Partitioned Mixed-Criticality Scheduling on Multiprocessor Platforms

Chuancai Gu[1], Nan Guan[1,2], Qingxu Deng[1] and Wang Yi[1,2]
[1]Northeastern University, China
[2]Uppsala University, Sweden

***Abstract*—Scheduling mixed-criticality systems that integrate multiple functionalities with different criticality levels into a shared platform appears to be a challenging problem, even on single-processor platforms. Multi-core processors are more and more widely used in embedded systems, which provide great computing capacities for such mixed-criticality systems. In this paper, we propose a partitioned scheduling algorithm MPVD to extend the state-of-the-art single-processor mixed-criticality scheduling algorithm EY to multiprocessor platforms. The key idea of MPVD is to evenly allocate tasks with different criticality levels to different processors, in order to better explore the asymmetry between different criticality levels and improve the system schedulability. Then we propose two enhancements to further improve the schedulability of MPVD. Experiments with randomly generated task sets show significant performance improvement of our proposed approach over existing algorithms.**

I. Introduction

Multi-core processors are more and more widely used in modern real-time embedded systems, which provide great computing capacities to integrate multiple functionalities with different criticality levels into a shared hardware platform. Such mixed-criticality systems bring significant challenges to the design of real-time systems.

Vestal [14] formalized the single-processor mixed-criticality scheduling problem. Traditional real-time scheduling techniques, such as EDF, may lead to poor schedulability when applied to mixed-criticality systems. Researchers proposed different techniques to better explore the asymmetry between different criticality levels and improve the schedulability [1], [4], [7], [9], [11]. Recently, Ekberg and Yi [8] introduced a scheduling algorithm, called EY in this paper, which exhibits very good schedulability. EY is based on the idea of EDF-VD (EDF with virtual deadlines) proposed by Baruah et al. [3], which balances the schedulability on different criticality levels by tuning the *virtual deadlines* and has been extended to multiprocessors [2]. In Section II-B we will give a brief introduction to EY. The multiprocessor scheduling algorithm proposed in this paper is based on a variant of EY.

Traditional (single-criticality) multiprocessor scheduling algorithms are usually categorized into two paradigms [6]: *global scheduling*, in which each task can execute on any available processor at run-time, and *partitioned scheduling,* in which each tasks is assigned to a processor beforehand, and at run-time each task only executes on this particular processor.

978-3-9815370-2-4/DATE14/©2014 EDAA

Recent work in multiprocessor scheduling has shown that partitioned scheduling typically has better schedulability than global scheduling for hard real-time systems [5]. Therefore, in this paper we use the partitioned approach to schedule mixed-criticality systems.

The choice of the packing (task allocation) strategy greatly affects the performance of partitioned scheduling. In partitioned scheduling of traditional real-time systems, the first-fit (FF) packing strategy typically performs better than others. Therefore, it sounds a natural extension to apply FF to partitioned scheduling of mixed-criticality systems. For example, [10] evaluates several combinations of different packing strategies and task sorting orders, and conclude that the combination of FF and criticality-decreasing task ordering performs the best among all the evaluated solutions. However, as shown in this paper, FF actually cannot very well explore the asymmetry between different criticality levels, and thus leads to unsatisfactory schedulability performance.

In this paper, we propose a partitioned scheduling algorithm MPVD, which treats high- and low-criticality tasks differently. MPVD first uses worst-fit (WF) packing strategy to allocate high-criticality tasks, then uses FF to allocate low-criticality tasks. By such a hybrid packing strategy, high-criticality tasks are evenly allocated to different processors, which gives us a better chance to use the EY algorithm to balance the workload on different criticality levels and improve system schedulability.

The performance of MPVD may degrade when the number of processors is large. To address this problem, we propose two enhancements to MPVD to further improve the schedulability. Firstly, we consider the situation that heavy low-criticality tasks may fail to be allocated because the free capacity of each individual processor is not enough, although there still remains much total available capacity on all processors. To solve this problem, we preserve resource for heavy low-criticality tasks before allocating high-criticality tasks. Secondly, we propose an optimized virtual deadlines tuning algorithm in EY to improve the performance of our partitioned algorithm.

Experiments with randomly generated task systems are conducted to evaluate the performance of our proposed algorithms. Experiment results show that our new algorithm MPVD, especially with the two enhancements, can significantly improve the system schedulability comparing with existing multiprocessor mixed-criticality scheduling algorithms.

