Journal of Systems Architecture 129 (2022) 102613

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Schedulability Performance Improvement via Task Split in Real-Time Systems

Jinkyu Lee

Dept. of Computer Science and Engineering, Sungkyunkwan University (SKKU), Republic of Korea

ARTICLE INFO

Keywords:
Real-time systems
FPS (fixed priority scheduling),
RTA (Response Time Analysis)
Multiprocessor Platform

ABSTRACT

In this paper, we focus on splitting a task into multiple sub-tasks to improve schedulability performance for a set of tasks subject to timing constraints. Targeting FPS (Fixed Priority Scheduling) as a scheduling algorithm and RTA (Response Time Analysis) as a schedulability analysis on a multiprocessor platform, we develop a systematic method to utilize the task split, which addresses (i) how to apply the task split without violating the timing requirement of the original tasks and (ii) how to apply each task so as to make an unschedulable task set to be schedulable. Our simulation results demonstrate that RTA with the method finds 16.1%–19.4% (on average) additional task sets that were not proven schedulable by the vanilla RTA.

1. Introduction

Timely execution of a set of tasks is the main focus of real-time systems. To achieve this, numerous studies have paid attention to how to determine the execution order of tasks (called scheduling algorithm) and how to guarantee no deadline miss of tasks before run-time (called schedulability analysis), e.g., [1–5]. However, there is still a room for further schedulability performance improvement (i.e., providing timing guarantees for more task sets), as most schedulability analyses are only sufficient (not necessary). For example, there has been no closed-form exact (i.e., sufficient *and* necessary) schedulability analysis even for the most fundamental scheduling algorithms such as global[1] FPS (Fixed-Priority Scheduling) and EDF (Earliest Deadline First) on a multiprocessor platform.

In this paper, we develop a simple yet effective way to improve schedulability performance, which splits a task into multiple sub-tasks. While the task split can be generally applicable to most (if not all) existing scheduling algorithms and their schedulability analyses, the main challenges are (CH1) how to apply the task split without violating the timing requirement of the original tasks, and (CH2) how to split each task so as to make an unschedulable task set to be schedulable. Targeting FPS as a scheduling algorithm and RTA (Response Time Analysis) as a schedulability analysis on a multiprocessor platform, we develop a systematic method that addresses CH1 and CH2. Note that there have been some studies that utilize the task split [6–8], which focus on how to efficiently split a task (to be executed in multiple processors while each of other tasks is executed in a designated processor) in order to efficiently occupy each processor in semi-partitioned scheduling. Therefore, they are totally different from the proposed method, which splits multiple tasks in global scheduling.

To verify the effectiveness of the proposed method, we generate a number of task sets for varying number of processors, and compare the number of task sets proven schedulable by the vanilla RTA [3] and that by RTA with the method. Our simulation results show that RTA with the method significantly improves the schedulability performance in all settings; it covers 16.1%–19.4% (on average) additional task sets that were not proven schedulable by the vanilla RTA.

This paper makes the following contributions:

- Developing a systematic method to utilize the task split for schedulability improvement of global multiprocessor scheduling, and
- Applying the method to RTA for FPS, and demonstrating its effectiveness through simulation results.

System model. In this paper, we consider a set of sporadic real-time tasks (denoted by τ), in which each task $\tau_i \in \tau$ is represented by T_i (the period) and C_i (the worst-case execution time). A job of τ_i, once released at t, should finish its execution (that amounts to at most C_i) until its absolute deadline $t + T_i$, and the release times of two consecutive jobs of τ_i are separated by at least T_i. Let the length of the quantum be one time unit; all task parameters $\{T_i\}$ and $\{C_i\}$ are natural numbers. We consider a multiprocessor platform consisting of m identical processors. We target FPS (Fixed-Priority Scheduling) in which each task has its own pre-defined task-level priority; at any time,

E-mail address: jinkyu.lee@skku.edu.

[1] By global scheduling, we mean that a task can be executed in any processor as opposed to partitioned scheduling. From now on, we will omit the term "global" when it is not necessary to refer.

https://doi.org/10.1016/j.sysarc.2022.102613
Received 31 March 2022; Received in revised form 27 May 2022; Accepted 8 June 2022
Available online 15 June 2022

FPS chooses to execute (up to) m jobs whose invoking tasks' priorities are the highest. Let $\mathtt{HP}(\tau_k)$ denote a set of tasks in τ, whose priority is higher than τ_k.

