/T_i$ respectively. System-level normalized utilizations are defined as $U_L^L \overset{\text{def}}{=} \sum_{\tau_i \in \tau_L} u_i^L/m$, $U_H^L \overset{\text{def}}{=} \sum_{\tau_i \in \tau_H} u_i^L/m$ and $U_H^H \overset{\text{def}}{=} \sum_{\tau_i \in \tau_H} u_i^H/m$, where m is the number of cores.

MC Modes: The system is said to be in LO-criticality mode or LO-mode if all the tasks $\tau_i \in \tau$ signal completion before exceeding LO-WCET. The system is said to be in HI-criticality mode or HI-mode if any HI-task $\tau_i \in \tau_H$ executes beyond its LO-WCET and signals completion before exceeding its HI-WCET. Mode switch is defined as the change in criticality level of the system from LO to HI. All LO-tasks are immediately discarded by the system at mode switch. We focus on the above MC model because this is the standard model in many studies on MC scheduling.

III. Fairness and Taskset Generator

In this section we describe the fairness properties that are essential for any MC taskset generator. We also describe our new taskset generator *MC-FairGen*, and compare its fairness properties against several existing generators.

A. Essential Fairness Properties

The performance of MC scheduling algorithms depend on several taskset parameters. Among them, the most important include task periods and deadlines, proportion of LO- and HI-criticality tasks, maximum individual task utilization, and system utilization parameters $|U_H^H - (U_H^L + U_L^L)|$, $|U_H^L - U_L^L|$ and $|U_H^H - U_H^L|$. The minimum required number of tasks in the system is $m+1$. The total utilization ($U_B = max(U_H^H, U_H^L + U_L^L)$) must range across all possible values. Thus, the essential fairness properties can be summarized as follows.

1) *Period*: Task periods must be chosen from a wide range and should have an appropriate distribution that is not biased. One way to achieve this is by choosing periods using uniform or log-uniform distribution. It has been shown that fixed-priority algorithms perform well when periods are chosen using log-uniform distribution [13].
2) *Deadline*: Task deadlines, in the case of constrained-deadline tasksets, must also be drawn from an appropriate distribution that is not biased. For example, drawing deadline values from a uniform distribution between C_i^H (or C_i^L) and T_i is one way to achieve this.
3) *Criticality*: The percentage of HI-criticality tasks in a taskset must also have an appropriate unbiased distribution (e.g., uniform across the scale from 0 to 100). The performance of algorithms (such as *criticality-aware* partitioning [7]) tend to vary when there are very few LO- or HI-criticality tasks in a taskset. Therefore it is essential to consider the boundary cases for this parameter.
4) *Maximum Task Utilization*: Maximum individual task utilization, $max\{max_i(u_i^L), max_i(u_i^H)\}$, must be fairly distributed across the range $(0, 1]$.
5) *System Utilization*: The normalized utilizations of a MC taskset include U_H^H, U_H^L and U_L^L. The three important parameters related to these utilizations are *total utilization difference* ($|U_H^H - (U_H^L + U_L^L)|$), *LO-mode utilization difference* ($|U_H^L - U_L^L|$) and *HI-criticality utilization difference* ($|U_H^H - U_H^L|$). Most of the algorithms tend to perform relatively poorly as these parameters increase in value. It is therefore essential that these three parameters are fairly distributed across the range $[0, 1]$.
6) *Independence of Parameter Distributions*: To gain further insights into behaviour of MC algorithms, beyond what could be obtained from the overall schedulability evaluations, it is necessary to evaluate them against specific parameters independent of remaining parameters. Therefore, it is essential that for each parameter, the remaining parameters are fairly distributed across possible values.

In the past, studies have used other parameters such as ‘criticality factor’ ([14], [5], [15], [6], [8], [11]) and ‘number of tasks’ [14] in their generator. Criticality factor is defined as the ratio of HI-mode to LO-mode utilization of a HI-task. Varying the criticality factor indirectly impacts the *HI-criticality utilization difference* ($|U_H^H - U_H^L|$). This variation in utilization difference is captured by the *system utilization* property. Further, by fixing the criticality factor, the maximum task utilization $max(u_i^H)$ is restricted as a function of u_i^L. It is therefore reasonable to choose system utilization parameters rather than criticality factor. Whereas, varying the number of tasks impacts the individual utilization of tasks. This parameter is captured by the *maximum task utilization* property.

