Graceful Degradation of Low-Criticality Tasks in Multiprocessor Dual-Criticality Systems

Lin Huang*, I-Hong Hou*, Sachin S. Sapatnekar† and Jiang Hu*
*Department of Electrical and Computer Engineering, Texas A&M University
†Department of Electrical and Computer Engineering, University of Minnesota
liushui0820@tamu.edu, ihou@tamu.edu, sachin@umn.edu and jianghu@tamu.edu

***Abstract*—According to the conventional mixed-criticality (MC) system model, low-criticality tasks are completely discarded in high-criticality system mode. Allowing such loss of low-criticality tasks is controversial and not obviously necessary. We study how to achieve graceful degradation of low-criticality tasks by continuing their executions with imprecise computing or even precise computing if there is sufficient utilization slack. Schedulability conditions under this Variable-Precision Mixed-Criticality (VPMC) system model are investigated for partitioned scheduling and fpEDF-VD scheduling. It is found that the two scheduling methods in VMPC retain the same speedup factors as in conventional MC systems. We develop a precision optimization approach that maximizes precise computing of low-criticality tasks through 0-1 knapsack formulation. Experiments are performed through both software simulations and Linux prototyping with consideration of overhead. The results show that schedulability degradation caused by continuing low-criticality task execution is often very small. The proposed precision optimization can largely reduce computing errors compared to constantly executing low-criticality tasks with imprecise computing in high-criticality mode. The prototyping results indicate that partitioned scheduling in VPMC outperforms the latest work based on fluid model.**

I. Introduction

Mixed-criticality system scheduling has attracted a great deal of research attention in the past 10 years [1]–[6]. Its key elements can be described through a dual-criticality system, which starts with low-criticality mode. Once a high-criticality job executes longer than its worst case execution time (WCET), the system is switched to high-criticality mode, where high-criticality tasks are treated with increased WCET and low-criticality tasks are completely discarded. Such handling of low-criticality tasks is controversial [6], [7]. One approach to addressing this controversy is an elastic scheme [8], [9], where low-criticality tasks are continued with extended period in high-criticality mode. Another method is to reduce the priorities of low-criticality tasks [9]. However, it is noticed [7] that task period and priority are usually functional requirements and cannot be easily changed.

A more viable approach to avoiding complete loss of low-criticality tasks is making use of imprecise computing [9]–[12]. Each low-criticality task can alternatively be executed with imprecise computing, which causes inaccuracy in computing results but costs relatively short execution time. When a system is in high-criticality mode, low-criticality tasks can continue to execute with imprecise computing instead of being dropped. Such approach allows graceful degradation of low-criticality tasks in high-criticality mode. This model is called Imprecise Mixed-Criticality (IMC) system [11]. The concept of imprecise computing appeared more than two decades ago [13]. Recently, its realization has made a lot of progress [14] and its application in mixed-criticality systems becomes more practical.

The schedulability conditions for several uniprocessor scheduling algorithms in IMC are established [9]–[11]. When deadline constraints are not tight, there is usually processor utilization slack for low-criticality tasks to continue with even precise computing. Based on this observation, the work of [12] extends the fluid-based multiprocessor scheduling [15], [16] to maximize precise computing of low-criticality tasks in high-criticality mode. A mixed-criticality system with such treatment of low-criticality tasks can be called Variable-Precision Mixed-Criticality (VPMC) system. This is perhaps the only published literature on VPMC and the only published work considering graceful degradation of low-criticality tasks for multiprocessor scheduling. Although the fluid-based scheduling [15], [16] is optimal in theory, it cannot be implemented with its original form on hardware as it depends on an unrealistic assumption that each processor can be partitioned into arbitrary fractions. MC-DP-Fair [15] is a practically implementable scheduling algorithm with schedulability equivalent to the fluid-based scheduling. However, the work of [12] has little discussion on MC-DP-Fair or the implementation issue, and MC-DP-Fair can result in an excessive amount of context switchings and hence a considerable overhead.

In this work, we study VPMC systems for some well-known multiprocessor scheduling methods that have not been well considered for the graceful degradation option yet. These include partitioned scheduling, fpEDF-VD based global scheduling [4] and MC-DP-Fair scheduling. The schedulability conditions of these methods are extended for considering low-criticality task executions in high-criticality mode. We found that the speedup factors of partitioned scheduling and fpEDF-VD are the same as before when they are applied in VPMC. We develop a precision optimization technique that maximizes precise computing of low-criticality tasks in high-criticality mode. This optimization is through formulation of 0-1 knapsack problem, which is optimally solved by dynamic programming. The proposed techniques are evaluated with both software simulation and Linux prototyping with consideration of overhead on context switching, execution monitoring and mode changes. Since continuing low-criticality tasks stresses processors more

RTNS '18, October 10-12, 2018, Chasseneuil-du-Poitou, France
©2018 Copyright is held by the owner/author(s). Publication rights licensed to ACM.ACM ISBN 978-1-4503-6463-8/18/10 $15.00
https://doi.org/10.1145/3273905.3273909

than dropping them, it is important to validate in a realistic setup [17]. The results show that the schedulability degradation due to continuing low-criticality tasks is often very small. In addition, VPMC systems with our precision optimization lead to significantly smaller errors than IMC systems. Although the fluid-based VPMC system [12] has the best schedulability in theory, it is outperformed by partitioned scheduling when the overhead is considered.

The contributions of this work include:

- To the best of our knowledge, this is the first extensive study of IMC and VPMC systems on multiprocessors for several well-known scheduling approaches including partitioned, fpEDF-VD and MC-DP-Fair scheduling. The study covers schedulability analysis for these methods. We also show that the speedup factors of partitioned scheduling and fpEDF-VD scheduling are not changed in the VPMC model.
- An offline precision optimization technique is proposed to minimize errors from using imprecise computing by low-criticality tasks subject to schedulability constraints. This optimization can be optimally solved by dynamic programming.
- To the best of our knowledge, this is the first study of IMC, VPMC and their computing errors with consideration of overhead. This work includes the first prototyping implementation of VPMC in contrast to only software simulations in the related previous works [10]–[12].

II. Related Work

There are only a few studies on imprecise computing of low-criticality tasks in mixed-criticality systems. Most of them are built upon scheduling methods derived for the conventional MC model. One early work is [9], which is an extension of adaptive mixed-criticality scheduling [18]. Since this work also covers the other two approaches, reducing the priorities or increasing the periods of low-criticality tasks, its discussion on imprecise computing is restricted to response time analysis. Later, a mixed-criticality scheduling work dedicated to the imprecise computing model is introduced in [10]. This is a uniprocessor scheduling based on the fluid model [15], [16] and the algorithm is proved to have speedup factor of $\frac{4}{3}$. Another uniprocessor scheduling considering imprecise computing for low criticality tasks is [11], which is an extension to EDF-VD (Virtual Deadline) scheduling [2]. It derives a sufficient condition and speedup factor for allowing imprecise computing of low-criticality tasks under EDF-VD. All these works [9]–[11] execute low-criticality tasks with imprecise computing in high-criticality mode. By contrast, the latest work [12] allows some low-criticality tasks to be executed with full precision in high-criticality mode. It formulates an integer linear programming to decide which low-criticality tasks can continue with precise computing. Like [10], this work is also based on the fluid model and is applied with multiprocessors. The goal of [12] is very close to our work - maximizing precise computing of low-criticality tasks on multiprocessors. The key difference is that [12] is more focused on theoretical conditions while our work emphasizes more on practical realizations. Moreover, we study IMC and VPMC for other well-known multiprocessor scheduling techniques that have not been investigated before. Although fluid-based scheduling is theoretically very competitive, it cannot be directly implemented on hardware due to its restrictive assumption. Compared to [12], whose validation is by only software simulation, our work contains Linux prototyping validation that considers various overheads.

III. Imprecise MC and Variable-Precision MC System Model

The system contains a set of independent sporadic tasks $\mathcal{T} = \{\tau_1, \tau_2, ...\}$ with implicit deadlines. Each task τ_i is composed by an infinite sequence of jobs $\{J_i^1, J_i^2, ...\}$, and is characterized by $(T_i, \chi_i, C_i^{LO}, C_i^{HI})$ where T_i is the minimal time interval between two consecutive jobs of task τ_i, χ_i is the criticality level and C_i is the execution time. If job J_i^j is released at time r_i^j, its deadline is $r_i^j + T_i$. In this work, we consider dual-criticality system model, which is justified in [5]. Then, the task set $\mathcal{T}$ is composed by two disjoint subsets of low-criticality tasks $\mathcal{T}_{lo}$ and high-criticality tasks $\mathcal{T}_{hi}$, and $\chi_i \in \{lo, hi\}$ indicates if task τ_i has low or high-criticality. Also, C_i^{LO} and C_i^{HI} indicate the execution time at low and high system criticality mode, respectively, and are based on the Worst Case Execution Time (WCET).

A such system starts with low-criticality mode. Once the execution time of a high criticality job J_i^j exceeds its C_i^{LO}, the system is switched to high-criticality mode. If $\chi_i = hi$, $C_i^{HI} > C_i^{LO}$. In other words, high-criticality tasks are budgeted with more execution time in high-criticality mode. So far, the model has no fundamental difference from the conventional one [1], [6]. The key difference lies in the treatment of low-criticality tasks in high-criticality mode. We list related schemes below.

