Bulletin of Networking, Computing, Systems, and Software – www.bncss.org, ISSN 2186-5140
Volume 13, Number 1, pages 1–7, January 2024

A Scheme Reducing Task Drops for Data Dependent Tasks on Mixed Criticality Systems

1st Reo Nagura
Graduate School of Science and Technology
Keio University
Kouhoku-ku, Yokohama-shi, Kanagawa, Japan, 223-8852
nagura@ny.ics.keio.ac.jp

2nd Nobuyuki Yamasaki
Graduate School of Science and Technology
Keio University
Kouhoku-ku, Yokohama-shi, Kanagawa, Japan, 223-8852
yamasaki@ny.ics.keio.ac.jp

***Abstract*—In embedded real-time systems, some tasks have varying execution times due to data dependency such as the number of input data from outside. Mixed criticality systems (MCS) can deal with these tasks by discarding non-critical tasks to obtain the computing resource when critical tasks overrun. However, there are cases in which the execution time of the task can be estimated as a form of function that has computation order depending on the algorithm inside it. Classic MCS model has difficulty dealing with this kind of task because it needs to estimate the execution time in a certain number of levels, two in most cases. We define a novel MC task model which has a worst case execution time as a function and propose a run-time task control scheme that selects the non-critical tasks which should be discarded by calculating the amount of utilization required by critical tasks overrun. We evaluate its performance in simulation and demonstrate its effectiveness.**

***Index Terms*—real-time systems, processor scheduling, mixed criticality systems**

I. Introduction

One of the recent trends in embedded real-time systems is the systems that integrate the components with different importance or criticality levels into one hardware platform. These systems are called Mixed Criticality Systems (MCS), and are supported on safety-critical systems such as avionics [1] systems.

Since the original concept of MCS was proposed [2], many studies have been published. In a typical MCS, tasks are classified into two criticality levels (high or low) based on whether violating a time constraint has a significant impact on the system. High-criticality tasks have multiple WCETs estimated at different pessimism levels, while low-criticality tasks have only one WCET. If all high-criticality tasks are completed within the optimistic WCET, the system remains in normal mode. However, if one of the high-criticality tasks continues to execute beyond the optimistic WCET (overrun), the system changes mode to a higher level and discards all low-criticality tasks, so that the overrun high-criticality task continues to run until the pessimistic WCET.

Therefore, MCS can handle tasks with varying WCET. For example, some tasks have data dependency such as the number and the size of external input data. A task maintains its normal state as long as the number of inputs is less than a predefined threshold value that is estimated to be the normal value. However, if the number of inputs exceeds the threshold value, the task enters an emergent state and may cause system overload. Such data-dependent tasks include multimedia processing which depends on image size, network processing which depends on the number of nodes, and data placement processing such as sorting. For such tasks, WCET is expressed as a function of the number of input data and can be estimated by internal algorithms or experimental measurements.

However, there are several problems when considering handling such data-dependent tasks in a general MCS.

First, in many existing studies, the WCET of a high-criticality task is estimated at a certain number of levels, in most cases two levels, even if it can be estimated by the WCET functional formula, resulting in the waste of processor utilization.

Second, as a fundamental problem in MCS, a typical MCS study uses system-level mode switching to discard all low-criticality tasks when one high-criticality task exceeds its optimistic WCET. This leads to a very low execution rate of low-criticality tasks and decreases the QoS of the whole system. This is because it is assumed that all high-criticality tasks overrun simultaneously and continue to run at pessimistic WCET for some time. However, this is not realistic because tasks that depend on different types of sensors and devices should overrun independently of each other.

To address these issues, we offer the following suggestions. First, we present the novel MC task model in which the WCET is a function of the number of input data. While the traditional MC task model is a non-clairvoyant task model that assumes that whether a task will overrun cannot be predicted until it overruns, this model is a kind of clairvoyant task model [3] that the number of input data is given at task release and its WCET can be calculated at that moment. This assumption is practical for data-dependent tasks because this type of task needs to obtain the size of its input from the I/O device before it is released from the scheduler.

Second, to avoid excessive degradation of QoS, we propose a scheme reducing the low-criticality task drops by selecting tasks to be discarded in the time slot required by the overrun high-criticality task. By introducing a task-level mode switch [4], we can handle high-criticality tasks that overrun independently of each other and low-criticality tasks that should

not be discarded at the same time. By considering the task model described above, the proposed method can calculate how many timeslots are required when high-criticality tasks overrun at release. Thus, it is possible to estimate the minimum number of jobs that should be dropped to meet the required resources. The virtual deadline of the Pfair (Proportionate fairness) scheduling algorithm is used to calculate at least how many timeslots each low-criticality task will use within a certain time range.

In summary, the contribution of this paper is below.