2. Schedulability improvement via task split

In this section, we target RTA (Response Time Analysis) for FPS on a multiprocessor platform [3], and develop a task split method that improves the schedulability performance of FPS associated with RTA.

The basic structure of RTA for FPS on a multiprocessor platform [1,2] is as follows. We target an interval of length ℓ, which starts from the release time of the job of τ_k of interest. We define $I_{k\leftarrow i}(\ell)$ as a cumulative length of intervals in which jobs of τ_i execute while the job of τ_k of interest cannot in the target interval of length ℓ. Considering the job of τ_k of interest cannot execute only when m other higher-priority jobs are executed, the following inequality is a sufficient condition for the job of τ_k of interest to be schedulable [1,2]:

$$\sum_{\tau_i\in\mathtt{HP}(\tau_k)} \min\big(I_{k\leftarrow i}(\ell), \ell - C_k + 1\big) < m\cdot(\ell - C_k + 1). \tag{1}$$

Then, the remaining step is to upper-bound $I_{k\leftarrow i}(\ell)$ for all jobs of τ_k. By definition, $I_{k\leftarrow i}(\ell)$ can be upper-bounded by the maximum execution of jobs of τ_i in an interval of length ℓ, which is calculated by $W_i(\ell)$ [1,2] as follows.

$$W_i(\ell) = N_i(\ell)\cdot C_i + \min\big(C_i, \ell + T_i - C_i - S_i - N_i(\ell)\cdot T_i\big), \tag{2}$$

where $N_i(\ell) = \left\lfloor \frac{\ell+T_i-C_i-S_i}{T_i} \right\rfloor$. Note that S_i denotes a slack value of τ_i, meaning that every job of τ_i finishes its execution within $(D_i - S_i)$ from its release; we will explain how to calculate S_i later.

The calculation of $W_i(\ell)$ considers the case where the amount of execution of jobs of τ_i in an interval of length ℓ is maximized as follows. The first job of τ_i starts its execution as late as possible, and the following jobs of τ_i executes as early as possible; in addition, the interval of interest of length ℓ starts when the first job starts its execution. Then, $N_i(\ell)$ calculates the number of jobs of τ_i whose deadline is within the interval, each contributing the full execution C_i. On the other hand, the second term of Eq. (2) calculates the amount of execution of the carry-out[2] job of τ_i in the interval of length ℓ. Then, if we replace $I_{k\leftarrow i}(\ell)$ with $W_i(\ell)$, Eq. (1) becomes a schedulability analysis for FPS [1,2].

In [9], it has been revealed that a critical instant of a job of τ_k under FPS on a multiprocessor platform occurs only when there are at most $(m-1)$ higher-priority carry-in[3] jobs in the interval that starts at the release time of the job of τ_k. In the case for a task τ_i not to have any carry-in job in an interval of length ℓ, the maximum execution of jobs of τ_i in the interval is calculated by $E_i(\ell)$ [3] as follows.

$$E_i(\ell) = \left\lfloor \frac{\ell}{T_i} \right\rfloor \cdot C_i + \min\left(C_i, \max\left(0, \ell - \left\lfloor \frac{\ell}{T_i} \right\rfloor \cdot T_i\right)\right), \tag{3}$$

which is no larger than $W_i(\ell)$. The calculation of $E_i(\ell)$ considers the case where every job starts its execution as early as possible, and the interval of interest of length ℓ starts when the first job's execution starts.

Instead of applying $W_i(\ell)$ as an upper bound of $I_{k\leftarrow i}(\ell)$ in Eq. (1), we can apply $E_i(\ell)$ for non-carry-in jobs, yielding a tighter schedulability analysis for FPS as follows.

Lemma 1 (*In [3]*). *A task set τ is schedulable by FPS on an m-processor platform, if every $\tau_k \in \tau$ has $C_k \le \ell \le T_k$ that satisfies Eq. (4).*

$$\Delta_k(\ell) + \sum_{\tau_i\in\mathtt{HP}(\tau_k)} \min\big(E_i(\ell), \ell - C_k + 1\big) < m\cdot(\ell - C_k + 1), \tag{4}$$

where $\Delta_k(\ell)$ is the sum of $(m-1)$ largest $\big(W_i(\ell), \ell - C_k + 1\big) - \big(E_i(\ell), \ell - C_k + 1\big)$ among $\tau_i \in \mathtt{HP}(\tau_k)$.