II. Preliminaries

A. MC System Model and Notations

We adopt the same mixed-criticality (MC) task model as in [8]. Similar with the traditional sporadic task model, we define an MC sporadic system π as a finite set of independent MC sporadic tasks, each of which may generate a potentially infinite sequence of MC jobs $\langle J_i^1, J_i^2, \cdots \rangle$. Each MC task is characterized by a 4-tuple $\tau_i = (T_i, D_i, \zeta_i, C_i)$, where

- $T_i \in R^+$ is the minimal inter-arrival separation.
- $D_i \in R^+$ is the relative deadline.
- $\zeta_i \in \{\text{LO}, \text{HI}\}$ is the criticality level of task τ_i, where LO indicates the low criticality and HI indicates the high criticality.
- $C_i : \{\text{LO}, \text{HI}\} \rightarrow R^+$ is the worst-case execution time (WCET) function, which specifies the WCET of the task at each criticality level ℓ: $C_i(\text{LO})$ denotes the estimated WCET of task τ_i under the low-criticality level, and $C_i(\text{HI})$ denotes the more rigidly estimated WCET under high-criticality level, i.e., $C_i(\text{LO}) \leq C_i(\text{HI})$.

Note that the relative deadlines can be arbitrary positive real numbers without any restrictions regarding the task periods, i.e., D_i can be larger than, smaller than or equal to T_i.

To describe the workload on different criticality levels, we use U_i^{LO} and U_i^{HI} to denote the low- and high-criticality utilization of task τ_i respectively:

$$U_i^{\text{LO}} \triangleq C_i(\text{LO})/T_i \text{ and } U_i^{\text{HI}} \triangleq C_i(\text{HI})/T_i.$$

We use $\text{LO}(\pi) \triangleq \{\tau_i \in \pi | \zeta_i = \text{LO}\}$ to denote the low-criticality task subset of π, and use $\text{HI}(\pi) \triangleq \{\tau_i \in \pi | \zeta_i = \text{HI}\}$ to denote the high-criticality subset. We define the total utilization $U_{\text{LO}}(\pi)$ and $U_{\text{HI}}(\pi)$ as follows:

$$U_{\text{LO}}(\pi) \triangleq \sum_{\tau_i \in \pi} U_i^{\text{LO}} \text{ and } U_{\text{HI}}(\pi) \triangleq \sum_{\tau_i \in \text{HI}(\pi)} U_i^{\text{HI}}.$$

The semantic of the MC system is as follows. When the system starts, it runs in the low-criticality mode and each task $\tau_i \in \pi$ could release jobs in sequence as long as no released jobs execute beyond their respective run-time deadlines. Otherwise, once any job has executed for its WCET under low-criticality level without signaling its completion of execution, the system will switch to high-criticality mode immediately. In order to guarantee the high-criticality jobs still meet their deadlines even though they execute for up to the more strict WCET estimated under such high-criticality mode, all of the low-criticality jobs need not meet deadlines any more, and will be discarded totally after the mode switching. Therefore, the low-criticality jobs will make no interference with the scheduling of high-criticality jobs.

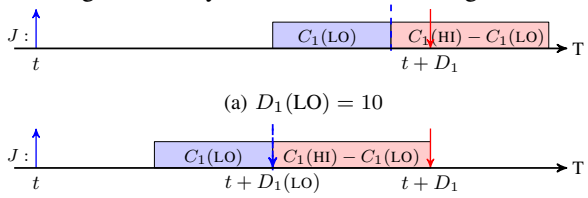
To schedule such an MC system successfully, all of the jobs released from high-criticality tasks must always meet their deadlines both in low-criticality and high-criticality modes, but the jobs released from low-criticality tasks only need to meet deadlines in low-criticality mode.

B. The EY Approach

The partitioned multiprocessor scheduling algorithm proposed in this paper uses EY [8] to test and tune the schedulability on each individual processor. The main idea of EY can be summarized as follows. When the system runs in the low criticality mode, each high-criticality task τ_i uses a virtual deadline $D_i(\text{LO})$ (shorter than its original deadline D_i) in the EDF scheduling decision. This helps the high-criticality task to finish its low-criticality workload earlier, and thus have a better chance to finish the remaining high-criticality workload before its real deadline if the criticality mode-switch occurs. Consider the example task set illustrated as below:

Task	T_i	D_i	ζ_i	$C_i(\text{LO})$	$C_i(\text{HI})$	U_i^{LO}	U_i^{HI}
τ_1	10	10	HI	3	7	0.3	0.7
τ_2	5	5	LO	3	3	0.6	0.6