- We present the novel task model which has a functional WCET.
- We propose the task-selecting scheme to minimize the number of jobs that should be dropped by calculating required timeslots using the task model above.
- Our evaluation shows that our scheme improves the drop rate of low-criticality tasks.

II. Related Work

Since Vestal proposed the concept of MCS [2], a variety of research on MCS has been studied. The mainstream of MCS research has been to improve the scheduling algorithms that guarantee schedulablity while providing the mode switch mechanism of MCS. For example, EDF-VD (Earliest Deadline First Virtual Deadline) [5] has been studied in both uniprocessor and multiprocessor theoretical MCS research. EDF-VD is an extension of EDF [6] that gives higher priority to tasks with shorter deadlines. One of the difficulties of MCS scheduling is how to handle carry-over jobs that trigger mode switch by overrun. If, at the moment of overrun, the remaining execution time is longer than the time remaining to the deadline, the job will certainly overrun even if all low-criticality tasks are dropped. To avoid this problem, many studies in the EDF scheduling scheme use virtual deadlines that are shorter than the real deadlines to raise the priority of high-criticality tasks, and provide them as much margin as possible for overruns. However, EDF scheduling on multiprocessors is inefficient compared to optimal scheduling algorithms. In addition, the introduction of virtual deadlines results in tighter schedulable utilization and reduced task set acceptability. Several studies have attempted to remedy this drawback [7] [8] [9].

Another trend in MCS research is to solve the problem of practical situations. As mentioned earlier, conventional MCS triggers a mode switch after a single overrun and then drops all low-criticality tasks. In addition, some theoretical studies do not consider mode return from HI, assuming that overruns are extremely rare events. These are problems when using such MCS in real-world applications. Our study is of this kind and attempts to reduce the excessive dropping of low-criticality tasks.

There are some studies that reduce the task drops. Huang et al. proposed an offline mapping between low-criticality and high-criticality tasks using ICG (Interference Constraint Graph) [4], and Gu et al. presented an offline task grouping for low-criticality task drops [10]. As the online scheme, Lee et al. proposed an MC-ADAPT scheme that selects task drops using EDF-VD based runtime schedulablity analysis [11]. In our method, task drops are selected using an online algorithm.

III. Model

A. The overview of conventional MCS

Before presenting our scheme, we describe the conventional MCS. We assume the multiprocessor system with identical m cores, two system modes (LO and HI), and two task criticality levels (low and high).

Task Model. We consider an implicit-deadline periodic task system (denoted τ) of n MC tasks. Each of MC tasks τ_i is defined as $(T_i, C_i^{LO}, C_i^{HI}, \chi_i)$, where

- T_i: period
- C_i^L: WCET estimated in LO mode (LO-WCET)
- C_i^H: WCET estimated in HI mode (HI-WCET)
- χ_i: criticality level of task (low or high)

MC task model is characterized by two WCETs for each system mode: C_i^L is the normal execution time in LO mode, called LO-WCET; C_i^H is the emergency execution time in HI mode, called HI-WCET. As mentioned earlier, we have $C_i^L < C_i^H$ for high-criticality tasks and $C^H = 0$ for low-criticality tasks since LO-WCET is an optimistically estimated WCET, and HI-WCET is a pessimistically estimated WCET for high-criticality tasks, and low-criticality tasks are dropped in HI mode. For notational convenience, we refer to high-criticality tasks as hi-tasks, and low-criticalitys task as lo-tasks.

Utilization of each task τ_i is defined respectively in each system mode below.

$$\begin{cases} u_i^L = \dfrac{C_i^L}{T_i} & (1) \\ u_i^H = \dfrac{C_i^H}{T_i} & (2) \end{cases}$$

Utilization of the whole task system τ is also defined in each system mode below.

$$\begin{cases} U^L = U_{lo}^L + U_{hi}^L = \displaystyle\sum_{\tau_i|\chi_i=\text{low}} u_i^L + \sum_{\tau_i|\chi_i=\text{high}} u_i^L & (3) \\ U^H = U_{hi}^H = \displaystyle\sum_{\tau_i|\chi_i=\text{high}} u_i^H & (4) \end{cases}$$

HI mode utilization U^H does not have U_{lo}^H, since $C^H = 0$ holds for all lo-tasks.

System Behavior. Fig.1 shows the scheduling of the conventional MCS. Initially, the system operates in LO mode. All tasks are executed in LO mode at LO-WCET. When a hi-task is executed beyond LO-WCET (overrun), the system mode changes from LO to HI. At that moment, the running lo-task is interrupted and immediately discarded. In HI mode, the lo-task is not released and the hi-task can be executed in HI-WCET. As mentioned earlier, some theoretical studies do not consider the conditions for the mode transition from HI to LO, but here we assume that the HI mode can be returned if none of the hi-tasks overrun.