Extension to Multi-Criticality: These properties can also be generalized to multi-criticality systems. The *period* and *deadline* properties remain the same. Extending *criticality* property to multi-criticality requires all possible values of task criticality to be considered for each criticality level. Extending the *maximum task utilization* property is quite straightforward; it needs to consider task utilizations across all the criticality levels. To extend the *system utilization* property one needs to consider all combinations of system utilization differences. For example, the property on *total utilization difference* ($|U_H^H - (U_H^L + U_L^L)|$) needs to be expanded to consider the utilization difference between all pairs of adjacent criticality levels.

Algorithm 1 MC-FairGen

```
Input: $m$,$u_{min}$,$u_{max}$
Output: Taskset $\tau$
 1: for $U_H^H \in [0.1, 0.2, ..., 1.0]$ do
 2:     for $U_H^L \in [0.05, 0.15, ..., U_H^H]$ do
 3:         for $U_L^L \in [0.05, 0.15, ..., 1 - U_H^L]$ do
 4:             for $P_H \in [0.1, 0.2, ..., 0.9]$ do
 5:                 Minimum required total HI-tasks, $N_{min}^H = \lceil(U_H^H * m/u_{max})\rceil$
 6:                 Minimum required total LO-tasks, $N_{min}^L = \lceil(U_L^L * m/u_{max})\rceil$
 7:                 Minimum required total tasks, $N_{min} = max(m + 1, \lceil(N_{min}^H/P_H)\rceil, \lceil(N_{min}^L/(1 - P_H))\rceil)$
 8:                 Total tasks, $N = uniform[N_{min}, 10 * m]$
 9:                 Total HI-tasks, $N_H = max((P_H * N), N_{min}^H)$
10:                 Total LO-tasks, $N_L = N - N_H$
11:                 $\forall i \in N$, the period $T_i$ = uniform$[5, 100]$
12:                 HI-task HI-utilizations $\{u_i^H\} = MRandFixedSum(U_H^H * m, N_H, u_{min}, u_{max})$
13:                 HI-task LO-utilizations $\{u_i^L\} = BoundedUniform(U_H^L, U_H^H, m, N_H, u_{min}, \{u_i^H\})$
14:                 LO-task utilizations $\{u_i^L\} = MRandFixedSum(U_L^L * m, N_L, u_{min}, u_{max})$
15:                 $\forall i \in N$, the execution requirement $C_i^L = u_i^L * T_i$
16:                 $\forall i \in N_H$, the execution requirement $C_i^H = u_i^H * T_i$
17:                 $\forall i \in N_L$, the relative deadline $D_i = uniform[C_i^L, T_i]$
18:                 $\forall i \in N_H$, the relative deadline $D_i = uniform[C_i^H, T_i]$
19:             end for
20:         end for
21:     end for
22: end for
```

B. MC-FairGen Taskset Generator

The taskset parameters considered in our generator are described as follows:

- Minimum and maximum individual task utilization u_{min} ($= 0.0001$) and u_{max} ($= 0.99$). u_{min} is required to guarantee all possible values for the percentage of HI-criticality tasks in a taskset. u_{max} is required to ensure a reasonable execution time for many schedulability tests, particularly those based on demand bound functions.
- $m \in \{2, 8\}$ denotes the total number of cores.

MC-FairGen is described in Algorithm 1. The minimum required total HI-tasks(N_{min}^H) and total LO-tasks(N_{min}^L) in the system is given by Steps 5 and 6 in Algorithm 1. The ceiling of utilization bound ensures individual task utilization to be ≤ 1. The division by u_{max} allows task utilizations to be bounded by u_{max}. The minimum required total tasks in the system N_{min} is given by Step 7, which ensures the percentage of HI-criticality tasks P_H. Further, it lower bounds the number of tasks in the system by $m + 1$. The total number of tasks in the system is then drawn uniformly at random from $[N_{min}, 10 * m]$. The upper bound($10 * m$) on the number of tasks in the system is to allow for all possible values of P_H. HI-task HI-utilizations $\{u_i^H\}$ and LO-task utilizations $\{u_i^L\}$ are obtained using *MRandFixedSum* algorithm [13].

HI-task LO-utilizations $\{u_i^L\}$ are obtained using *BoundedUniform* shown in Algorithm 2. *BoundedUniform* sorts the $\{u_i^H\}$ values in descending order, and for each u_i^H it assigns u_i^L subject to two conditions: (1) sum of the total allocated u_i^L and total minimum remaining u_i^L do not exceed the utilization bound($m * U_H^L$) and (2) each $u_i^L \leq u_i^H$.