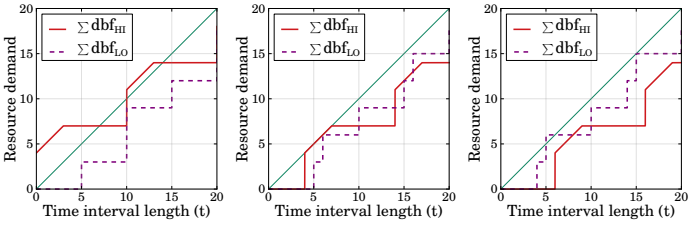
Suppose the high-criticality task τ_1 releases a job J at t in the low-criticality mode. Under EDF scheduling, in the worst-case τ_1 finishes it low-criticality workload one time unit before its absolute deadline $t + D_1$, as shown in Figure 1a. If the execution of τ_1 overflows and the system switches to the high-criticality mode, it is impossible to finish the remaining workload $C_1(\text{HI}) - C_1(\text{LO})$ before its absolute deadline $t + D_1$, as shown in Figure 1a. If we set τ_1's virtual deadline $D_1(\text{LO}) = 6$, we can guarantee that τ_1 finishes its low-criticality workload no later than $t + D_1(\text{LO})$. In this case, there is enough time for τ_1 to finish its high-criticality workload before its real deadline $t + D_1$ after the system runs into the high-criticality mode, as shown in Figure 1b.



(a) $D_1(\text{LO}) = 10$

(b) $D_1(\text{LO}) = 6$

Figure 1. Impact of different virtual deadlines.



(a) $D_1(\text{LO})$=10 (b) $D_1(\text{LO})$=6 (c) $D_1(\text{LO})$=4

Figure 2. Impact on demand of different virtual deadlines.

Using shorter virtual deadlines is beneficial to the schedulability in the high-criticality mode, but at the same time the system is more difficult to be scheduled in the low-criticality mode (since each high-criticality task must meet a *shorter* virtual deadline). In the above example, if we set $D_1(\text{LO}) = 4$, the system is not schedulable in the low-criticality mode. In EY, this phenomenon is formally captured by the high-criticality demand bound function $\text{dbf}_{\text{HI}}(\tau_i, \Delta)$ and low-criticality demand bound function $\text{dbf}_{\text{LO}}(\tau_i, \Delta)$. The system is guaranteed to be schedulable in the low-criticality mode if it holds

$$\forall t \geq 0 : \sum_{\tau_i \in \pi} \text{dbf}_{\text{LO}}(\tau_i, t) \leq t. \quad (1)$$

The system is guaranteed to be schedulable in the high-criticality mode if it holds

$$\forall t \geq 0 : \sum_{\tau_i \in \pi \wedge \zeta_i = \text{HI}} \text{dbf}_{\text{HI}}(\tau_i, t) \leq t. \quad (2)$$

Due to space limit, we do not recite the computation of $\text{dbf}_{\text{HI}}(\tau_i, \Delta)$ and $\text{dbf}_{\text{LO}}(\tau_i, \Delta)$, but refer interested readers to [8] for details.

For each high-criticality task τ_i, if its virtual deadline $D_i(\text{LO})$ is decreased, the high-criticality demand bound function dbf_{HI} decreases while the low-criticality demand bound function dbf_{LO} increases. Figure 2 shows the demand bound functions when τ_1 uses different virtual deadlines. If the virtual deadline is too short, the system is non-schedulable in the low-criticality mode as shown in Figure 2a. If the virtual deadline is too long, the system is non-schedulable in the high-criticality mode, as shown in Figure 2c. The system is schedulable in both modes with a proper $D_i(\text{LO})$ as shown in Figure 2b.

The virtual deadlines of high-criticality tasks are used as the nob to tune the schedulability in different criticality modes. However, it leads to a larger search space to find the optimal virtual deadline configuration. EY uses an effective heuristic to tune the virtual deadlines. It tunes each $D_i(\text{LO})$ from D_i to $C_i(\text{LO})$ monotonically to control the overall time complexity. At each tuning point, EY decreases certain $D_i(\text{LO})$ to $D_i(\text{LO}) - 1$, and EY always greedily selects the one which gets the maximal value of the expression:

$$\text{dbf}_{\text{HI}}(\tau_k, D_k(\text{LO})) - \text{dbf}_{\text{HI}}(\tau_k, D_k(\text{LO}) - 1). \quad (3)$$

III. MPVD Partitioning Algorithm

Partitioned scheduling for EDF-VD has been studied in [3]. And in this section we will discuss the parti- tioning schemes for the higher performance algorithm EY.