As for the scheduling algorithm, many studies adopt the earlier EDF-VD [5]. The details differ among the studies, but

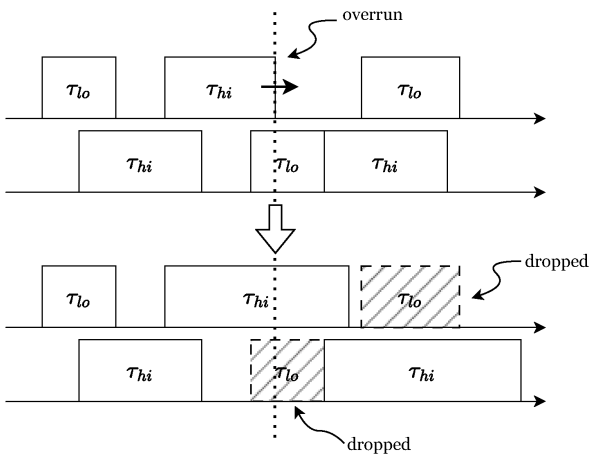


Fig. 1: The scheduling example of the conventional MCS. When a hi-task overruns, the system enters HI mode, and lo-tasks are immediately discarded and not released in HI mode.

in general, the virtual deadline $\hat{T}_i = x \cdot T_i (0 < x < 1)$ is assigned to every hi-task in LO mode, which increases their priority in EDF scheduling. The value of x can be calculated by using schedulability analysis both in LO and HI mode.

B. The design of proposed model

Next, we explain our proposed scheme, a Data-dependent MC (DMC) task model, and an online method reducing lo-task drops.

Target System. Similar to the conventional MCS described above, we consider a multiprocessor system with identical m cores and two task criticality levels. However, to address the problem described in the previous section, instead of the system-level mode switch, we introduce the task-level mode switch, described below. Thus, the system need not have the system mode; each task has its mode.

DMC Task Model. To efficiently handle data-dependent tasks whose execution time depends on the amount of external input data, we define a novel task model, the Data-dependent MC (DMC) task model. Each of DMC task τ_i is represented as $(T_i, C_i(n), l_i, h_i, \chi_i, \text{mode}_i)$, where

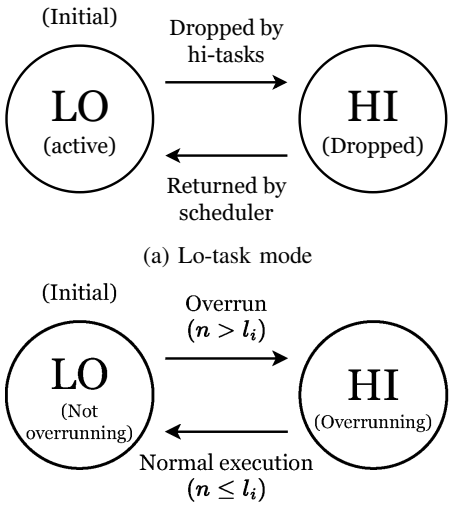
- T_i: period
- $C_i(n)$: WCET (function of the number of inputs n)
- l_i: the maximum number of inputs in LO mode
- h_i: the maximum number of inputs in HI mode
- χ_i: criticality level of task (high or low)
- mode_i: task-level mode (LO or HI)

The main feature of the DMC task model is that WCET $C_i(n)$ is a function of the number of input data , whereas in conventional MC, WCET C_i^L, C_i^H are fixed values for each system mode. The number of input data n is given at the release time of each job, and the job's WCET can be calculated simultaneously using $C_i(n)$.

Each $C_i(n)$ can be estimated with both the computational order of internal algorithms and experimental measurements. For example, the computational order of the bubble sort is generally $o(n^2)$. Thus, the execution time of the bubble sort task can be formulated as $A \cdot n^2 + B$, where A is the constant of proportionality, and B is the time of pre-processing and post-processing such as initialization and file manipulation sections. Each A and B is predicted from experimental measurements.

The parameters l_i, h_i are the acceptable limits of n in the task mode (LO, HI) described below. l_i is used in the condition of the task-level mode switch and $C_i(l_i)/T_i$ is used as a normal utilization of the task τ_i in the task set acceptance tests, performed before system startup. h_i represents the maximum utilization of the task, and we have at least $C_i(h_i) \leq T_i$ since the utilization of the task must not exceed 1. Note that since we assume only hi-tasks to be data-dependent, $C_i(n)$ for lo-tasks is independent of n, and l_i, h_i are meaningless.

Task-level Mode Switch. The DMC task model applies the task-level mode switching mechanism [4] rather than system-level mode-switching. Unlike previous studies, the tasks should be managed individually in practical situations, since all hi-tasks do not overrun simultaneously, and not all lo-tasks should not be abandoned by a single overrun of the hi-tasks. How task-level modes work is illustrated in Fig.2. Mode switching is performed by the scheduler.