Proof. In order to apply at most $(m-1)$ higher-priority carry-in jobs for the critical instant of FPS on a multiprocessor platform, we add the interference as $\min(E_i(\ell), \ell - C_k + 1)$ for each higher-priority tasks, and then add the difference between $\min(W_i(\ell), \ell - C_k + 1)$ and $\min(E_i(\ell), \ell - C_k + 1)$ for $(m-1)$ higher-priority tasks whose difference is the largest. Then, for any combination of at most $(m-1)$ higher-priority carry-in tasks, the amount of interference is upper-bounded by the LHS of Eq. (4), which is derived by the condition of the critical instant of FPS on a multiprocessor platform and the meaning of $W_i(\ell)$ and $E_i(\ell)$. The detailed proof is given in [3]. □

Then, RTA for FPS operates as follows [1]. Initially, S_i for every task is set to zero. Then, Eq. (4) is checked from the highest-priority task to the lowest-priority task.[4] If a task τ_k is schedulable (i.e., Eq. (4) holds), we calculate an upper bound of the response time of τ_k (denoted by R_k), which is the sum of C_k and the LHS (left-hand-side) of Eq. (4) divided by m; also, if $R_k < T_k$ holds, we set S_k to $(T_k - R_k)$. Finally, the task set τ is deemed schedulable, if Eq. (4) holds for every task. Otherwise, we repeat the whole process with update slack values $\{S_k\}$; the repetition ends with the task set deemed schedulable, or with no more slack update (yielding the task set deemed unschedulable).

Then, the problem to be solved in this paper is as follows. For given $\{\tau_i \in \tau\}$ and their priorities for FPS, we find how to split each task $\tau_i \in \tau$ into $\tau'_i \in \tau'$, which satisfies the following statement: if we schedule $\{\tau'_i \in \tau'\}$ according to FPS with their original task priorities, RTA for FPS guarantees that there is no job deadline miss of $\{\tau'_i \in \tau'\}$, meaning that no job deadline miss of $\{\tau_i \in \tau\}$ if we apply the schedule for $\{\tau'_i \in \tau'\}$.

We outline the proposed method for the problem as follows. First, we exemplify how Eq. (4) in RTA for FPS is changed if we split tasks, giving an intuition for the task split (in Example 1). Second, we develop how to split each task even when its parameters are not a multiple of the quantum. Third, we derive a relationship between the schedulability of the original task set and that of the split task set (in Theorem 1). Finally, we develop a systematic way to split all tasks in the target task set in order to make the set schedulable (in Algorithm 1).

Now, we investigate how the task split (also known as the period transformation [10]) impacts on the schedulability of each task set. As a first step, we define a split factor of τ_i as a natural number α_i. That is, α_i transforms τ_i into τ'_i in which the period and the worst-case execution time of τ_i will be divided by α_i; therefore, $\alpha_i = 1$ means no task split. For example, $\tau_1(T_1 = 8, C_1 = 4)$ will be transformed into $\tau'_1(T'_1 = 4, C'_1 = 2)$ by $\alpha_1 = 2$; it is easily checked that the timing guarantee of τ'_1 implies that of the original task τ_1. We will explain how to divide the parameters by α_i, if they are not a multiple of α_i. The following example explains how Eq. (4) changes if we split tasks.

Example 1. Consider a set of three tasks $\tau_1(T_1 = 8, C_1 = 4)$, $\tau_2(8, 4)$ and $\tau_3(12, 6)$ on a two-processor platform. τ_1 and τ_3 have the highest and the lowest priority, respectively.

Since $E_1(12) = E_2(12) = 8$ and $\min(E_1(12), 12 - 6 + 1) = \min(E_2(12), 12 - 6 + 1) = 7$ hold, the LHS of Eq. (4) is the same as the RHS (right-hand-side); therefore, τ_3 cannot satisfy Eq. (4).