A. Motivating the Hybrid Packing Strategy

Before introducing our proposed partitioning algorithm with a hybrid packing strategy, we first discuss the drawback of extending EY to multiprocessor scheduling with the FF packing strategy and motivate the hybrid packing strategy used in this paper. [10] presented a partitioning strategy using FF packing and decreasing-criticality task ordering (called FFDC for short). FFDC packs as many high-criticality tasks as possible to one processor until it is full, then picks the next processor to pack the remaining high-criticality tasks. After all high-criticality tasks are successfully allocated, FFDC continues to pack low-criticality tasks also with the FF strategy. FFDC has the best performance among all the partitioning algorithms evaluated in [10]. Unfortunately, the combination of the FFDC strategy and the EY approach does not yield satisfactory performance, the reason of which is as follows. The strength of EY is the capability of balancing the schedulability between high- and low-criticality levels. In FFDC, the allocation of high-criticality tasks is typically unbalanced, so the room for EY to tune the virtual deadlines is relatively smaller. Consider the following task set to be scheduled on two processors:

Task	T_i	D_i	ζ_i	$C_i(\text{LO})$	$C_i(\text{HI})$	U_i^{LO}	U_i^{HI}
τ_1	10	10	HI	2	3	0.2	0.3
τ_2	10	10	HI	2	3	0.2	0.3
τ_3	10	10	LO	7	7	0.7	0.7
τ_4	10	10	LO	7	7	0.7	0.7

By FFDC, the two high-criticality tasks τ_1 and τ_2 will be allocated to processor P_1, after which we allocate low-criticality tasks. However, the low-criticality utilization sum of τ_1 and τ_2 is already 0.4, so τ_3 or τ_4 cannot be allocated to P_1, no matter how do we tune the virtual deadlines of τ_1 and τ_2 (since 0.4+0.7>1). On the other hand, it is infeasible to allocate both τ_3 and τ_4 to P_2 since 0.7+0.7>1. So the partitioning of this task set is failed with the FF packing strategy.

If we use worst-fit (WF) packing strategy to partition this task set, we can allocate τ_1 and τ_3 to P_1, and allocate τ_2 and τ_4 to P_2. By applying the virtual deadline tuning of EY, we see that tasks on both processors are schedulable if the virtual deadlines of τ_1 and τ_2 are set to be 6. In summary, WF distributes high-criticality tasks to different processors more evenly than FF, which gives us more room to utilize the virtual deadline tuning in EY to improve the schedulability.

In the allocation of high-criticality tasks, the virtual deadlines of high-criticality tasks are tuned to guarantee that they are schedulable in the high-criticality mode. As soon as all high-criticality tasks are allocated, the virtual deadlines of all high-criticality tasks are fixed. So when we start to allocate low-criticality tasks, the remaining capacity on each processor is fixed. So the partitioning of low-criticality tasks is similar to the traditional workload partitioning problem (without multiple criticality levels). Since FF has proven to be the best packing strategy for such a problem, we shall use FF as the packing strategy for low-criticality tasks.

B. MPVD

Now we introduce our new partitioning algorithm MPVD (Mixed-criticality Partitioning with Virtual Deadlines) in detail. We first introduce the notation of (high- or low-criticality) *remaining utilization* of a processor, which denotes the difference between 1 and the total (high- or low-criticality) utilization of tasks that have been allocated to this processor. For example, if only one task with utilization 0.3 is allocated to processor P_1, then the remaining utilization of P_1, is 1-0.3=0.7. We use $U_{\text{HI}}(P_1)$ and $U_{\text{LO}}(P_1)$ to denote the high- and low-criticality remaining utilization of processor P_1.

MPVD works in the following three steps:

1) Allocate high-criticality tasks to processors by the WF packing strategy, i.e., always select the processor P_x with the maximal high-criticality remaining utilization $U_{\text{HI}}(P_x)$ to allocate tasks. The tasks are sorted in the decreasing order of their high-criticality utilization U_{HI}.
2) Tune the virtual deadlines of high-criticality tasks allocated to each processor by the tuning algorithm in EY

[8] to meet the dbf_{HI} constraint (2), i.e., to guarantee that the high-criticality task subset on each processor is schedulable in the high-criticality mode. If the tuning algorithm in EY fails on any processor, the partitioning algorithm MPVD fails.

3) Allocate low-criticality tasks to processors by the FF packing strategy. The tasks are sorted in the decreasing order of their low-criticality utilization U_{LO}. In this step, we use the dbf_{LO} constraint (1) to check whether an unallocated low-criticality task can be assigned to a candidate processor, i.e., whether the tasks already assigned to this processor and the current task can meet deadlines in the low-criticality mode if they are scheduled together on this processor.

Note that the first step of MPVD only allocates high-criticality tasks to processors, but does not guarantee anything about the schedulability. After all high-criticality tasks are allocated, we tune their virtual deadlines and decide whether they are schedulable in the high-criticality mode. The virtual deadline tuning algorithm in EY is of pseudo-polynomial time complexity, and in practice rather time consuming. It will be extremely inefficient if the virtual deadline tuning algorithm has to be invoked iteratively. MPVD only needs to invoke the virtual deadline tuning algorithm once on each processor.