(a) Lo-task mode

(b) Hi-task mode

Fig. 2: Task-level mode in each task criticality. For lo-tasks, the mode indicates whether the task is currently active or dropped. For hi-tasks, the mode indicates whether the task is currently overruns ($n > l_i$) or not ($n \leq l_i$).

Based on the above, the system behavior is as follows.

- Initially, all tasks operate in LO mode.
- When a hi-task overruns, switch its mode from LO to HI and switch the mode of some lo-tasks from LO to HI.
- When an overrunning hi-task finishes, switch its mode from HI to LO.
- At the next release time of dropped lo-tasks, switch their modes from HI to LO.

Propotionate Fairness Scheduling. In our model, Propotionate-fairness (Pfair) scheduling [12] [13] is applied for

both high schedulability and the algorithm of our scheme. Pfair scheduling is one of the optimal multiprocessor scheduling algorithms, which guarantees that every task set within the maximum utilization of the system can be scheduled without missing deadlines. The Pfair algorithm schedules each task to run proportionally from its release time to its deadline, subject to the following conditions.

$$S(\tau_i, t) = \begin{cases} 1 & \tau_i \text{ executed in } [t, t+1) \\ 0 & \tau_i \text{ not executed in } [t, t+1) \end{cases} \quad (5)$$

$$lag(\tau_i, t) = U_i \cdot t - \sum_{u=0}^{t-1} S(\tau_i, u) \quad (6)$$

$$\forall \tau_i, t :: -1 < lag(\tau_i, t) < 1 \quad (7)$$

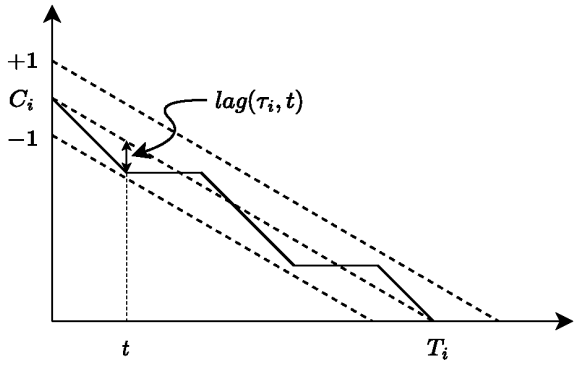


Fig. 3: Pfair scheduling. The distance to the line connecting C_i and T_i is $lag(\tau_i, t)$. $lag(\tau_i, t)$ must be in the range $(-1, 1)$.

Fig.3 shows the meaning of these conditions. Scheduling that satisfies condition (7) is defined as to be Pfair. Then Pfair scheduling is proved to be optimal scheduling. To schedule tasks as Pfair, each task τ is divided as a union of the subtasks to be executed for 1 time unit. The j_{th} subtask τ^j of the task τ_i has its own virtual release time $\hat{r_i^j}$ and virtual deadline $\hat{d_i^j}$.

$$\begin{cases} \hat{r_i^j} = \lfloor \dfrac{j-1}{U_i} \rfloor = \lfloor \dfrac{(j-1) \cdot T_i}{C_i} \rfloor & (8) \\ \hat{d_i^j} = \lceil \dfrac{j}{U_i} \rceil = \lceil \dfrac{j \cdot T_i}{C_i} \rceil & (9) \end{cases}$$

By scheduling these subtasks based on EDF manner using virtual parameters above, the scheduling is proved to be Pfair.

Although Pfair scheduling offers the highest schedulability, it has highly frequent context switches as the trade-off, which results in huge overhead for saving context information, and task migrations in a multiprocessor. Some studies attempt to improve them [14] [15] [16].

In addition, since the value of WCET directly affects the number of subtasks and their respective timing constraints, Pfair scheduling does not work well with conventional MCS, which runs up to LO-WCET and then rapidly increases the value of WCET. However, in this model, Pfair scheduling is applied because it assumes that WCET can be calculated from the number of input data at task release and does not require a virtual deadline mechanism such as EDF-VD.

IV. Scheme Reducing Task Drops

In this section, we describe the main topic of our study, a scheme for reducing task drops. Our approach is performed at the release of each hi-task job.