1) Conventional MC system [1]–[6]: all low-criticality tasks are discarded in high-criticality mode. As such, C_i^{HI} for a low-criticality task is equivalent to zero.
2) Imprecise MC (IMC) system [10], [11]: a low-criticality task τ_i has a precise computing realization with execution time $\hat{C}_i$ and an imprecise implementation with execution time $\tilde{C}_i < \hat{C}_i$. The works of [10], [11] let $C_i^{LO} = \hat{C}_i$ and $C_i^{HI} = \tilde{C}_i$, which is a non-zero constant. When the system is switched to high-criticality mode, if a low-criticality job has executed longer than C_i^{HI}, it would be aborted for this period, otherwise it would continue to execute till C_i^{HI}.
3) Variable-Precision MC (VPMC) system [12]: for a low-criticality task τ_i, $C_i^{LO} = \hat{C}_i$ and $C_i^{HI} \in \{\hat{C}_i, \tilde{C}_i\}$ corresponds to a decision variable. Since imprecise computing leads to errors, the objective of VPMC is to minimize computing errors, or maximize precise computing, for low-criticality tasks in high-criticality mode.

Please note VPMC and IMC follow the same schedulability condition for a scheduling method. When a schedulability condition is satisfied, there may be some processor utilization slack in high-criticality mode. IMC ignores the slack and applies imprecise computing for all low-criticality tasks. In contrast, VPMC attempts to utilize the slack in exchange for reducing imprecise computing and associated errors.

For each task τ_i, its utilizations in low and high-criticality mode are defined as

$$U_i^{LO} = \frac{C_i^{LO}}{T_i}, \text{ and } U_i^{HI} = \frac{C_i^{HI}}{T_i},$$

respectively. The total utilizations of all low-criticality tasks are

$$U_{lo}^{LO} = \sum_{\chi_i = lo} U_i^{LO} \text{ and } U_{lo}^{HI} = \sum_{\chi_i = lo} U_i^{HI}.$$

The total utilizations for high-criticality tasks are defined as

$$U_{hi}^{LO} = \sum_{\chi_i=hi} U_i^{LO} \text{ and } U_{hi}^{HI} = \sum_{\chi_i=hi} U_i^{HI}.$$

Given m identical processors, a conventional scheduling is to decide when to execute each job on which processor. We consider preemptive scheduling, where a low-priority job can be preempted by a high-priority job during its execution.

IV. Background

In this section, we review some existing knowledge, upon which our work is built.

A. Imprecise/Approximate Computing

Imprecise computing, which is also known as approximate computing, is to intentionally allow small or occasional computing errors such that either computing time or/and computing power dissipation is reduced. It can be carried out at circuit, architecture and algorithm levels for datapath or numerical computations. Circuit level approximation is mostly focused on imprecise arithmetic circuit designs [14], [19] or voltage overscaling [20]. The overscaling [20] and some dedicated designs [21] make computing accuracy runtime configurable. Algorithm level imprecise computing is not an explicitly named discipline although approximation widely exists in algorithm designs. As an example, algorithmic approximation can be realized by relaxing the convergence criterion in iterative algorithms. Overall, the substantial progress [14] on imprecise/approximate computing has made runtime computing precision reconfiguration feasible.

B. IMC System Scheduling on Uniprocessor

The imprecise mixed-criticality (IMC) scheduling on uniprocessor [11] is based on EDF-VD (Earliest Deadline First with Virtual Deadlines) [2]. In EDF-VD, the implicit deadlines of all high-criticality tasks are scaled by a factor x. That is, for each high-criticality task τ_i, its virtual implicit deadline is $\hat{T}_i = x \cdot T_i$. The sufficient schedulability condition in low-criticality mode is given by the following theorem.

Theorem 1. *(Theorem 1 in [2]) If the following condition is satisfied, sporadic task set $\mathcal{T}$ is schedulable with EDF-VD method on uniprocessor in low-criticality mode.*

$$U_{lo}^{LO} + \frac{U_{hi}^{LO}}{x} \leq 1 \tag{1}$$

The scaling factor x can be decided by $x = U_{hi}^{LO}/(1 - U_{lo}^{LO})$. In high-criticality mode, the method of [11] continues to execute low-criticality tasks with imprecise computing, and derives the following sufficient schedulability condition.

Theorem 2. *(Theorem 2 in [11]) If the following condition is satisfied, sporadic task set $\mathcal{T}$ is schedulable with EDF-VD method on uniprocessor in high-criticality mode.*

$$xU_{lo}^{LO} + (1-x)U_{lo}^{HI} + U_{hi}^{HI} \leq 1 \tag{2}$$

Theorem 3. *(Theorem 3 in [11]) Given a task set, if $\frac{U_{hi}^{LO}}{1-U_{lo}^{LO}} \leq \frac{1-(U_{hi}^{HI}+U_{lo}^{HI})}{U_{lo}^{LO}-U_{lo}^{HI}}$, where $U_{hi}^{HI} + U_{lo}^{HI} < 1$ and $U_{lo}^{LO} < 1$ and $U_{lo}^{LO} > U_{lo}^{HI}$, then this task set can be scheduled by EDF-VD with a deadline scaling factor x chosen in the following range*

$$x \in [\frac{U_{hi}^{LO}}{1-U_{lo}^{LO}}, \frac{1-(U_{hi}^{HI}+U_{lo}^{HI})}{U_{lo}^{LO}-U_{lo}^{HI}}] \tag{3}$$

C. Partitioned Scheduling on Multiprocessors

In this approach [4], $n = |\mathcal{T}|$ tasks are partitioned onto m unit-speed processors. After the partitioning, each task is never changed to another processor. As such, there is a fixed subset of tasks on each processor. Then, uniprocessor scheduling methods can be applied to each processor individually. In [4], a 2-phase task partitioning algorithm is described. In phase 1, high-criticality tasks are assigned to each processor one by one as long as the high-criticality utilization U_{hi}^{HI} for each processor does not exceed $\frac{3}{4}$. In phase 2, low-criticality tasks are further assigned to processors one by one following the condition that the low-criticality utilization $U_{lo}^{LO} + U_{hi}^{LO}$ is no greater than $\frac{3}{4}$. Alternatively, one can follow a different order of task assignment, which is applied with the same schedulability check. This order is obtained by sorting with non-increasing utilization (U_i^{LO} for low-critical tasks and U_i^{HI} for high-critical tasks) regardless task criticality. It is shown in [22] that this alternative order never sacrifices overall schedulability and sometimes improves schedulability.

D. Global Scheduling by fpEDF-VD on Multiprocessors

In global scheduling, jobs of the same task can be executed on different processors. One global scheduling method is fpEDF (fixed-priority EDF) [23], where the priority of each job cannot be changed during its execution. For a system with m unit-speed processors, fpEDF sets at most $m-1$ tasks with utilization greater than $\frac{1}{2}$ to have the highest priority and the priorities of the other tasks follow the EDF (Earliest Deadline First) principle. Based on [23], the schedulability condition of fpEDF is described as follows.

Lemma 1. *Given a task set $\mathcal{T}$ and m unit-speed processors, assume a subset $\mathcal{T}_{hp}$ have been assigned with the highest priority and allocated to $m_{hp} = |\mathcal{T}_{hp}|$ processors by fpEDF, then $m_{EDF} = m - m_{hp}$ is the number of processors for the remaining tasks $\mathcal{T}_{EDF}$, whose priority follows EDF. Let U_{EDF}^{total} and U_{EDF}^{max} denote the total utilization and the maximum utilization of tasks in $\mathcal{T}_{EDF}$. If $U_{EDF}^{total} \leq m_{EDF} - (m_{EDF} - 1) \cdot U_{EDF}^{max}$, this task set is schedulable by fpEDF.*

In [4], fpEDF is extended to mixed-criticality scheduling as follows. In low-criticality mode, each low-criticality task τ_i is treated with (T_i, C_i^{LO}) and each high-criticality task τ_k is regarded as $(\hat{T}_k, C_k^{LO})$, where $\hat{T}_k$ is the virtual implicit deadline decided by $\hat{T}_k = x \cdot T_k$ as in EDF-VD. In high-criticality mode, all low-criticality tasks are dropped and each high-criticality task τ_k is processed as $(T_k - \hat{T}_k, C_k^{HI})$. Lemma 1 continues to hold for fpEDF-VD scheduling [4]. The schedulability condition for fpEDF-VD method is that both task systems $(\bigcup_{\chi_i=lo}(T_i, C_i^{LO})) \bigcup (\bigcup_{\chi_i=hi}(x * T_i, C_i^{LO}))$ and $\bigcup_{\chi_i=hi}((1-x) * T_i, C_i^{HI})$ are each (separately) schedulable on m processors by fpEDF.