At run-time, the tasks allocated to each processor are scheduled in the same way as EY. When the system is in the low-criticality mode, tasks are scheduled by EDF and each high-criticality task uses its virtual deadline in the EDF scheduling decision. When the system runs into the high-criticality mode, all the low-criticality tasks are abandoned immediately, and all high-criticality tasks use their original deadlines in the EDF scheduling decision.

IV. Enhancements to MPVD

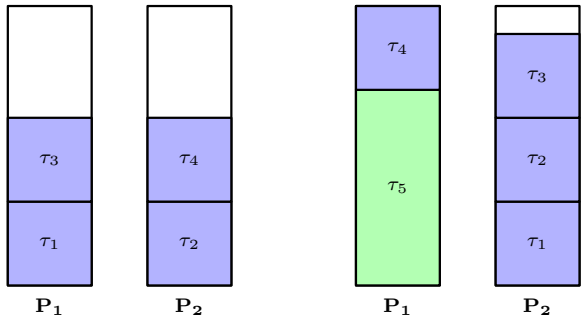
The MPVD partitioning algorithm introduced in last section may still lead to unsatisfactory performance under certain circumstances. In this section, we enhance MPVD by optimized task allocation strategies and virtual deadline tuning algorithms to further improve the schedulability.

A. Heavy Low-Criticality Task Aware Partitioning

One of the most important issues in partitioned scheduling is to avoid the situation that a heavy (high-utilization) task cannot be allocated while the processors still have relatively large available capacities. The decreasing utilization task ordering is an effective way to avoid the above situation. Therefore, we choose to use decreasing utilization to order tasks in the allocation of high- and low-criticality tasks (step 1 and 3) respectively. However, since MPVD first allocates all high-criticality tasks before allocating any low-criticality task, it may happen the situation that some heavy low-criticality tasks cannot be allocated to any individual processor although there remains much total remaining capacity on all processors. For example, consider the task set in Table I to be partitioned on two processors. By MPVD, each processor will be assigned 2 high-criticality tasks as illustrated in Figure 3a. Therefore, the low-criticality task τ_5 cannot be allocated to any of the two processors no matter how do we tune the virtual deadlines of the high-criticality tasks (since 0.2+0.2+0.7>1).

Table I
An Example Task Set

Task	T_i	D_i	ζ_i	$C_i(\text{LO})$	$C_i(\text{HI})$	U_i^{LO}	U_i^{HI}
τ_1	10	10	HI	2	3	0.2	0.3
τ_2	10	10	HI	2	3	0.2	0.3
τ_3	10	10	HI	2	3	0.2	0.3
τ_4	10	10	HI	2	3	0.2	0.3
τ_5	10	10	LO	7	7	0.7	0.7



(a) MPVD (b) MPVD-HA

Figure 3. Task allocation results in different approaches

To address this problem, we enhance MPVD by a heavy low-criticality task aware policy. The main idea is to make a tradeoff between high-criticality workload balancing and the schedulability of low-criticality tasks by preserving moderate free resources for *heavy low-criticality tasks*, which are defined as follows:

Definition IV.1 (Heavy Low-Criticality Task)**.** Given a mixed-criticality system π, a low criticality task τ_k is heavy if

$$U_k^{\text{LO}} > 1 - \frac{U_{\text{LO}}(\pi_{\text{HI}})}{m}. \tag{4}$$

The enhanced heavy low-criticality task aware MPVD partitioning algorithm, called MPVD-HA for short, differs from MPVD by adding one step before starting MPVD:

- Select all of the heavy low-criticality tasks satisfying (4) and relate each heavy low-criticality task to one processor. If a heavy low-criticality task τ_b is related to processor P_x, then we set the initial high-criticality remaining utilization of P_x as
$$U_{\text{HI}}(P_x) \leftarrow 1 - U_b^{\text{LO}}. \tag{5}$$

After that, the partitioning algorithm follows the same procedure as MPVD. It is easy to see that if the number of heavy low-criticality tasks is greater than the processor number m, then the task set cannot be successfully scheduled by any algorithm (since in that case the total low-criticality utilization of the task set is larger than m).

In the example of Table I, since $1 - U_{\text{LO}}(\pi_{\text{HI}})/2 = 0.6 < U_5^{\text{LO}} = 0.7$, τ_5 is a heavy low-criticality task and is related to P_1. So $U_{\text{HI}}(P_1)$ is set to $1 - 0.7 = 0.3$. After that, following the MPVD partitioning algorithm, τ_1, τ_2, τ_3 will be allocated to P_2 and τ_4 will be allocated to P_1 as illustrated in Figure 3b. Consequently, P_1 reserves enough free capacity to adopt the heavy low-criticality task τ_5 in step 3 of MPVD, and the task set is successfully partitioned.