Algorithm 2 BoundedUniform

```
Input: $U_H^L, U_H^H, m, N_H, u_{min}, \{u_i^H\}$
Output: $\{u_i^L\}$
 1: Sort $\{u_i^H\}$ in decreasing order
 2: $U_L^{rem} = U_H^L * m$
 3: $U_H^{rem} = U_H^H * m$
 4: $N_H^{rem} = N_H - 1$
 5: for $u_i \in \{u_i^H\}$ do
 6:     $U_H^{rem} = U_H^{rem} - u_i$
 7:     $u_i^L = uniform(max(u_{min}, U_L^{rem} - U_H^{rem}), min((U_L^{rem} - (N_H^{rem} * u_{min})), u_i))$
 8:     $N_H^{rem} = N_H^{rem} - 1$
 9:     $U_L^{rem} = U_L^{rem} - u_i^L$
10: end for
```

C. Fairness Properties of MC-FairGen

1) *Period, Deadline and Criticality*: MC-FairGen explicitly considers these fairness properties in the taskset generation process. All the three parameters are drawn from uniform distribution.
2) *Maximum Task Utilization*: Given system utilization values and number of tasks, *MRandFixedSum* draws task utilization values uniformly from the given range [13]. Since we consider all possible combinations of system

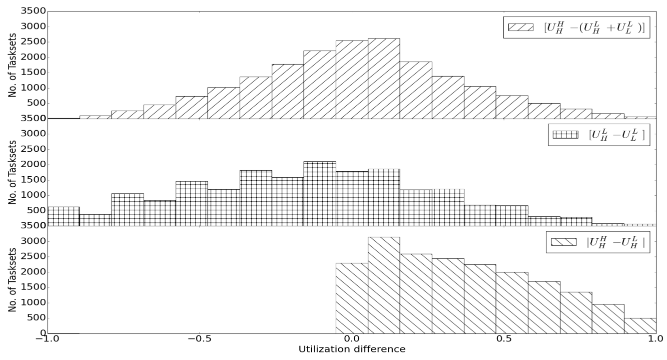
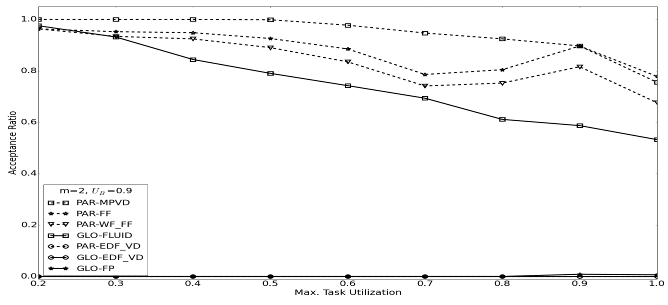


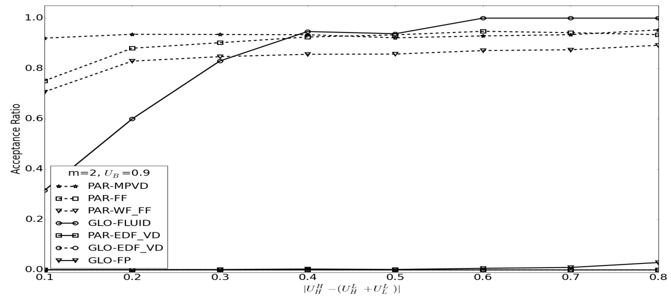
Fig. 1: Utilization distribution of MC-FairGen

utilization values, the resulting tasksets have a folded normal distribution for $max\{max_i(u_i^L), max_i(u_i^H)\}$ with mean (ν) 1.0 and standard deviation (σ) 0.46. We propose classifying the tasksets into two equal-sized classes based on the value of maximum task utilization. That is, tasksets with maximum task utilization no more than $\nu(1 - 3\sigma/4) = 0.655$ would be categorized into the "small" class, and those with value greater than this bound would be categorized into the "large" class. Figure 2a shows the performance of MC algorithms for values of maximum task utilization. We can observe that the variation in performance is not uniform across the parameter values; the performance drop is significant for larger values when compared to smaller values. This is consistent with the classification presented above; algorithm performance is more or less stable when maximum task utilization values are in the small class, and only decrease when these values are in the large class. Hence, based on this classification, we can claim that MC-FairGen satisfies the fairness property for this parameter.