E. MC-DP-Fair Scheduling on Multiprocessors

DP (Deadline Partition) Fair [24] is a scheduling method for multiprocessor real-time system without mixed-criticality. Each task τ_i is defined with a density $\delta_i = \frac{C_i}{T_i}$, where C_i is its WCET and T_i is its period. Time is divided into slices by deadline partitions, each of which is a distinct job release time or deadline. A time slice is a time interval between two consecutive partitions. If the length of a time slice is l, DP-Fair executes $\delta_i \cdot l$ amount of task τ_i in this slice. The schedulability condition of DP-Fair is specified as follows.

Theorem 4. *(Lemma 14 in [15]) A non-MC task set $\mathcal{T}$ is schedulable under DP-Fair iff $\sum_{\tau_i \in \mathcal{T}} \delta_i \leq m$, where m is the number of processors.*

MC-DP-Fair [15] is an extension of DP-Fair for mixed-criticality systems. A main change is that each task τ_i is assigned a virtual deadline $0 < V_i \leq T_i$. Let Γ be the earliest deadline partition after a system is switched to high-criticality mode. The virtual deadlines and the original deadlines are enforced before and after Γ, respectively. By carefully choosing the values of virtual deadlines, MC-DP-Fair has schedulability equivalent to MC-Fluid [15], which is speedup-optimal for dual-criticality scheduling [16].

V. VPMC System Scheduling on Multiprocessors

Since VPMC and IMC systems follow the same schedulability conditions, we sometimes mention only one of them when a description is applicable for both kinds of models. Their difference is on how to exploit different computing precisions under the same schedulability constraints. The scheduling methods can be applied with different error models and almost any kind of imprecise computing techniques.

A. Partitioned Scheduling

1) VPMC Partitioning with EDF-VD Scheduling: For EDF-VD on uniprocessor VPMC systems, we introduce a sufficient schedulability condition that has a form similar to that in conventional MC systems.

Lemma 2. *If a task set in VPMC system satisfies the condition $max(U_{lo}^{LO} + U_{hi}^{LO}, U_{lo}^{HI} + U_{hi}^{HI}) \leq \frac{3}{4}$, it is schedulable by EDF-VD on uniprocessor.*

Proof: According to Lemma 2 in [11], if $max(b + \alpha c, \lambda b + c) \leq S(\alpha, \lambda)$, then $\frac{\alpha c}{1-b} \leq \frac{1-(c+\lambda b)}{b - \lambda b}$, where $U_{hi}^{HI} = c$, $U_{hi}^{LO} = \alpha c$, $U_{lo}^{LO} = b$, $U_{lo}^{HI} = \lambda b$ and $S(\alpha, \lambda) = \frac{(1-\alpha\lambda)((2-\alpha\lambda-\alpha)+(\lambda-1)\sqrt{4\alpha-3\alpha^2})}{2(1-\alpha)(\alpha\lambda-\alpha\lambda^2-\alpha+1)}$. Based on Theorem 4 in [11], $S(\alpha, \lambda) \geq \frac{3}{4}$. As such, if $max(U_{lo}^{LO} + U_{hi}^{LO}, U_{lo}^{HI} + U_{hi}^{HI}) \leq \frac{3}{4}$, then $\frac{U_{hi}^{LO}}{1-U_{lo}^{LO}} \leq \frac{1-(U_{hi}^{HI}+U_{lo}^{HI})}{U_{lo}^{LO}-U_{lo}^{HI}}$, which is the sufficient schedulability condition for EDF-VD according to Theorem 3. ■

In this method, the given tasks are first partitioned onto m unit-speed processors in the same order as that described in Section IV-C. When a task is assigned to a processor, the schedulability check is based on Lemma 2 instead of the conventional approach [4]. This change is to accommodate the IMC/VPMC model. This partitioning method is called *VPMC partitioning*. After the partitioning, the tasks on each processor are scheduled in the same way as EDF-VD under the IMC model [11] (Section IV-B). Under the same schedulability constraints, VPMC further allows some low-criticality task to execute with full precision in high-criticality mode.

Lemma 3. *If the VPMC partitioning is successfully completed, all tasks on each processor are schedulable using EDF-VD method.*

Proof: Each time a task is assigned to a processor in the VPMC partitioning, the schedulability condition specified by Lemma 2 is satisfied. If VPMC partitioning is successfully completed, the tasks on each processor satisfy the schedulability condition of Lemma 2 and therefore are schedulable by EDF-VD. ■

It is shown in [4] that the partitioned scheduling of conventional MC model can achieve speedup factor of $\frac{8m-4}{3m}$ for m unit-speed processors. We show that the same speedup factor can be achieved for IMC/VPMC model through a proof similar to [4].

Theorem 5. *The speedup factor for VPMC partitioning with EDF-VD scheduling on m unit-speed processors is $\frac{8m-4}{3m}$.*

Proof: Suppose $i-1$ tasks have been successfully assigned and we are attempting to assign the i_{th} task τ_i onto a processor during the partitioning. Let $\tau(p_k)$ denote the set of tasks that have been successfully assigned to processor $p_k, 1 \leq k \leq m$. If the assignment of τ_i fails according to the schedulability check, then at least one of the following two inequalities must hold.

$$U_i^{LO} + \sum_{\tau_j \in \tau(p_k)} U_j^{LO} > \frac{3}{4} \tag{4}$$

$$U_i^{HI} + \sum_{\tau_j \in \tau(p_k)} U_j^{HI} > \frac{3}{4} \tag{5}$$

We can sum up inequality (4) for all the m processors to get $\sum_{j=1}^{i-1} U_j^{LO} > \left(\frac{3}{4} - U_i^{LO}\right) m \Leftrightarrow \sum_{j=1}^{i} U_j^{LO} > (\frac{3}{4} - U_i^{LO})m + U_i^{LO}$, from which we can conclude

$$U_{lo}^{LO} + U_{hi}^{LO} > \left(\frac{3}{4} - U_i^{LO}\right) m + U_i^{LO} \tag{6}$$

Similarly, we can sum up inequality (5) for all the m processors to obtain

$$U_{lo}^{HI} + U_{hi}^{HI} > \left(\frac{3}{4} - U_i^{HI}\right) m + U_i^{HI} \tag{7}$$

If this task set can be scheduled by an optimal scheduling algorithm on m processors of speed s, we have $U_i^{LO} \leq s$, $U_i^{HI} \leq s$, $U_{lo}^{LO} + U_{hi}^{LO} \leq m \cdot s$ and $U_{lo}^{HI} + U_{hi}^{HI} \leq m \cdot s$.

If inequality (6) holds, $U_{lo}^{LO} + U_{hi}^{LO} > (\frac{3}{4} - U_i^{LO})m + U_i^{LO} \Leftrightarrow m \cdot s > \frac{3}{4}m - (m-1)s \Leftrightarrow s > \frac{3m}{8m-4}$. Likewise we can obtain $s > \frac{3m}{8m-4}$ if inequality (7) holds. If $s \leq \frac{3m}{8m-4}$, this task set can be scheduled by the partitioning followed by EDF-VD on m unit-speed processors. Therefore, the speedup factor of the VPMC partitioning with EDF-VD is $\frac{8m-4}{3m}$. ■

2) Enhanced VPMC Partitioning: We introduce two techniques to enhance the VPMC partitioning described in Section V-A1. The first improvement is to change the schedulability check in the partitioning from Lemma 2 to Theorem 3. From the proof of Lemma 2, we can tell the schedulability condition in Lemma 2 is sufficient for the schedulability condition in Theorem 3. We use an example to demonstrate that the Lemma 2 condition is not necessary for the Theorem 3 condition. The characteristics of this example task set are shown in Table I. If the partitioning is based on Lemma 2, τ_1 and τ_2 are first assigned to processor p_1 and p_2, respectively. When we try to assign τ_3 to processor p_1, $max(U_{lo}^{LO} + U_{hi}^{LO}, U_{lo}^{HI} + U_{hi}^{HI}) = 0.9 > 0.75$. Alternatively, when we try to assign τ_3 to processor p_2, $max(U_{lo}^{LO} + U_{hi}^{LO}, U_{lo}^{HI} + U_{hi}^{HI}) = 0.8 > 0.75$. Hence, the assignment for τ_3 fails for both p_1 and p_2 according to Lemma 2. However, the assignment of τ_3 to p_2 satisfies the schedulability condition in Theorem 3 since $\frac{U_{hi}^{LO}}{1-U_{lo}^{LO}} = \frac{3}{5} \leq \frac{1-(U_{hi}^{HI}+U_{lo}^{HI})}{U_{lo}^{LO}-U_{lo}^{HI}} = \frac{2}{3}$. Thus, the condition in Lemma 2 is more conservative than Theorem 3 and applying Theorem 3 can identify more schedulable task sets.

The second enhancement technique is to balance the utilizations of each processor between the two different criticality modes. More specifically, we attempt to make the difference

TABLE I. SCHEDULING ON 2 UNIT-SPEED PROCESSORS.

Task	χ_i	U_i^{LO}	U_i^{HI}
τ_1	hi	0.4	0.7
τ_2	hi	0.3	0.6
τ_3	lo	0.5	0.2

between $U_{lo}^{LO} + U_{hi}^{LO}$ and $U_{lo}^{HI} + U_{hi}^{HI}$ on each processor as small as possible. The intuition is that a small difference or balanced utilization can avoid one criticality mode being a bottleneck of the whole system. This is inspired by the work on conventional MC systems [22], but applies to IMC/VPMC systems as well. Each time a task τ_i is to be assigned to a processor, all the processors are sorted in non-decreasing order of $U_{lo}^{HI} + U_{hi}^{HI} - U_{lo}^{LO} - U_{hi}^{LO}$ and indexed from 1 to m. If $\chi_i = hi$, the attempts of assigning τ_i to a processor are in the order from 1 to m. Otherwise, the attempts follow the order from m to 1.