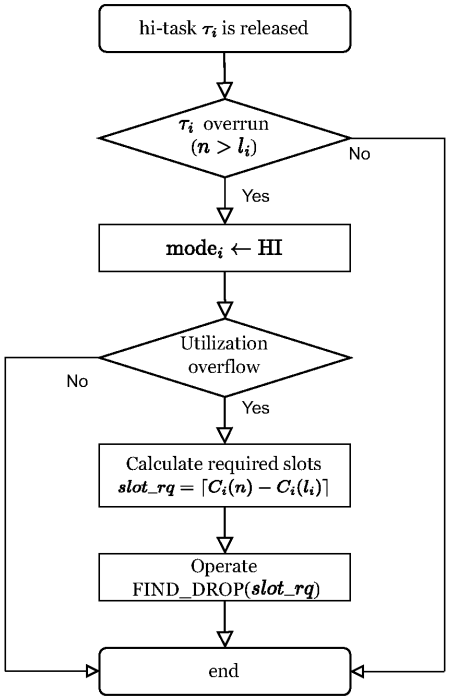


Fig. 4: Operation in hi-tasks release time

Fig.4 illustrates the operational flow of this scheme. When the hi-task τ_i is released at timing t, given the number of inputs n. Whether τ_i overruns is predicted by the condition $n > l_i$. If $n \leq l_i$, τ_i runs with normal execution time, and no mode switch occurs. Otherwise, τ_i enters the emergent state and mode$_i$ switches from LO to HI. However, since Pfair scheduling is applied in this model, there is no need to drop tasks as long as the total utilization does not exceed the number of cores. This can be calculated by the following equation.

$$(\text{current utilization}) + \frac{C_i(n) - C_i(l_i)}{T_i} > (\#\text{ of cores}) \quad (10)$$

When the condition (10) is satisfied, some lo-tasks need to be dropped due to the overrun of the hi-tasks. In the conventional MCS, once a hi-task overruns, the system-level mode switches unconditionally, resulting in an excessive number of lo-tasks being dropped. To avoid this problem, we consider selecting the minimum number of lo-tasks jobs to obtain the time slots required for overruns. Therefore, the next step is to calculate $C_i(n) - C_i(l_i)$ to obtain the required number of time slots $slots\_rq$. Then, we consider finding lo-task jobs that will run for at least $slots\_rq$ by the algorithm FIND_DROP.

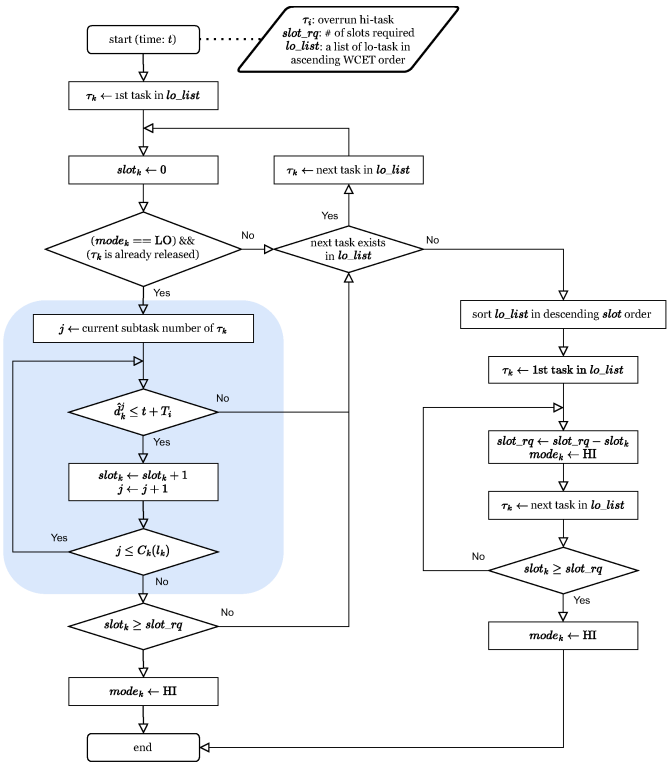


Fig. 5: Flowchart of FIND_DROP algorithm

Fig.5 illustrates the FIND_DROP algorithm. The approach of this algorithm is pretty simple, making a room in $[t, t+T_i)$ for hi-task τ_i to overrun. To do that, the scheduler needs to know how much time slots each lo-task will spend in $[t, t+T_i)$. As a premise, this algorithm requires a list of lo-tasks in ascending WCET order lo_list and the number of required slots $slots\_rq$ as an argument. First, for the current job of each lo-task in the lo_list, unless it has been already dropped nor it has not been released yet, how many subtask windows exist in $[t, t+T_i)$ is calculated (the blue part of Fig.5).