3) *System Utilization*: Figure 1 shows the distribution of three system utilization differences for our generator. It can be seen that all the distributions are either normal or folded normal, as in the case of maximum task utilization. Therefore, we again classify each of these parameters into two classes, "small" and "large", as above. The cut-off values for this classification are as follows: 0.2 for $|U_H^H - (U_H^L + U_L^L)|$, 0.2 for $|U_H^L - U_L^L|$ and 0.35 for $|U_H^H - U_H^L|$. To verify that this classification is reasonable in terms of ensuring fairness with respect to the existing algorithms, we present the variation in schedulability as a function of $|U_H^H - (U_H^L + U_L^L)|$ in Figure 2b. As can be observed, the variation in performance is significant when the parameter value is less than 0.2, and minimal when the parameter value is greater than 0.2. That is, the variation *is not uniform* across this parameter. Although we do not present figures for the other two parameters due to lack of space, similar results have been observed. Thus, based on this classification of the system utilization parameters, we can claim that MC-FairGen satisfies the corresponding fairness property.



(a) Maximum Individual Task Utilization



(b) Total Utilization Difference ($|U_H^H - (U_H^L + U_L^L)|$)

Fig. 2: Varying taskset parameters

4) *Independence of Parameter Distributions*: To evaluate the schedulability performance of an algorithm against a particular parameter, it is necessary to negate the impact of all the other parameters. Let us consider the parameter $|U_H^H - (U_H^L + U_L^L)|$ whose metrics are shown in Table I. 50.41% of the tasksets are in the small class and 49.59% of the tasksets are in the large class with respect to this parameter. In each of these two classes, the distribution of tasksets for the remaining parameters are also well distributed. Note that for task periods, deadlines and criticality distribution, since we choose them independently using uniform distribution, they would also be fairly distributed in these two classes. Similar metrics have been observed for the remaining parameters as well, but we do not present them here for brevity. Thus, we can conclude that MC-FairGen also satisfies the *independence of parameter distribution* property.

Discussion on uniform distribution: In MC systems, it is extremely challenging, if not impossible, to have a uniform distribution across system utilization and maximum task utilization parameters. This is due to the constraints between the parameters.

The three system utilizations U_H^H, U_H^L and U_L^L characterize a MC system. Any valid MC system should satisfy the following two conditions: 1) $U_H^H \geq U_H^L$ and 2) $U_H^L + U_L^L \leq 1$. Satisfying the above two conditions restricts the range of values for some utilizations. Given an U_H^H value, U_H^L is bounded by U_H^H, and given an U_L^L value, U_L^L is bounded by 1-U_H^L. Say, we want to have a uniform distribution for the $|U_H^H - (U_H^L + U_L^L)|$ parameter. Lets fix U_H^H for a given $|U_H^H - (U_H^L + U_L^L)|$ value.

TABLE I: Total System Utilization Difference $|U_H^H - (U_H^L + U_L^L)|$

Parameter	Classification	% of Tasksets	Classification				Classification	
			Small	Large	Small	Large	Small	Large
			$\|U_H^L - U_L^L\|$		$\|U_H^H - U_H^L\|$		$max(max(u_i^H), max(max_i^L))$	
$\|U_H^H - (U_H^L + U_L^L)\|$	**Small**	50.41	43.65	56.35	51.46	48.54	51.15	48.85
	Large	49.59	45.36	54.64	43.10	56.90	48.51	51.49

Then we have two choices when picking $(U_H^L + U_L^L)$. Picking U_H^L or U_L^L decides the other parameter. One thing to consider here is that U_H^L value is restricted by U_H^H. This in turn restricts the U_L^L value, thereby affecting the distribution of the other two parameters $|U_H^L - U_L^L|$ and $|U_H^H - U_H^L|$. Therefore, it is reasonable to consider a normal distribution for the parameters rather than a uniform distribution, particularly given the performance variation of existing scheduling algorithms.

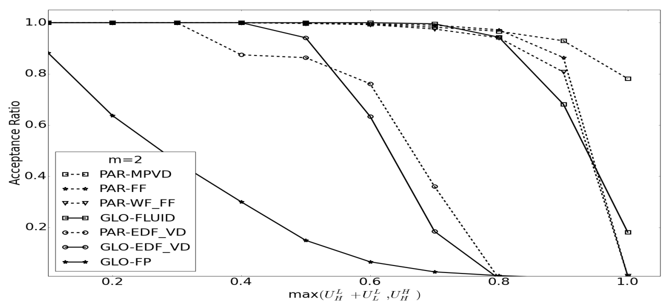
D. Comparison of Existing Generators

In this section we evaluate the existing MC taskset generators in terms of the fairness properties presented in Section III-A. We classify the existing set of generators into two major categories. The group of generators that consider the same utilization bound for both LO- and HI-mode utilizations ($U_H^L + U_L^L$ and U_H^H) fall under the first category. The group of generators that consider independent utilization bounds for LO- and HI-mode fall in the second category (denoted as class D). We further classify the first category into three classes (denoted as A,B and C) based on their taskset properties.