Lemma 4. *If the enhanced VPMC partitioning is successfully completed, all tasks on each processor are schedulable with EDF-VD scheduling.*

Proof: A successful assignment of tasks to a processor indicates the satisfaction of the condition in Theorem 3, which is a sufficient schedulability condition for EDF-VD. ■

Lemma 5. *The speedup factor for the enhanced partitioning with EDF-VD scheduling is no greater than* $\frac{8m-4}{3m}$.

Proof: Since the condition of Lemma 2 is sufficient condition for the condition in Theorem 3, a failure of the enhanced partitioning, which implies violation of the condition in Theorem 3, indicates violation of the condition of Lemma 2, i.e., either inequality (4) or inequality (5) holds. Then, one can follow the same proof as Theorem 5 to show that the speedup factor is no greater than $\frac{8m-4}{3m}$. ■

B. Global Scheduling by fpEDF-VD

1) Extension of fpEDF-VD for IMC and VPMC: When fpEDF-VD scheduling, which is briefly reviewed in Section IV-D, is applied with IMC/VPMC, the main change is that C_i^{HI} for each low-criticality task τ_i is no longer 0. This execution time change causes utilization change in high-criticality mode.

The tricky part is the transition from low-criticality mode to high-criticality mode. Let t^{hi} be the moment when the system enters high-criticality mode. We define d^{HI} as the earliest deadline (virtual deadline for high-criticality tasks) among all jobs that are active right after t^{hi}. We further define r^{HI} to be the earliest release time among jobs released after t^{hi}. We call $t^{HI} = \min(d^{HI}, r^{HI})$ the *critical moment*. After the critical moment, the schedulability check of high-criticality mode can be applied without ambiguity. However, the transition time interval from t^{hi} to t^{HI} needs special consideration for IMC/VPMC systems. During the transition interval, there can exist carry-over jobs, which are jobs that are released before t^{hi} and have not been completed at t^{hi}. By the EDF-VD algorithm design, high-criticality carry-over jobs can be guaranteed to complete before their deadlines if the schedulability check is passed. If a low-criticality carry-over job J_i^j has already executed at least $\tilde{C}_i$ amount of time at t^{hi}, we take its imprecise computing result [13] and quit this job. If J_i^j has executed less than $\tilde{C}_i$, we continue it till t^{HI} and then quit. By disallowing low-criticality carry-over jobs after t^{HI}, the schedulability of all high-criticality jobs can be maintained. In the worst case, a low-criticality task may lose its job once during the transition interval.

The schedulability condition for our fpEDF-VD method is that both task systems $(\bigcup_{\chi_i=lo}(T_i, C_i^{LO}))\bigcup(\bigcup_{\chi_i=hi}(x * T_i, C_i^{LO}))$ and $(\bigcup_{\chi_i=lo}(T_i, C_i^{HI}))\bigcup(\bigcup_{\chi_i=hi}((1-x) * T_i, C_i^{HI}))$ are each (separately) schedulable on m processors by fpEDF (for low-criticality tasks, $C_i^{HI} = \tilde{C}_i$). The schedulability condition of fpEDF is Lemma 1 in Section IV-D.

Lemma 6. *If a task set satisfies* $\frac{U_{hi}^{LO}}{1-U_{lo}^{LO}/s} + \frac{U_{hi}^{HI}}{1-U_{lo}^{HI}/s} \leq s$, *then it is schedulable using our fpEDF-VD method on a speed s processor.*

Proof: fpEDF reduces to regular EDF on single processor [23]. If a task set satisfies $U_{lo}^{LO} + \frac{U_{hi}^{LO}}{x} \leq s$, or equivalently

$$x \geq \frac{U_{hi}^{LO}}{s - U_{lo}^{LO}} \tag{8}$$

task collection $(\bigcup_{\chi_i=lo}(T_i, C_i^{LO}))\bigcup(\bigcup_{\chi_i=hi}(x * T_i, C_i^{LO}))$ is schedulable using EDF on a speed s processor.

If the task set satisfies $U_{lo}^{HI} + \frac{U_{hi}^{HI}}{1-x} \leq s$, or equivalently

$$x \leq 1 - \frac{U_{hi}^{HI}}{s - U_{lo}^{HI}} \tag{9}$$

task collection $(\bigcup_{\chi_i=lo}(T_i, C_i^{HI}))\bigcup(\bigcup_{\chi_i=hi}(T_i - x * T_i, C_i^{HI}))$ is schedulable using EDF on a speed s processor.

We can prove this lemma when combining inequality (8) and inequality (9). ■

Lemma 7. *If a task set satisfies* $max(U_{lo}^{LO} + U_{hi}^{LO}, U_{lo}^{HI} + U_{hi}^{HI}) \leq s$, *then it is schedulable by our fpEDF-VD method on a speed ks processor , where* $k = \frac{\sqrt{5}+1}{2}$.

Proof: If a task set satisfies $U_{lo}^{LO} + U_{hi}^{HI} \leq ks$, we know that it is schedulable on a speed ks processor, if not we need to show that (from Lemma 6),

$\frac{U_{hi}^{LO}}{1-U_{lo}^{LO}/ks} + \frac{U_{hi}^{HI}}{1-U_{lo}^{HI}/ks} \leq ks$, or equivalently,

$ks(U_{lo}^{LO}+U_{hi}^{LO})+ks(U_{lo}^{HI}+U_{hi}^{HI})-U_{lo}^{LO}(U_{lo}^{HI}+U_{hi}^{HI})-U_{hi}^{LO}U_{lo}^{HI} \leq (ks)^2$

From $max(U_{lo}^{LO}+U_{hi}^{LO}, U_{lo}^{HI}+U_{hi}^{HI}) \leq s$, we have $U_{lo}^{LO}+U_{hi}^{LO} \leq s$, $U_{lo}^{HI} + U_{hi}^{HI} \leq s$, and from $U_{lo}^{LO} + U_{hi}^{HI} > ks$, we have $U_{lo}^{LO} > ks - U_{hi}^{HI} > ks - s$, then

$ks(U_{lo}^{LO}+U_{hi}^{LO})+ks(U_{lo}^{HI}+U_{hi}^{HI})-U_{lo}^{LO}(U_{lo}^{HI}+U_{hi}^{HI})-U_{hi}^{LO}U_{lo}^{HI} < 2ks^2 - (k-1)s^2 = (k+1)s^2 = (ks)^2$, because $k+1 = \frac{\sqrt{5}+3}{2} = k^2$.

■

Corollary 1. *(Corollary 1 in [3]) If a task set cannot be scheduled by algorithm fpEDF on m unit-speed processors, then it cannot be scheduled by preemptive uniprocessor EDF on a processor of speed (m+1)/2.*

From Lemma 7 and Corollary 1, we can have Corollary 2 as in [4],

Corollary 2. *If a task set satisfies* $max(U_{lo}^{LO} + U_{hi}^{LO}, U_{lo}^{HI} + U_{hi}^{HI}) \leq m$, *then it is schedulable by our fpEDF-VD method on m speed* $(\sqrt{5}+1)$ *processors.*

Corollary 3. *Any task set that can be scheduled by an optimal clairvoyant scheduling algorithm on m unit speed processors* *can be scheduled by our fpEDF-VD method on m speed ($\sqrt{5}+1$) processors.*

Proof: If a task set can be scheduled by an optimal clairvoyant scheduling algorithm on m unit speed processors, it is necessary that $U_{lo}^{LO} + U_{hi}^{LO} \leq m$ and $U_{lo}^{HI} + U_{hi}^{HI} \leq m$, then this task set is schedulable by our fpEDF-VD method on m speed $(\sqrt{5}+1)$ processors from Corollary 2. ■

From Corollary 3, we can show that our fpEDF-VD method for IMC/VPMC has the speedup factor of $(\sqrt{5}+1)$, which is the same as the fpEDF-VD scheduling of conventional MC systems.

2) Dual Virtual-Deadlines for fpEDF (fpEDF-DVD): As pointed in Section V-B1, a direct extension of fpEDF-VD to IMC/VPMC model may result in one-time job abandonment for a low-criticality task during the transition from low-criticality to high-criticality mode. To avoid this loss, we propose to apply the virtual-deadline technique for low-criticality tasks in addition to high-criticality tasks. More specifically, each low-criticality task τ_i has deadlines $y \cdot T_i$ and $(1-y)\cdot T_i$ for low-criticality mode and high-criticality mode, respectively, where y is a scaling factor between 0 and 1. The value of y is found by sweeping between 0 and 1 and selecting the one that satisfies schedulability conditions (both task systems $(\bigcup_{\chi_i=lo}(y*T_i, C_i^{LO}))\bigcup(\bigcup_{\chi_i=hi}(x*T_i, C_i^{LO}))$ and $(\bigcup_{\chi_i=lo}((1-y)*T_i, C_i^{HI}))\bigcup(\bigcup_{\chi_i=hi}((1-x)*T_i, C_i^{HI}))$ are each (separately) schedulable on m processors by fpEDF). This method is called fpEDF-DVD (fpEDF with dual virtual-deadlines).