B. Improving the Virtual Deadline Tuning Algorithm

A major reason for EY to achieve extremely good schedulability is that it employs a very effective heuristic algorithm to tune the virtual deadlines of high-criticality tasks. However, the tuning algorithm in EY is not completely suitable to MPVD. Recall that when the tuning algorithm is invoked in step 2 of MPVD, only high-criticality tasks have been allocated and no information of low-criticality is known at that moment. If one uses (3) to choose the task to shrink its virtual deadline, it may lead to very fast increase of dbf_{LO}, which is harmful to the schedulability in the low-criticality mode. To address this problem, we use the *balance factor* as the metric to choose tasks for virtual deadline tuning.

Definition IV.2 (Balance Factor)**.** For a high criticality task τ_k and time interval size t, the balance factor is defined as

$$\phi(\tau_k, t) = \frac{\text{dbf}_{\text{HI}}(\tau_k, D_k(\text{LO})) - \text{dbf}_{\text{HI}}(\tau_k, D_k(\text{LO}) - 1)}{C_k(\text{LO})/(D_k(\text{LO}) - 1) - C_k(\text{LO})/D_k(\text{LO})}. \quad (6)$$

The denominator $C_k(\text{LO})/(D_k(\text{LO})-1) - C_k(\text{LO})/D_k(\text{LO})$, intuitively, represents the *cost* of the schedulability in the low-criticality mode when the virtual deadline of a high-criticality task is decreased by 1. Our new tuning algorithm always chooses the task with the maximal $\phi(\tau_k, t)$ value, which improves the schedulability in the high-criticality mode at the minimal *cost* of the schedulability in the low-criticality mode.

V. Experimental Evaluation

We conduct experiments with randomly generated task systems to compare the performance, in terms of acceptance ratio, of the algorithms proposed in this paper and previous multiprocessor scheduling algorithms for mixed-criticality systems. The previous algorithms evaluated in our experiments include both global and partitioned scheduling algorithms. Our experiments show that the performance of partitioned scheduling is significantly better than global scheduling (the algorithms in [12], [13]), which is similar to the multiprocessor scheduling of traditional real-time task systems. Therefore, in this section we only report the comparison between the algorithms proposed in this paper and other partitioned scheduling algorithms. The evaluated algorithms include:

- MPVD: the partitioning algorithm in Section III.
- MPVD-HA: MPVD enhanced by the heavy low-criticality task aware allocation policy in Section IV-A.
- MPVD-HA-BF: MPVD-HA further enhanced by the optimized virtual deadline tuning in Section IV-B.
- DC-Audsley: the partitioned scheduling algorithm based on the Audsley approach in [10].
- EY-FF: the straightforward extension of EY to partitioned scheduling with the FF packing strategy as discussed in the beginning of Section III.
- MC-Partition: the partitioned scheduling algorithm based on EDF-VD approach in [2].

A. Random Task Set Generation

Our experiments use dual-criticality implicit deadlines sporadic task model on an m identical unit speed multiprocessor platform. We use a similar approach as in [8] to generate random mixed-criticality task sets. The random task is generated by four tunable parameters: the probability P_{HI} of being of high-criticality, the maximal ratio R_{HI} between high- and low-criticality execution time of each high-criticality task, the maximal low-criticality execution time $C_{\text{LO}}^{\max}$ and the maximal period $T^{\max}$. Each new task τ_i is generated as follows:

- $\zeta_i = \text{HI}$ with probability P_{HI}, otherwise $\zeta_i = \text{LO}$.
- $C_i(\text{LO})$ is a randomly generated integer uniformly drawn from $[1, C_{\text{LO}}^{\max}]$.
- $C_i(\text{HI})$ is a randomly generated integer uniformly drawn from $[C(\text{LO}), R_{HI} \cdot C_i(\text{LO})]$ if $\zeta_i = \text{HI}$. Otherwise, $C_i(\text{HI}) = C_i(\text{LO})$.
- T_i is a randomly generated integer uniformly drawn from $[C_i(\text{HI}), T^{\max}]$.
- $D_i = T_i$ because of the implicit deadline constraint.