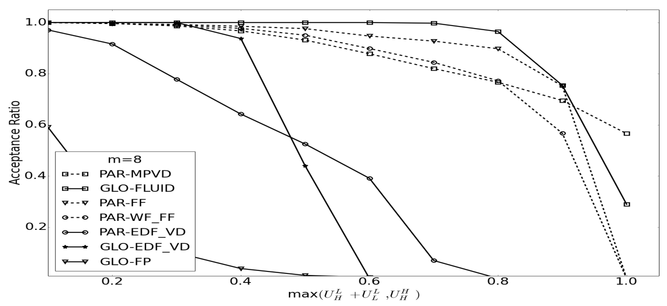
All the existing generators consider the *period* and *deadline* property. Class A generators [8] do not consider the *maximum task utilization* property. They have the property that all the generated tasksets are confined to small system utilization values. Like Class A, Class B generators ([14], [5], [6], [15], [11]) also do not satisfy the *maximum task utilization* property. Unlike Class A however, the generated tasksets of these class of generators are not confined to small system utilization values. Class C generators ([10], [16]) consider all the fairness properties except the *system utilization* properties through a set of different experiments. However, these class of generators have a high taskset discard ratio when the utilization bounds are small. Class D [7] is a reasonable generator for MC systems because it considers independent utilization bounds for LO- and HI-mode utilization. It however considers a fixed number of HI-criticality tasks in the generation process, and hence does not satisfy the *criticality* property. None of the above generators satisfy the *system utilization* properties. Thus, it is reasonable to conclude that none of the existing generators adhere to all the fairness properties listed in Section III-A.

IV. Experiments and Results

In this section we evaluate the schedulability performance of multi-core MC scheduling algorithms using MC-FairGen. These include global fpEDF [16], partitioned EDF_VD [16], global fixed-priority [15], global fluid [10], an extension of GREEDY [5] with first-fit packing strategy [7], another



(a) $m = 2$



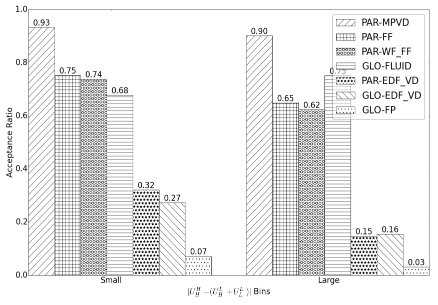
(b) $m = 8$

Fig. 3: Overall Schedulability (implicit,uniform)

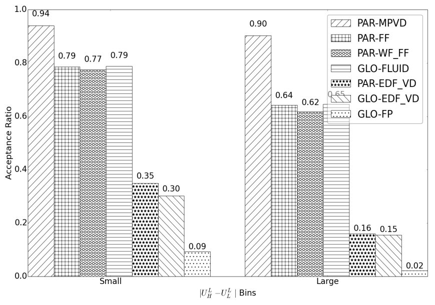
extension of GREEDY with both worst-fit and first-fit strategy [7] and MPVD with heavy low-critical task aware allocation [8] represented as GLO-EDF_VD, PAR-EDF_VD, GLO-FP, GLO-FLUID, PAR-FF, PAR-WF_FF and PAR-MPVD respectively. All the results presented are for implicit deadline task systems with uniform distribution of task periods and deadlines. Similar results were obtained for constrained deadlines. We also evaluated for log-uniform distribution of periods and deadlines, and found that there was not much variation in the performance of the algorithms except for FP scheduling. We therefore do not present them here for brevity.

In Figure 3 we present the overall schedulability of the algorithms. We plot the acceptance ratios of the algorithms i.e., fraction of schedulable tasksets, versus total utilization U_B varying over $m \in \{2, 8\}$. Each data point corresponds to at least 5000 tasksets. For $m = 2$, the partitioned demand bound function (DBF) based tests perform better than the other algorithms as shown in Figure 3a. The results obtained are consistent with the results from previous studies except for the partitioned algorithms [10].