Theorem 6. *If the virtual deadline based utilization of all tasks satisfy schedulability conditions in both low-criticality and high-criticality mode, the fpEDF-DVD scheduling guarantees all job completions before their deadlines and no job is abandoned.*

Proof: If the schedulability conditions are satisfied, all tasks are evidently schedulable by fpEDF in low-criticality mode and high-criticality mode. Special attention needs to be paid to carry-over jobs, which are released before the moment t^{hi} entering high-criticality mode and have not been completed at t^{hi}. Then, the low-criticality mode virtual-deadline for each carry-over job must be after t^{hi}. The virtual-deadlines partition a task period into low-criticality mode portion, which are $x \cdot T_i$ and $y \cdot T_i$, and high-criticality mode portion, which are $(1-x)\cdot T_i$ and $(1-y)\cdot T_i$, respectively. For the carry-over jobs, one can treat their low-criticality mode virtual-deadlines as their high-criticality mode release times, which are after t^{hi}. As the schedulability conditions are satisfied, even if the carry-over jobs start execution at their low-criticality virtual-deadlines, they are all schedulable for completion by their actual deadlines.

■

C. Extension of MC-DP-Fair Scheduling for IMC and VPMC Systems

MC-DP-Fair is one realization of the fluid-based scheduling [15], which is not directly implementable by itself. Fluid-based scheduling associating Quality of service for low critical tasks has been studied for VPMC in [12] and the method is called MCFQ, however, MC-DP-Fair scheduling for VPMC is barely discussed in [12]. Here, we show how to extend MC-DP-Fair scheduling to VPMC-DP-Fair scheduling. In DP-Fair scheduling, an important concept is task density δ_i for task τ_i, which is usually equal to $\frac{C_i}{T_i}$ with a few exceptions. Fluid-based scheduling uses another concept, execution rate θ_i for τ_i, which is the fraction of a unit-speed processor allocated for executing τ_i.

For a low-critical task τ_i in VPMC-DP-Fair, $\delta_i^{LO} = \theta_i^{LO} = U_i^{LO}$ and $\delta_i^{HI} = \theta_i^{HI} = U_i^{HI}$, where the superscripts ^{LO} and ^{HI} indicate low-criticality and high-criticality mode, respectively. Its virtual deadline $V_i = T_i$. Please note $\delta_i^{HI} = 0$ in MC-DP-Fair. Let w_i be the length of time interval from job release time of τ_i to Γ, which is the earliest deadline partition after the system enters high-criticality mode.

Lemma 8. *In VPMC-DP-Fair scheduling, a low-criticality carry-over job of τ_i can be executed for at least $\tilde{C}_i$ time, where $\tilde{C}_i$ is the execution time of imprecise implementation.*

Proof: Let C_i^{TR} denote the actual execution time of a carry-over job of τ_i.

$$C_i^{TR} = w_i \cdot \delta_i^{LO} + (T_i - w_i)\delta_i^{HI} = w_i \cdot U_i^{LO} + (T_i - w_i)U_i^{HI}$$
$$\geq w_i \cdot U_i^{HI} + (T_i - w_i)U_i^{HI} = T_i \cdot U_i^{HI} = \tilde{C}_i$$

■

For a high-criticality task τ_i, $\delta_i^{LO} = \theta_i^{LO}$, which is proved to be no greater than U_i^{HI} [12], and virtual deadline $V_i = C_i^{LO}/\theta_i^{LO}$. Its density in high-criticality mode is specified by [15]

$$\delta_i^{HI} = \frac{C_i^{HI} - \delta_i^{LO} \cdot w_i}{T_i - w_i}. \tag{10}$$

Lemma 9. *Given a task set that is deemed to be schedulable by MCFQ [12], if it is scheduled by VPMC-DP-Fair, then $\delta_i^{LO} \leq \theta_i^{LO}$ and $\delta_i^{HI} \leq \theta_i^{HI}$ for each task τ_i.*

Proof: For each task τ_i, we have $\delta_i^{LO} = \theta_i^{LO}$. For each low-criticality task τ_i, we have $\delta_i^{HI} = \theta_i^{HI}$. For each high-criticality task τ_i, since δ_i^{HI} is a variable depending on w_i according to Equation (10), we need to show that the maximum value of δ_i^{HI} is no greater than θ_i^{HI}. Consider the derivative of δ_i^{HI} with respect to w_i

$$\frac{d\delta_i^{HI}}{dw_i} = \frac{C_i^{HI} - \delta_i^{LO} \cdot T_i}{(T_i - w_i)^2} = \frac{U_i^{HI} - \delta_i^{LO}}{T_i \cdot (T_i - w_i)^2}. \tag{11}$$

Since $\delta_i^{LO} = \theta_i^{LO} \leq U_i^{HI}$ [12], the derivative is non-negative and the function of Equation (10) is monotonically increasing. By definition, we know $w_i \leq V_i$. Thus, δ_i^{HI} has the maximum value when $w_i = V_i$,

$$\delta_{i,max}^{HI} = \frac{U_i^{HI} - U_i^{LO}}{1 - U_i^{LO}/\theta_i^{LO}} \tag{12}$$

In MCFQ [12], $\theta_i^{HI} = \frac{U_i^{HI} - U_i^{LO}}{1 - U_i^{LO}/\theta_i^{LO}}$, which is equal to $\delta_{i,max}^{HI}$, then we have $\delta_i^{HI} \leq \theta_i^{HI}$ for high-criticality tasks. ■

Lemma 10. *Given a task set that is deemed to be schedulable by MCFQ, it is schedulable by VPMC-DP-Fair.*

Proof: Given a task set that is deemed to be schedulable by MCFQ, we have $\sum_{\tau_i \in \mathcal{T}} \theta_i^{LO} \leq m$ and $\sum_{\tau_i \in \mathcal{T}} \theta_i^{HI} \leq m$, then we have $\sum_{\tau_i \in \mathcal{T}} \delta_i^{LO} \leq \sum_{\tau_i \in \mathcal{T}} \theta_i^{LO} \leq m$ and $\sum_{\tau_i \in \mathcal{T}} \delta_i^{HI} \leq \sum_{\tau_i \in \mathcal{T}} \theta_i^{HI} \leq m$ from Lemma 9. Hence, low-criticality mode schedulability and high-criticality mode schedulability by Theorem 4 are satisfied and the task set is schedulable by VPMC-DP-Fair. ■

VI. Precision Optimization for VPMC Systems

A. Optimization Kernel

Under the VPMC model, there can be utilization slack for some processors when schedulability conditions are satisfied.

The slack allows some low-criticality tasks to be executed with precise computing in high-criticality mode while the schedulability conditions are still satisfied. For a low-criticality task τ_i, the error of its imprecise computing is denoted by e_i. The error of a low-criticality task τ_i execution in high-criticality mode is denoted by e_i^{HI}, which is equal to e_i if it is executed with imprecise computing and otherwise 0. If each task τ_i has a weighting factor η_i indicating its importance, the precision optimization problem is stated as follows.

Problem 1. *Given a set of independent sporadic tasks $\mathcal{T} = \{\tau_1, \tau_2, ...\}$ in VPMC model and a scheduling method $\mathcal{S}$, decide if each low-criticality task τ_i is executed with precise or imprecise computing in high-criticality mode such that the total weighted error $\sum_{\chi_i=lo} \eta_i \cdot e_i^{HI}$ is minimized while the schedulability conditions for $\mathcal{S}$ are maintained.*

For each low-criticality task τ_i, let ΔU_i denote the additional processor utilization when its execution is changed from imprecise to precise computing and thus

$$\Delta U_i = \frac{\hat{C}_i - \tilde{C}_i}{T_i}. \tag{13}$$

Let $\bar{U}_{lo}^{HI}$ denote the maximal possible U_{lo}^{HI} under the schedulability constraint for a scheduling method. The *utilization slack* Ψ for low-critical tasks in high-criticality mode is defined as

$$\Psi = \bar{U}_{lo}^{HI} - U_{lo}^{HI} \tag{14}$$

Then, Problem 1 is essentially 0-1 knapsack problem. Let z_i be a binary decision variable for each low-criticality task τ_i. When $z_i = 1$, task τ_i is assigned to precise computing; otherwise it is executed with imprecise computing in high-criticality mode. The knapsack problem formulation is as follows.

$$\begin{aligned} \text{maximize} & \sum_{\chi_i=lo} \eta_i \cdot e_i \cdot z_i \\ \text{subject to} & \sum_{\chi_i=lo} \Delta U_i \cdot z_i \leq \Psi \\ & z_i \in \{0,1\}, \quad \forall \tau_i \in \mathcal{T}_{lo} \end{aligned} \tag{15}$$

In this formulation, the objective is to maximize the error reduction obtained from using precise computing compared to IMC model. The 0-1 knapsack problem is a well-known NP-complete problem. It can be optimally solved by dynamic programming with pseudo-polynomial complexity.