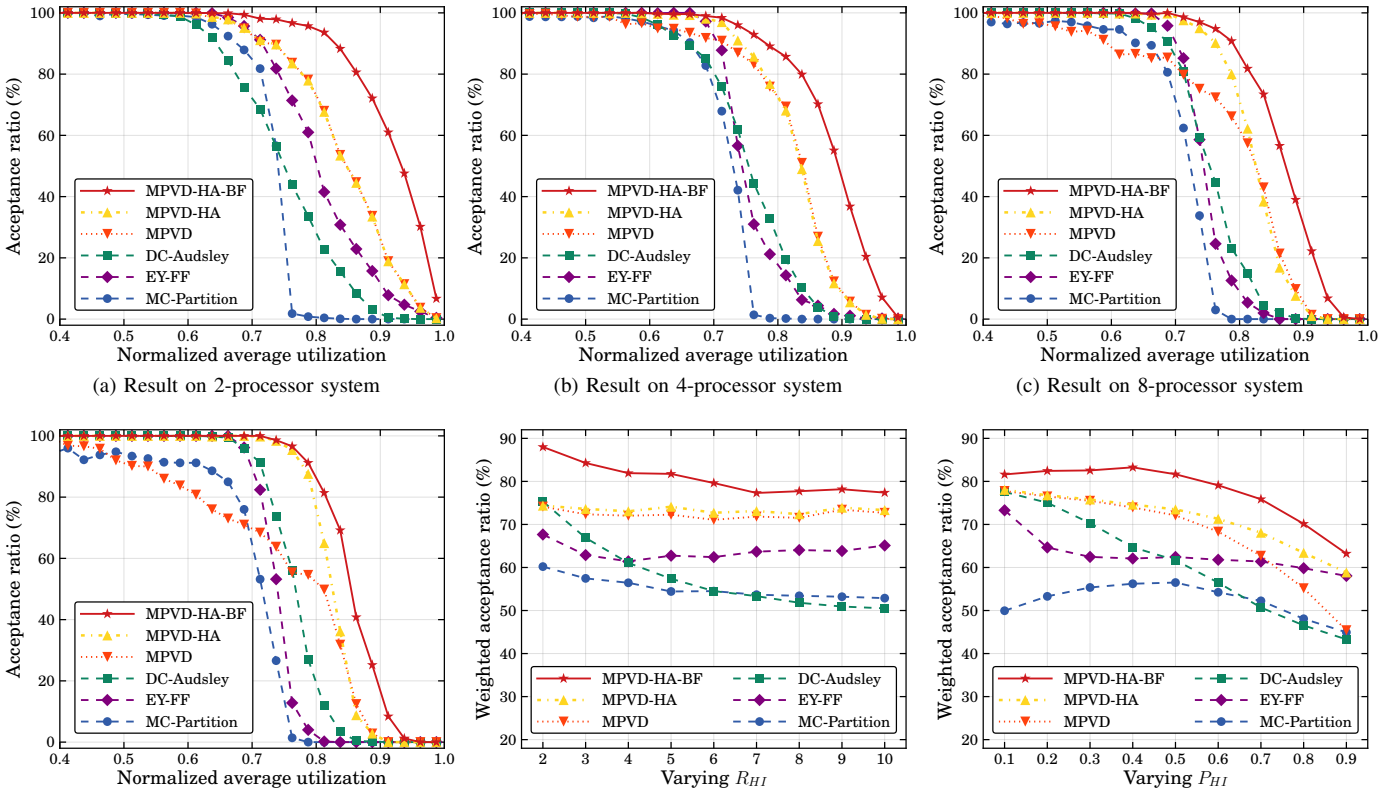
Each random task set is generated with a target *normalized average utilization* U^* with a acceptable range of errors: $U^*_{\min} = U^* - 0.005 \cdot m$ and $U^*_{\max} = U^* + 0.005 \cdot m$.

A random task set is generated by starting with an empty task set $\pi = \emptyset$, to which random tasks are successively added. We generate a new random task and append it to π continually as long as $\min(U_{\text{LO}}(\pi), U_{\text{HI}}(\pi)) < U^*_{\min}$. If a task is added such that $\max(U_{\text{LO}}(\pi), U_{\text{HI}}(\pi)) > U^*_{\min}$, we discard the whole task set and start with a new empty task set. If a task is added such that $U^*_{\min} \leq \min(U_{\text{LO}}(\pi), U_{\text{HI}}(\pi))$ and $\max(U_{\text{LO}}(\pi), U_{\text{HI}}(\pi)) \leq U^*_{\max}$, the task set is finished, unless all tasks in π have the same criticality level or $U_{\text{LO}}(\pi), U_{\text{HI}}(\pi) > 0.99$, in which case the task set is instead discarded.