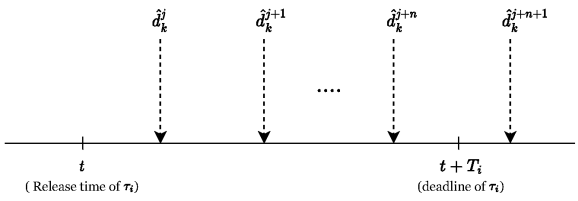


Fig. 6: Time slot calculation using virtual deadline of subtasks

Fig.6 shows how to calculate the number of subtask windows in $[t, t+T_i)$. Unlike other scheduling algorithms, in Pfair scheduling, how each task will be scheduled can be predicted in the form of the subtask window. Thus, each task is executed in $[t, t+T_i)$ with at least as many subtasks in the range. In the case of τ_k in Fig.6, the deadline of the j_{th} subtask to the deadline of the $(j+n)_{\text{th}}$ subtask lies within $[t, t+T_i)$, so the current job of τ_k will execute at least n time slots within this range. Since the deadline for $(j+n+1)_{\text{th}}$ subtask is after $t+T_i$, the subtask may but is not guaranteed to be executed by $t+T_i$.

When a $slot$ of some subtask τ_k^j is found to be able to fill the $slot\_rq$ during this operation, the mode of τ_k^j (mode$_k$) is changed from LO to HI, and then the job is discarded until the next job is released.

If there is no lo-task to satisfy the $slot\_rq$ after searching every task in lo_list, the operation proceeds to the right side of Fig.5. As the next step, the lo_list is sorted in the descending order of the $slot$ calculated in the previous step. Then, the following steps are repeated until a combination of lo-tasks that fills $slot\_rq$ is found.

1) Drop the first task in lo_list, remove it from lo_list, and subtract its $slot$ from $slot\_rq$.
2) Search the remaining tasks in lo_list.
3) If there is a task whose $slot$ satisfies condition $slot \geq slot\_rq$, drop that task, and then the operation ends.
4) Otherwise, repeat from 1).

As mentioned earlier, the FIND_DROP algorithm is only performed when the the task set utilization overflows due to overruns. Therefore, there should be enough lo-tasks to be dropped for $slot\_rq$.

V. Evaluation

In this section, we evaluate the performance of the DMC task model and FIND_DROP algorithm compared to the baseline model that applies as many of the features of the conventional MCS as possible. Table I briefly shows the difference between baseline model and proposed method.

TABLE I: Baseline model and proposed method

	Baseline	Proposed method
Task model	MC	DMC
Mode switch	system-level	task-level
Overrun detection	when tasks overrun	when tasks are released
tasks dropped	all lo-tasks	selected lo-tasks
scheduling algorithm	Pfair	Pfair

A. Targeted system

Although the evaluation is performed in a simulation, our target architecture is Responsive Multithreaded Processor (RMTP) [17], which is 8-way SMT (simultaneous multithread) processor for distributed real-time systems. Therefore, we assume the number of cores m is 7 (kernel occupies 1 core).

B. Task Generation

Task sets are generated with the following configuration.

- Utilization per core U_{ave}: each 5% in [40%, 110%] (with $\pm 1\%$ error)
- Task sets per utilization: 100
- Tasks per task set: 14 (high:7, low: 7)
- Utilization per task: each 5% in [15%, 60%]

Utilization per core U_{ave} is defined as below.

$$U_{ave} = \frac{\sum_{\tau_i \in \tau} C_i(l_i)/T_i}{m} \tag{11}$$

By nature, task sets that exceed 100% utilization are not accepted in acceptance tests, but are used to clearly demonstrate the schedulability of Pfair scheduling.

As described in model section, since lo-tasks have no data-dependency, the WCET function $C_i(n)$ of lo-tasks returns the fixed value generated randomly from $[1, 30]$ before system runs.

Parameters for Data-Dependent Tasks. We offers the parameters measured in RMTP for data-dependent hi-tasks. We measured the WCET of qsort benchmark in mibench [18] while giving each 50 in $[50, 1000]$ as the number of inputs (Fig.7). Since the computational order of the quick-

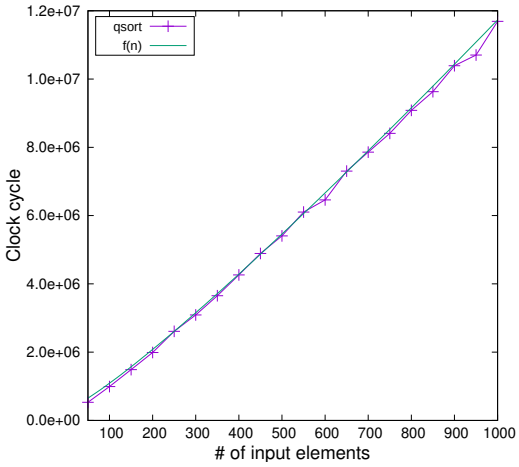


Fig. 7: The execution clock cycles of qsort with the number of inputs in $[50, 1000]$, running 7 threads in parallel.

sort is generally $o(n \log n)$, we defined the function $f(n)$ as $A \cdot n \cdot \log n + B$, and then we approximates the value of A and B using least squares method. We further define $C_i(n)$ below, where TICK = (Clock cycle per system tick) − (Scheduler overhead per system tick).

$$C_i(n) \leq \left\lceil \frac{f(n)}{\text{TICK}} \right\rceil \tag{12}$$

As a result, we get the followig value for each parameter.