B. Utilization Slack Estimation and Customization for Different Scheduling Methods

1) Slack Estimation and Precision Optimization for Partitioned Scheduling: For partitioned scheduling, if $U_{lo}^{LO} + U_{hi}^{HI} \leq 1$, all tasks can be scheduled with EDF and all low-criticality tasks can be executed with precise computing. Hence, the slack estimation and precision optimization is necessary only when $U_{lo}^{LO} + U_{hi}^{HI} > 1$. For both of the partitioned scheduling methods introduced in section V-A, utilization slack is estimated for individual processors. On each processor, the maximal schedulable utilization $\bar{U}_{lo}^{HI}$ can be derived according to Theorem 1 and Theorem 2.

Theorem 7. *The utilization slack of a processor after the VPMC partitioning is $\frac{1-U_{lo}^{LO}-U_{hi}^{LO}U_{lo}^{LO}-U_{hi}^{HI}+U_{lo}^{LO}U_{hi}^{HI}}{1-U_{lo}^{LO}-U_{hi}^{LO}} - U_{lo}^{HI}$.*

Proof: From inequality (1), we can find the range of the scaling factor as

$$x \geq \frac{U_{hi}^{LO}}{1 - U_{lo}^{LO}} \tag{16}$$

Further, we know from inequality (2) that

$$U_{lo}^{HI} \leq \frac{1 - xU_{lo}^{LO} - U_{hi}^{HI}}{1 - x} \tag{17}$$

Taking derivative with respective to x on right-hand-side of inequality (17), we have

$$\frac{1 - U_{lo}^{LO} - U_{hi}^{HI}}{(1-x)^2} \tag{18}$$

Since $U_{lo}^{LO} + U_{hi}^{HI} > 1$, the right-hand-side of inequality (17) is a decreasing function with respect to x. Then, $\bar{U}_{lo}^{HI}$ can be obtained by plugging RHS of inequality (16) into inequality (17):

$$\bar{U}_{lo}^{HI} = \frac{1 - U_{lo}^{LO} - U_{hi}^{LO}U_{lo}^{LO} - U_{hi}^{HI} + U_{lo}^{LO}U_{hi}^{HI}}{1 - U_{lo}^{LO} - U_{hi}^{LO}} \tag{19}$$

Therefore, the utilization slack is given by:

$$\Psi = \frac{1 - U_{lo}^{LO} - U_{hi}^{LO}U_{lo}^{LO} - U_{hi}^{HI} + U_{lo}^{LO}U_{hi}^{HI}}{1 - U_{lo}^{LO} - U_{hi}^{LO}} - U_{lo}^{HI} \tag{20}$$

■

2) Slack Estimation and Precision Optimization for fpEDF-VD Based Global Scheduling: Under fpEDF, a subset $\mathcal{T}_{hp} \subset \mathcal{T}$ of tasks are designated with the highest priority and $m_{hp} = |\mathcal{T}_{hp}|$ processors are allocated for them. Please note this allocation is not static, i.e., the m_{hp} processors at one time may be different from the m_{hp} processors at another time. The other tasks $\mathcal{T}_{EDF} = \mathcal{T} - \mathcal{T}_{hp}$ follow EDF priority and are executed on $m_{EDF} = m - m_{hp}$ processors. Each low-criticality task $\tau_i \in \mathcal{T}_{hp}$ can always execute with precise computing in high-criticality mode, since an entire processor is allocated to one task in $\mathcal{T}_{hp}$ and this allocation is sufficient for precise computing.

For the fpEDF-VD-VPMC method described in Section V-B1, the utilization slack of $\mathcal{T}_{EDF}$ is estimated by the following statement according to Lemma 1.

Proposition 1. *The utilization slack for $\mathcal{T}_{EDF}$ on the m_{EDF} processors under fpEDF-VD-VPMC scheduling is $m_{EDF} - (m_{EDF} - 1) \cdot U_{EDF}^{max} - U_{EDF}^{total}$, where U_{EDF}^{max} and U_{EDF}^{total} are the maximal task utilization and total utilization for $\mathcal{T}_{EDF}$, respectively.*

This estimation can be applied with fpEDF-DVD-VPMC method described in Section V-B2. However, the partition of $\mathcal{T}_{hp}$ and $\mathcal{T}_{EDF}$ in Section V-B2 is different from that in Section V-B1 due to the virtual-deadlines applied to low-criticality tasks.

3) Utilization Slack Estimation for VPMC-DP-Fair Scheduling: The utilization slack for VPMC-DP-Fair Scheduling is estimated by $\Psi = m - \sum_{\tau_i \in \mathcal{T}} \theta_i^{HI}$, where θ_i^{HI} is the execution rate of task τ_i in high-criticality mode, which is computed according to [12].

VII. Experimental Results

In our experiments, we evaluate the schedulability and computing errors of the following methods through software simulations and/or Linux prototyping:

- **Partition-MC**: Partitioned scheduling with the conventional MC model [4]. Since this method does not incorporate any approximations, its results are used to provide a reference level for schedulability, but cannot be used for comparing computing errors.
- **Partition-VPMC**: The partitioned scheduling method in Section V-A1 with precision optimization.

- **Partition-VPMC-E**: Enhanced partitioned scheduling (Section V-A2) with precision optimization.
- **fpEDF-VD-MC**: fpEDF-VD scheduling with the conventional MC model [4]. Since this method drops all low-criticality tasks in high-criticality mode, it is not included for error analysis.
- **fpEDF-VD-VPMC**: fpEDF-VD scheduling (Section V-B1) with precision optimization.
- **fpEDF-DVD-VPMC**: fpEDF dual virtual-deadline method (Section V-B2), with precision optimization.
- **Fluid-VPMC**: The MCFQ method [12] with precision optimization replaced by the dynamic programming technique in Section VI. Since the fluid-based scheduling is not directly implementable, this method is only evaluated with software simulation, to conduct the schedulability check and estimate error.
- **VPMC-DP-Fair**: The scheduling method described in Section V-C with precision optimization. Since this is an implementable realization of Fluid-VPMC, it is evaluated only through Linux prototyping.

A. *Simulation Setup and Results*

The testcases are randomly generated as follows:

- For each task set, the probability of a task being low-criticality (high-criticality) is 0.5.
- For a low-criticality (high-criticality) task τ_i, its utilization in low-criticality (high-criticality) mode U_i^{LO} (U_i^{HI}) is randomly chosen within the interval, $[0.05, 0.9]$, under a uniform distribution.
- The period T_i of each task is randomly chosen from a uniform distribution in $[50, 500]$.
- For a low-criticality task τ_i, we set $C_i^{LO} = \hat{C}_i = T_i \cdot U_i^{LO}$, $\tilde{C}_i = k_{lo} \cdot \hat{C}_i$, where the scaling factor k_{lo} is randomly chosen from a uniform distribution in $[K_{lo}, 0.9]$, where K_{lo} is a parameter.
- For a high-criticality task τ_i, we set $C_i^{HI} = T_i \cdot U_i^{HI}$, $C_i^{HI} = k_{hi} \cdot C_i^{LO}$ and $1.1 \le k_{hi} \le K_{hi}$, where K_{hi} is a parameter. For each low-criticality task τ_i, its imprecise computing error is randomly chosen from a uniform distribution between 1 and 10.

We set error weighting factors (defined in Section VI) $\eta_i = 1$.

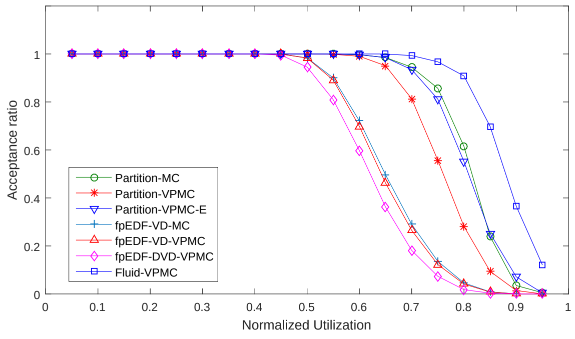

Fig. 1. Acceptance ratio vs. normalized utilization of 4 processors ($K_{lo} = 0.1$, $K_{hi} = 5$).

Evaluation of the acceptance ratio: We first evaluate the acceptance ratio at several values of the utilization, U_i. For each U_i, we generate 10,000 testcases, and for each testcase, we iteratively add new tasks till $\max(U_{lo}^{LO} + U_{hi}^{LO},\ U_{lo}^{HI} + U_{hi}^{HI})$ reaches U_i. The acceptance ratios on 4 processors is depicted in Figure 1.

We see from the plot that Fluid-VPMC provides the best acceptance ratio (this is not surprising as the fluid-based scheduling is optimal in theory), while the three variants of fpEDF-VD have the lowest acceptance ratio due to their very conservative schedulability conditions. The acceptance ratio of fpEDF-VD-VPMC is very close to that of fpEDF-VD-MC. This implies that continuing low-criticality tasks at high-criticality mode hardly degrades schedulability. The dual virtual-deadline technique reduces acceptance ratio, but it guarantees that no low-criticality job is dropped while the fpEDF-VD-VPMC cannot provide such guarantees. The result also shows that the enhancement techniques introduced in Section V-A2 can indeed improve schedulability of partitioned scheduling.