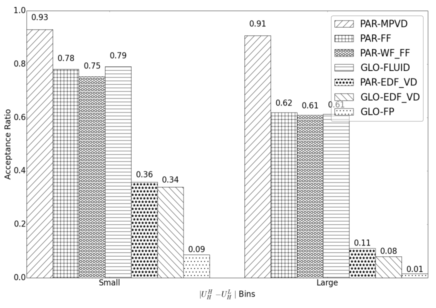
When normalized utilization nears 1.0, the performance of PAR-FF and PAR-WF_FF algorithms drop significantly compared to PAR-MPVD. The reason is that the partitioning



(a) Total Utilization $|U_H^H - (U_H^L + U_L^L)|$



(b) LO-mode Utilization $|U_H^L - U_L^L|$ $(m = 2)$



(c) HI-Crit. Utilization $|U_H^H - U_H^L|$ $(m = 2)$

Fig. 4: System Utilization Difference Distribution (implicit,uniform)

heuristics of these algorithms fail to successfully allocate the tasks. In case of PAR-FF, the tasks are allocated using first-fit strategy independent of its criticality. The problem with the first-fit is that the task utilizations are not balanced among the cores. In case of PAR-WF_FF, the HI-criticality tasks are allocated first using worst-fit approach and then the LO-criticality tasks are allocated using first-fit approach. The problem with this approach is that when there are heavy low-critical tasks in the system, it fails to get allocated to the core. Whereas, in case of PAR-MPVD, due to heavy LO-critical task aware partitioning and WF_FF bin packing approach, it performs well.

The performance of PAR-MPVD shown here is contradicting to the one presented in [11]. PAR-MPVD is known to perform better when there are heavy LO-critical tasks. As the generator in [11] generates tasksets only in low utilization ranges, it negatively affects the performance of PAR-MPVD.

To provide further insights on how algorithms perform with respect to specific parameters, we also present the performance results varying individual parameters. For brevity, we only present the schedulability results based on varying system utilization parameters i.e., $|U_H^H - (U_H^L + U_L^L)|$, $|U_H^L - U_L^L|$ and $|U_H^H - U_H^L|$ in Figure 4. All the algorithms perform well in the first class, where the three parameter values are small, and perform poorly when the values become large. GLO-FLUID algorithm performs well when $|U_H^H - (U_H^L + U_L^L)|$ is large as it mainly optimizes HI-mode execution, and performs poorly when $|U_H^L - U_L^L|$ or $|U_H^L - U_L^L|$ becomes large. All DBF based tests have more impact on $|U_H^L - U_L^L|$ and $|U_H^H - U_H^L|$ parameters when compared to $|U_H^H - (U_H^L + U_L^L)|$.

V. Summary

Taskset generators are an important tool in the evaluation of MC scheduling algorithms, mainly due to the hardness of these algorithms and the lack of quantifiable metrics such as speed-up bounds. In this paper we identified the factors that affect the schedulability of MC scheduling algorithms and presented the fairness properties that govern any MC taskset generator. We also proposed a new generator called MC-FairGen capable of generating tasksets that satisfy the fairness properties. We evaluated the performance of multi-core MC algorithms using the proposed generator. These evaluations have provided some new insights on how individual taskset parameters affect the existing algorithms, and could be used to develop improved algorithms in the future.

References

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

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

[3] B. Kalyanasundaram and K. Pruhs, "Speed is as powerful as clairvoyance [scheduling problems]," in *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, Oct 1995.

[4] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie, "Scheduling real-time mixed-criticality jobs," *Computers, IEEE Transactions on*, vol. 61, no. 8, Aug 2012.

[5] P. Ekberg and W. Yi, "Bounding and shaping the demand of mixed-criticality sporadic tasks," in *Real-Time Systems (ECRTS), 24th Euromicro Conference on*, July 2012.

[6] A. Easwaran, "Demand-based scheduling of mixed-criticality sporadic tasks on one processor," in *Real-Time Systems Symposium (RTSS), 34th IEEE International*, Dec 2013.

[7] 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)*, December 2013.

[8] 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.

[9] 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 *Real-Time Systems (ECRTS), 24th Euromicro Conference on*, July 2012.

[10] 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.

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

[12] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1-2, may 2005.

[13] P. Emberson, R. Stafford, and R. I. Davis, "Techniques for the synthesis of multiprocessor tasksets," in *1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, July 2010.

[14] S. Baruah, A. Burns, and R. Davis, "Response-time analysis for mixed criticality systems," in *Real-Time Systems Symposium (RTSS), 32nd IEEE International*, Nov 2011.

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

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