- A: 1,144
- B: 330,606
- TICK: 250,000

Note that A and B are the value after adding the positive error of least squares fitting to make the equation (12) an inequality.

The value $C_i(l_i)$ for each hi-task is randomly generated in $[5, 30]$, and then l_i is calculated using C_i. The number of inputs n is randomly generated at each release time of hi-tasks job in $[0, 1000]$ with the normal distribution shown in Fig.8. The maximum number of inputs in HI mode is h_i where $C_i(h_i)/T_i = 0.8$.

C. Results and Consideration

For notational convenience, we denote the baseline model as MC and the proposed method as DMC. The duration of simulation is 5,000 system ticks for each task set.

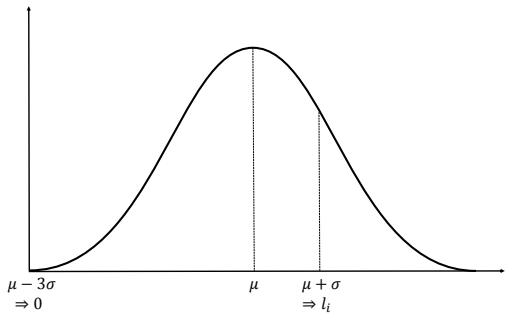


Fig. 8: The random value by normal distribution

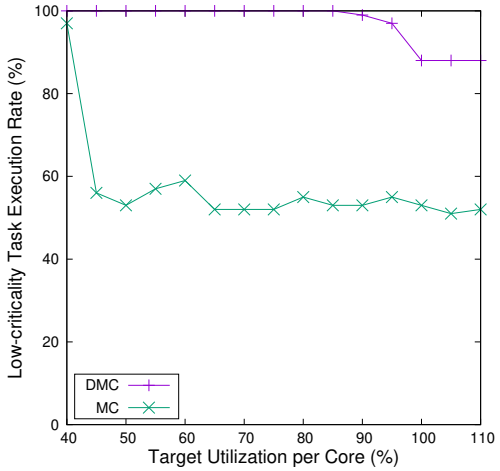


Fig. 9: Execution rate of lo-tasks

Execution Rate of Lo-tasks. Fig.9 shows the execution rate of lo-tasks defined as follows.

$$\frac{(\text{\# of lo-task jobs finished})}{(\text{\# of lo-task jobs released})} \tag{13}$$

The execution rate is decreased by task drops due to hi-task overruns. In case of DMC model, up to 80% utilization, there is no drop and the execution rate remains in 100% because the increased utilzation due to hi-task overruns does not cause the overflow in system utilization. From 85% utilization, task drops begin, but the execution rate remains around 90% by our scheme. In MC model, on the other hand, task drops start from 40% utilization, and execution rate remains around 60%. This is because all lo-tasks are dropped every time a hi-task overruns and the expected utilization U^H in HI mode exceeds the limits of system utilization. Therefore, lo-tasks drop rate in MC model is not related to the utilization of the task set, but to the frequency of hi-task overruns.

Scheduling Success Rate. Fig.10 is the result of scheduling success rate defined as follows.

$$\frac{(\text{\# of task sets finished without deadline misses})}{(\text{\# of task sets: 100})} \tag{14}$$

Note that the dropped jobs of lo-tasks are not included in the number of deadline misses. Due to the high schedulability

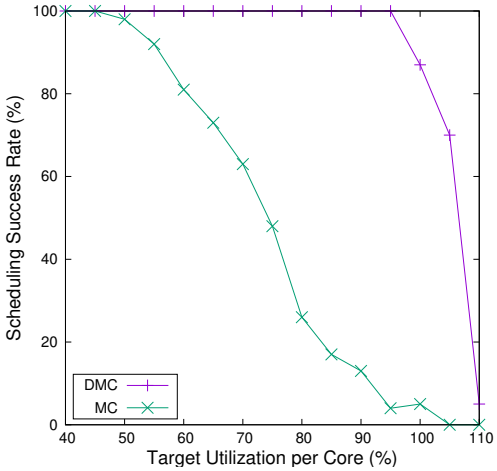


Fig. 10: Scheduling success rate

of Pfair scheduling, DMC model succeeds in scheduling up to 95% utilization, and rapidly fails when utilization exceeds 100%. The reason why deadline misses occur in task sets with 100% utilization may be due to the existence of task sets with 101% utilization, since 1% of error is allowed in the task set generation process. In MC model, however, deadline misses occur from 45% utilization even in Pfair scheduling. This is because the virtual parameters of subtasks are calculated using HI-WCET even in LO mode, and the apparent utilization of hi-tasks is 0.8 even in LO mode. As mentioned earlier, in Pfair scheudling, a sudden increase in the value of WCET is likely to cause deadline misses, so the value of WCET should be set to HI-WCET even if it does not overrun.