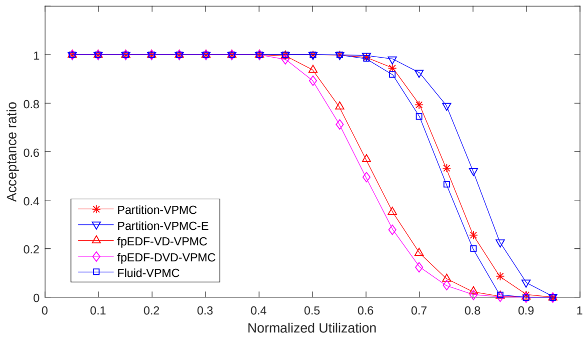

Fig. 2. Acceptance ratio versus normalized utilization of 4 processors with consideration of overhead.

The simulation for Figure 1 does not consider overhead, which is important in practice. Overhead includes the time on context switching, job migration among processors, execution monitoring, scheduling job executions, etc. For each of the VPMC methods, we estimate its overhead according to the Linux prototyping (Section VII-B) data. Then, the overhead is added into the task execution time for the simulation. The acceptance ratio result with consideration of overhead is shown in Figure 2. One can see that Fluid-VPMC is no longer the best due to its large overhead, and the best results are obtained from partitioned scheduling. The gap between Fluild-VPMC and fpEDF-VD-VPMC also becomes smaller.

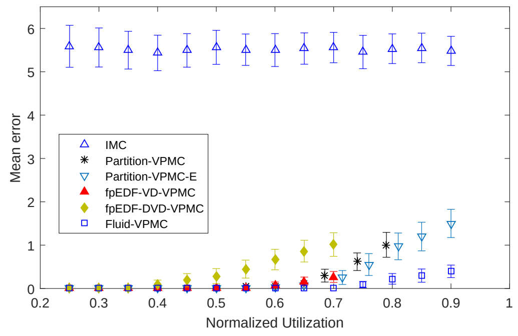

Fig. 3. Mean error (with standard deviation) vs. normalized utilization of 4 processors ($K_{lo} = 0.1$, $K_{hi} = 5$).

Evaluation of errors: Next, we evaluate computing errors of low-criticality tasks in high-criticality mode for different methods. Following the same testcase generation for evaluating the acceptance ratio, 1000 *schedulable* testcases are obtained at each utilization value. Figures 3 and 4 show the mean error with standard deviation among tasks as function of the normalized utilization. For a single testcase, minimizing mean error is equivalent to minimizing the total error as the number of tasks is a constant for the precision optimization. When evaluating multiple testcases, mean error is more like a normalized result that can avoid the result being dominated by a few cases. In both of the figures, errors from IMC is plotted besides those from other method. IMC is the model where all low-criticality tasks continue with imprecise computing in high-criticality mode. Hence, its error is the same for different scheduling methods. One can see that the VPMC model can provide large error reductions. Again, Fluid-VPMC provides the lowest error levels as its optimality allows more utilization slack for error reduction.

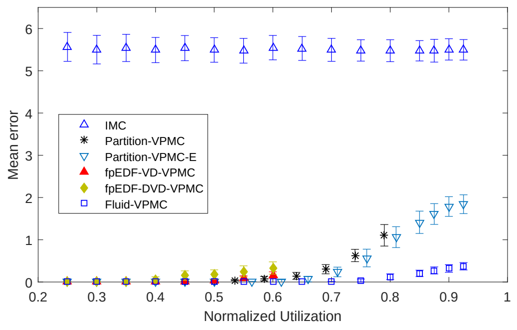

Fig. 4. Mean error (with standard deviation) vs. normalized utilization of 8 processors ($K_{lo} = 0.1$, $K_{hi} = 5$).

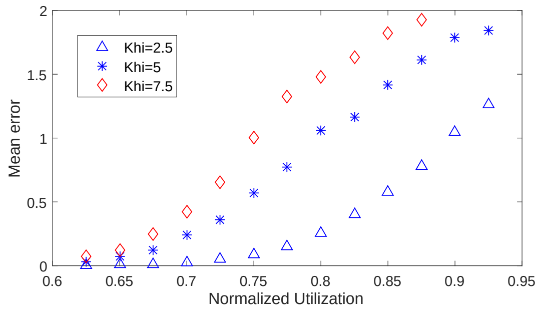

Fig. 5. The effect of K_{hi} on errors for Partition-VPMC-E on 8 processors.

Figures 5 shows the effect of K_{hi} on errors for Partition-VPMC-E. In general, a large K_{hi} tends to cause large errors. We also studied the effect of parameter K_{lo} with result shown in Figure 6. Interestingly, the error decreases as K_{lo} increases. For a large K_{lo}, the difference between precise and imprecise computing execution times is small, i.e., the additional utilization for changing imprecise computing to precise computing is small and applying precise computing becomes easier. On the other hand, a large K_{lo} increases the overall utilization and degrades schedulability.

B. Prototyping in Linux user space

We evaluate the proposed techniques in the VPMC model with prototyping in Linux user space. Such prototyping can

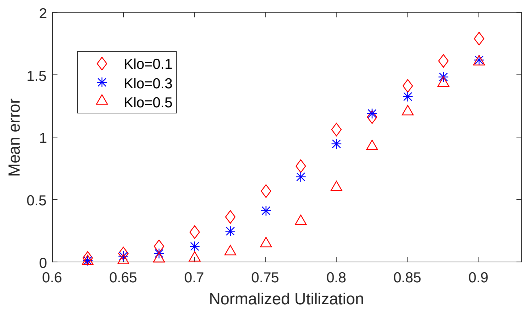

Fig. 6. The effect of K_{lo} on errors for Partition-VPMC-E on 8 processors.

account for the overhead, which is neglected in software simulation. Moreover, the error model of the prototyping is more realistic. The prototyping is implemented in the user space of Linux 4.10 on a 4-processor machine, where each processor is Intel Core i3 with frequency 1.9GHz.

There are multiple threads for task execution and scheduling, each of which has an associated priority for realizing task execution or preemption. We create a managing thread that conducts the task scheduling at runtime. The managing thread can generate job threads. Each job thread is either in execution mode, or waiting mode when it is released, not completed and not being executed. The job thread management is performed at every time unit of 0.01 second, which allows sufficient resolution for the testcases. At each time unit, the managing thread checks if a new job is released, a job execution is completed, a job execution exceeds its WCET and if there is deadline violation. According to specific scheduling method, the managing thread decides if to start a waiting job, if a low-priority job being executed needs to be preempted, if to switch low-criticality mode to high-criticality mode, etc. Processor affinity is employed to assign a thread to certain processor.

For partitioned scheduling, the partition is performed offline in advance and there is one managing thread for each processor. For global scheduling like fpEDF-VD-VPMC and VPMC-DP-Fair, only one managing thread is needed for all processors. For the VPMC-DP-Fair method, the managing thread needs to maintain and update the deadline partitions, which are rounded to the time unit.

TABLE II. TESTCASE CHARACTERISTICS FOR THE LINUX PROTOTYPING(THE UNIT OF EXECUTION TIME IS SECOND).

	Case 1				Case 2			
Task	χ_i	C_i^{LO}	C_i^{HI}	e_i	χ_i	C_i^{LO}	C_i^{HI}	e_i
τ_1	LO	0.76	0.47	20	LO	0.76	0.47	20
τ_2	HI	0.95	1.42	-	HI	0.95	1.42	-
τ_3	LO	0.67	0.21	0.5	LO	0.67	0.21	0.5
τ_4	HI	2.32	5.80	-	HI	2.32	5.80	-
τ_5	LO	1.65	0.98	5	LO	1.65	0.98	5
τ_6	HI	2.36	3.33	-	HI	2.36	3.33	-
τ_7	LO	1.40	0.71	8	LO	2.55	0.33	5
τ_8	HI	1.89	2.63	-	HI	4.10	5.10	-
τ_9	LO	0.76	0.27	5	LO	1.83	0.38	3
τ_{10}	LO	2.09	0.48	5	HI	1.07	2.31	-
τ_{11}	HI	0.47	0.73	-	LO	3.18	0.66	2
τ_{12}	-	-	-	-	HI	1.63	3.71	-

Two testcases are generated for the experiment in the Linux system. In the first case, all tasks are solving equations by the Newton-Raphson method. The second case is composed by tasks of both Newton-Raphson and the steepest decent method computing. For each case, tasks are randomly designated with low-criticality or high-criticality. We run these cases repeatedly to find the maximum execution time of each task. The WCET is obtained by adding safety margins to the measured maximum execution time. Since both Newton-Raphson and the steepest decent method are iterative algorithms, their imprecise computing is realized by relaxing the termination criterion. The precise computation, which has a tight termination criterion, also results in a small errors, which is negligible in comparison with that of imprecise computing. The imprecise computing errors from low-criticality tasks are obtained from the results of the prototyped implementation. The characteristics of the two cases are summarized in Tables II.