B. Results

The parameter configuration of the experiments in Figure 4 is as follows: $P_{\text{HI}} = 0.5$, $R_{\text{HI}} = 4$, $C_{\text{LO}}^{\max} = 10$ and $T^{\max} = 200$. Figure 4a to Figure 4d show the *acceptance ratio*, i.e., the portion of schedulable task sets out of all the random task sets generated at this utilization range, as a function of *normalized average utilization* with different processor numbers. Each point in any figure includes at least 2000 randomly generated task sets.

Figure 4e and Figure 4f illustrate the effect of varying random parameters R_{HI} and P_{HI} through the *weighted acceptance ratio* function of the varied parameters. For each certain varied parameter, we compute 20 acceptance ratios $A(U_i)$ with different U_i, and let $W(U_i) > 0$ be the weighted factor for target utilization U_i. The weighted acceptance ratio is denoted as $\sum_{\forall U_i} W(U_i) \cdot A(U_i) / \sum_{\forall U_i} W(U_i)$. Each point in the two figures includes at least 20000 randomly generated task sets.



(a) Result on 2-processor system (b) Result on 4-processor system (c) Result on 8-processor system

(d) Result on 16-processor system (e) Varying R_{HI} on 4-processor system (f) Varying P_{HI} on 4-processor system

Figure 4. Experiment results ($P_{\mathrm{HI}} = 0.5, R_{\mathrm{HI}} = 4, C_{\mathrm{LO}}^{\max} = 10 and T^{\max} = 200$)

Figure 4 shows that MPVD performs better than DC-Partition, DC-Audsley and EY-FF with fewer number of processors. However, the performance of MPVD degrades as the number of processors becomes greater. With 16 processors, MPVD may fail to partition task sets with rather low total normalized average utilization. This is mainly because of the problem we pointed out at the beginning of Section III. The enhanced algorithm MPVD-HA can solve this problem and thus steadily exhibits better performance than DC-Audsley and EY-FF. Moreover, with the optimized virtual deadline tuning algorithm based on the balance factor, the acceptance ratio of MPVD-HA-BF is further improved.

VI. Conclusions

In this paper we studied the partitioned scheduling algorithm for mixed-criticality systems on multiprocessors. We proposed new partitioned scheduling algorithms based on a hybrid task packing strategy and the state-of-the-art single-processor mixed-criticality scheduling algorithm EY [8]. Experiments with randomly generated task sets showed significant improvement of our proposed approach over existing algorithms.

Acknowledgements

Supported in part by China Fundamental Research Funds for the Central Universities under grant N100204001 and N110804003; and China Research Fund for the Doctoral Program of Higher Education under grant 20110042110021; and China Science Fund for Youths under grant 61300022.

References

[1] S. Baruah, H. Li, and L. Stougie, "Towards the design of certifiable mixed-criticality systems," in *RTAS*, 2010.

[2] S. Baruah, B. Chattopadhyay, H. Li, and I. Shin, "Mixed-criticality scheduling on multiprocessors," *Real-Time Systems*, 2013.

[3] S. K. Baruah, V. Bonifaci, G. D`Angelo, A. Marchetti-Spaccamela, S. Van Der Ster, and L. Stougie, "Mixed-criticality scheduling of sporadic task systems," in *Algorithms-ESA*, 2011.

[4] S. Baruah, A. Burns, and R. Davis, "Response-time analysis for mixed criticality systems," in *RTSS*, 2011.

[5] A. Bastoni, B. B. Brandenburg, and J. H. Anderson, "An empirical comparison of global, partitioned, and clustered multiprocessor edf schedulers," in *RTSS*, 2010.

[6] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah, "A categorization of real-time multiprocessor scheduling problems and algorithms," *Proceedings of the IEEE*, 2004.

[7] D. de Niz, K. Lakshmanan, and R. Rajkumar, "On the scheduling of mixed-criticality real-time task sets," in *RTSS*, 2009.

[8] P. Ekberg and W. Yi, "Outstanding paper award: Bounding and shaping the demand of mixed-criticality sporadic tasks," in *ECRTS*, 2012.

[9] N. Guan, P. Ekberg, M. Stigge, and W. Yi, "Effective and efficient scheduling of certifiable mixed-criticality sporadic task systems," in *RTSS*, 2011.

[10] O. R. Kelly, H. Aydin, and B. Zhao, "On partitioned scheduling of fixed-priority mixed-criticality task sets," in *TrustCom*, 2011.

[11] H. Li and S. Baruah, "An algorithm for scheduling certifiable mixed-criticality sporadic task systems," in *RTSS*, 2010.

[12] ——, "Outstanding paper award: Global mixed-criticality scheduling on multiprocessors," in *ECRTS*, 2012.

[13] R. Pathan, "Schedulability analysis of mixed-criticality systems on multiprocessors," in *ECRTS*, 2012.

[14] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *RTSS*, 2007.