In summary, our method performs well both in the execution rate of lo-tasks and the scheduling success rate. A possible trade-off is the computational overhead, but the large difference in results is also due to the lack of sophistication in the baseline model design.

VI. Conclusion and Future Work

In conventional mixed criticality systems, excessive task drops caused by a single overrun of a high-criticality task reduce the execution rate of low-criticality tasks. We define a novel DMC task model that focuses on the data dependence of real-time tasks and propose a scheme reducing task drops by calculating the required utilization of overrunning tasks. Our evaluation shows the advantages of our scheme in terms of execution rate of low-criticality tasks and scheduling success rate. However, this method relies on the accuracy of the WCET function, which may cause problems when this method is applied to actual applications: if the value of the WCET function is smaller than the actual execution time, unnecessary task drops will occur. Conversely, if the value of the WCET function is greater than the actual execution time, deadline misses due to Pfair scheduling will occur.

As future work, we investigate how to maintain the accuracy of WCET function, and evaluate our scheme on real hardware.

References

[1] P. Prisaznuk, "Integrated modular avionics," in *Proceedings of the IEEE 1992 National Aerospace and Electronics Conference (NAECON) 1992*, 1992, pp. 39–45 vol.1.

[2] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, 2007, pp. 239–243.

[3] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie, "Scheduling real-time mixed-criticality jobs," *IEEE Transactions on Computers*, vol. 61, no. 8, pp. 1140–1152, 2012.

[4] P. Huang, P. Kumar, N. Stoimenov, and L. Thiele, "Interference constraint graph — a new specification for mixed-criticality systems," in *2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA)*, 2013, pp. 1–8.

[5] 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 *2012 24th Euromicro Conference on Real-Time Systems*, 2012, pp. 145–154.

[6] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, p. 46–61, jan 1973.

[7] X. Gu and A. Easwaran, "Efficient schedulability test for dynamic-priority scheduling of mixed-criticality real-time systems," *ACM Trans. Embed. Comput. Syst.*, vol. 17, no. 1, nov 2017.

[8] H. Baek and K. Lee, "Contention-free scheduling for mixed-criticality multiprocessor real-time system," *Symmetry*, vol. 12, no. 9, 2020.

[9] N. JUNG, H. BAEK, D. LIM, and J. LEE, "Incorporating zero-laxity policy into mixed-criticality multiprocessor real-time systems," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E101.A, no. 11, pp. 1888–1899, 2018.

[10] X. Gu, A. Easwaran, K.-M. Phan, and I. Shin, "Resource efficient isolation mechanisms in mixed-criticality scheduling," in *2015 27th Euromicro Conference on Real-Time Systems*, 2015, pp. 13–24.

[11] J. Lee, H. S. Chwa, L. T. X. Phan, I. Shin, and I. Lee, "Mc-adapt: Adaptive task dropping in mixed-criticality scheduling," *ACM Trans. Embed. Comput. Syst.*, vol. 16, no. 5s, sep 2017.

[12] S. Baruah, "Fairness in periodic real-time scheduling," in *Proceedings 16th IEEE Real-Time Systems Symposium*, 1995, pp. 200–209.

[13] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, "Proportionate progress: A notion of fairness in resource allocation," in *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, ser. STOC '93. New York, NY, USA: Association for Computing Machinery, 1993, p. 345–354.

[14] S. Baruah, J. Gehrke, and C. Plaxton, "Fast scheduling of periodic tasks on multiple resources," in *Proceedings of 9th International Parallel Processing Symposium*, 1995, pp. 280–288.

[15] J. Anderson and A. Srinivasan, "Early-release fair scheduling," in *Proceedings 12th Euromicro Conference on Real-Time Systems. Euromicro RTS 2000*, 2000, pp. 35–43.

[16] G. Levin, S. Funk, C. Sadowski, I. Pye, and S. Brandt, "Dp-fair: A simple model for understanding optimal multiprocessor scheduling," in *2010 22nd Euromicro Conference on Real-Time Systems*, 2010, pp. 3–13.

[17] S. Nakabeppu, Y. Ide, M. Takahashi, Y. Tsukahara, H. Suzuki, H. Shishido, and N. Yamasaki, "Space responsive multithreaded processor (srmtp) for spacecraft control," in *2020 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS)*, 2020, pp. 1–3.

[18] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No.01EX538)*, 2001, pp. 3–14.