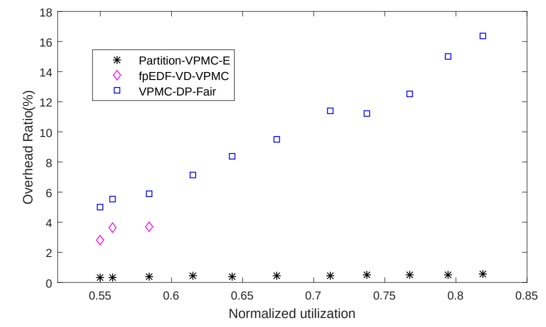

Fig. 7. Overhead ratio vs. utilization (overhead includes scheduling overhead and system overhead).

TABLE III. OVERHEAD COMPONENTS.

	Scheduling overhead	System overhead
Partition-VPMC-E	52%	48%
fpEDF-VD-VPMC	66%	34%
VPMC-DP-Fair	27%	73%

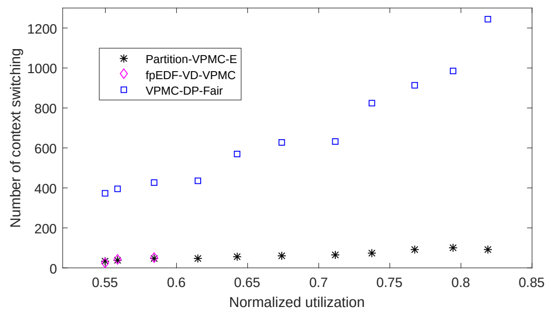

Fig. 8. Number of context switchings vs. utilization.

During the experiment, we vary the period T_i of each task τ_i to obtain different utilizations. For each scheduling method, its schedulability condition is checked offline. For only the utilization conditions where the check is successfully passed, the tasks are run in the Linux system, and each high-criticality job has an overrun probability of 0.9, so as to trigger system mode switching from low-criticality mode to high-criticality mode. In the first part of the Linux experiment, we investigate two components of overhead:

(1) Scheduling overhead: the time on scheduling job executions, execution monitoring, etc.

(2) System overhead: the time on context switching, job migration among processors, etc.

In Figure 7, we show the average results of overhead ratio, which is the ratio of system time expended on the overhead to the total computation time. The VPMC-DP-Fair has quite large overhead, as large as 16% when utilization is high. Partitioned scheduling has the lowest overhead, and its overhead does not change much as the utilization increases. One advantage of partitioned scheduling is that there is no inter-processor migration overhead. Since the fpEDF-VD-VPMC method has relatively low schedulability, it does not produce much data for this figure. The proportions of different overhead components are shown in Table III, which indicates that VPMC-DP-Fair has a relatively high ratio of system overhead due to frequent context switchings. In Figure 8, we compare the numbers of context switchings for different methods. A large number implies large overhead and the results are indeed correlated with the overhead ratio in Figure 7.

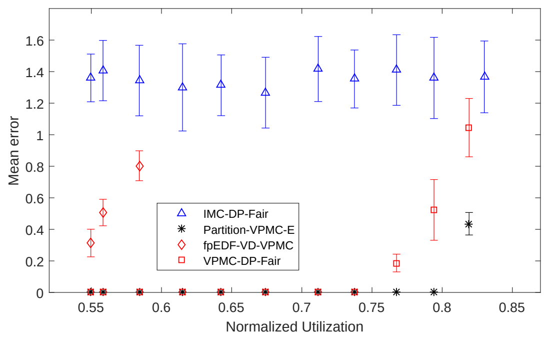

Fig. 9. Mean error (with standard deviation) versus utilization from Linux prototyping.

We further evaluated the errors under imprecise computing for different methods on the two cases. The average results are plotted in Figure 9. The largest errors correspond to IMC-DP-Fair, where all low-criticality tasks are continued with imprecise computing in high-criticality mode, while the smallest errors are from Partition-VPMC-E. The errors from fpEDF-VD-VPMC are not good mostly due to its conservative schedulability, which does not allow much utilization slack for converting imprecise computing to precise computing for low-criticality tasks. The errors from VPMC-DP-Fair are generally small, but greater than Partition-VPMC-E. This is because its large overhead leads to lower slack for precise computing.

VIII. CONCLUSIONS

The conventional mixed-criticality system model, despite its popularity, is controversial as it drops all low-criticality tasks in high-criticality mode. Recently, there are a few works for overcoming this drawback by continuing low-criticality tasks with imprecise computing or even precise computing. In this work, we develop such graceful degradation techniques for partitioned scheduling and fpEDF-VD scheduling on multiprocessors. The proposed techniques are evaluated with both software simulations and Linux prototyping where overhead is considered. The results show that the graceful degradation approach can significantly improve computing quality with little sacrifice on schedulability. When the overhead is considered, the proposed partitioned scheduling outperforms the previous approach of fluid-based scheduling.

Acknowledgement

This work was supported in part by the National Science Foundation(CCF-1525925 and CCF-1525749), and Office of Naval Research (N00014-18-1-2048).

References

[1] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degree of execution time assurance," in *Real-Time Systems Symposium (RTSS)*, pp. 239–243, IEEE, 2007.

[2] S. Baruah, V. Bonifaci, G. DAngelo, H. Li, A. Marchetti-Spaccamela, S. Van Der Ster, and L. Stougie, "The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems," in *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pp. 145–154, IEEE, 2012.

[3] H. Li and S. Baruah, "Global mixed-criticality scheduling on multiprocessors," in *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pp. 166–175, IEEE, 2012.

[4] 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.

[5] S. Baruah and Z. Guo, "Mixed-criticality job models: a comparison," Workshop on Mixed-Criticality Systems (WMC), 2015.

[6] A. Burns and R. I. Davis, "A survey of research into mixed criticality systems," *ACM Computing Surveys (CSUR)*, vol. 50, no. 6, p. 82, 2017.

[7] R. Ernst and M. Di Natale, "Mixed criticality systemsa history of misconceptions?," *IEEE Design and Test*, vol. 33, no. 5, pp. 65–74, 2016.

[8] H. Su and D. Zhu, "An elastic mixed-criticality task model and its scheduling algorithm," in *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pp. 147–152, IEEE, 2013.

[9] A. Burns and S. Baruah, "Towards a more practical model for mixed criticality systems," in *Workshop on Mixed-Criticality Systems*, 2013.

[10] S. Baruah, A. Burns, and Z. Guo, "Scheduling mixed-criticality systems to guarantee some service under all non-erroneous behaviors," in *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pp. 131–138, IEEE, 2016.

[11] D. Liu, J. Spasic, N. Guan, G. Chen, S. Liu, T. Stefanov, and W. Yi, "EDF-VD scheduling of mixed-criticality systems with degraded quality guarantees," in *Real-Time Systems Symposium (RTSS)*, pp. 35–46, IEEE, 2016.

[12] R. M. Pathan, "Improving the quality-of-service for scheduling mixed-criticality systems on multiprocessors," in *LIPIcs-Leibniz International Proceedings in Informatics*, vol. 76, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[13] J. W.-S. Liu, K.-J. Lin, W. K. Shih, A. C.-s. Yu, J.-Y. Chung, and W. Zhao, *Algorithms for Scheduling Imprecise Computations*, pp. 203–249. Springer US, 1991.

[14] J. Han and M. Orshansky, "Approximate computing: an emerging paradigm for energy-efficient design," in *Proceedings of IEEE European Test Symposium*, pp. 1–6, 2013.

[15] 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)*, pp. 41–52, IEEE, 2014.

[16] S. Baruah, A. Eswaran, and Z. Guo, "MC-Fluid: simplified and optimally quantified," in *Real-Time Systems Symposium (RTSS)*, pp. 327–337, IEEE, 2015.

[17] L. Sigrist, G. Giannopoulou, P. Huang, A. Gomez, and L. Thiele, "Mixed-criticality runtime mechanisms and evaluation on multicores," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2015 IEEE*, pp. 194–206, IEEE, 2015.

[18] S. Baruah, A. Burns, and R. Davis, "Response-time analysis for mixed criticality systems," in *Real-Time Systems Symposium (RTSS)*, pp. 34–43, IEEE, 2011.

[19] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 1, pp. 124–137, 2013.

[20] D. Mohapatra, V. K. Chippa, A. Raghunathan, and K. Roy, "Design of voltage-scalable meta-functions for approximate computing," in *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pp. 1–6, IEEE, 2011.

[21] R. Ye, T. Wang, F. Yuan, R. Kumar, and Q. Xu, "On reconfiguration-oriented approximate adder design and its application," in *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, pp. 48–54, IEEE Press, 2013.

[22] S. Ramanathan and A. Easwaran, "Utilization difference based partitioned scheduling of mixed-criticality systems," in *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pp. 238–243, IEEE, 2017.

[23] S. Baruah, "Optimal utilization bounds for the fixed-priority scheduling of periodic task systems on identical multiprocessors," *IEEE Transactions on Computers*, vol. 53, no. 6, pp. 781–784, 2004.

[24] S. Funk, G. Levin, C. Sadowski, I. Pye, and S. Brandt, "DP-Fair: a unifying theory for optimal hard real-time multiprocessor scheduling," *Real-Time Systems*, vol. 47, no. 5, pp. 389–429, 2011.