[2] A job is said to be *carry-out* in an interval, if the job is released before the end of the interval and has remaining execution at the end of the interval.

[3] A job is said to be *carry-in* in an interval, if the job is released before the beginning of the interval and has remaining execution at the beginning of the interval.

[4] How to efficiently find ℓ that satisfies the inequality is explained in [1], which is the same as that for the response time calculation on a uniprocessor platform.

On the other hand, suppose that we split τ_1 and τ_2 by two (i.e., $\alpha_1 = 2$ and $\alpha_2 = 2$), yielding $\tau'_1(T'_1 = 4, C'_1 = 2)$ and $\tau'_2(4, 2)$. Since $E'_1(12) = E'_2(12) = 6$ and $\min(E_1(12), 12-6+1) = \min(E_2(12), 12-6+1) = 6$ hold, the LHS of Eq. (4) is strictly smaller than the RHS; therefore, τ_3 satisfies Eq. (4).

As shown in the example, a task split of τ_i can reduce interference of τ_i to other tasks; this is because, $E_i(\ell)$ and $W_i(\ell)$ decrease or remain the same, if we divide T_i and C_i by the same natural number larger than 1. On the other hand, a task split of τ_i is not favourable to the schedulability of τ_i itself; this is because, the ratio between $E_k(\ell)$ and $E_k(\ell/\alpha_k)$ (or $W_k(\ell)$ and $W_k(\ell/\alpha_k)$) for $\tau_k \neq \tau_i$ is no smaller than α_k. Motivated by the example, we need to address the following two challenges to make the task split improve the schedulability performance: (CH1) how to apply the task split if T_i/α_i or C_i/α_i is not a multiple of the quantum, and (CH2) how to set α_i for every task to make τ schedulable.

For CH1, we apply the sustainability property [11], which holds for most schedulability algorithms (including) FPS with respect to increasing T_i and decreasing C_i. To this end, we consider three task sets: the original task set τ, a new task set τ'' with slight adjustment of $\{T_i\}$ and $\{C_i\}$, and another new task set τ' by applying the task split to τ''. In each $\tau''_i \in \tau''$, T''_i is set to $\lfloor \frac{T_i}{\alpha_i} \rfloor \cdot \alpha_i$ and C''_i is set to $\lceil \frac{C_i}{\alpha_i} \rceil \cdot \alpha_i$. Also, in each $\tau'_i \in \tau'$, T'_i is set to $\lfloor \frac{T_i}{\alpha_i} \rfloor$, and C'_i is set to $\lceil \frac{C_i}{\alpha_i} \rceil$.

Then, the three task sets have the following relationship. First, if τ' is schedulable, executing τ'' as if it were τ' does not yield any job deadline miss of τ''. This is because, every T''_i and C''_i are a multiple of T'_i and C'_i, respectively. Second, if τ'' is schedulable, τ is also schedulable; this holds from the fact that FPS is sustainable with respect to increasing T_i and decreasing C_i [11]. Therefore, the schedulability of τ' under FPS guarantees that of τ under FPS, recorded as follows.

Theorem 1. *For every $\tau_k \in \tau$, we define $\tau'_k \in \tau'$ such that $T'_k = \lfloor \frac{T_k}{\alpha_k} \rfloor$ and $C'_k = \lceil \frac{C_k}{\alpha_k} \rceil$. Then, a task set τ is schedulable by FPS on an m-processor platform, if every $\tau'_k \in \tau'$ has $C'_k \leq \ell \leq T'_k$ that satisfies Eq. (4).*

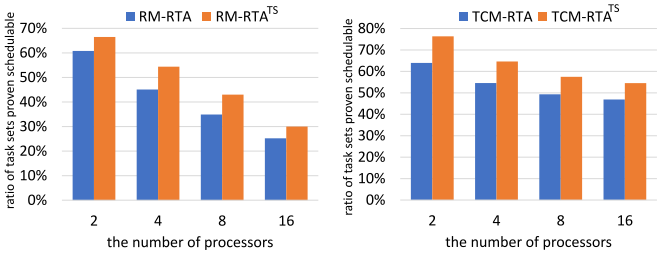
Proof. By Lemma 1, τ' is schedulable by FPS if the "if" statement holds. Also, since T''_i and C''_i are a multiple of T'_i and C'_i, respectively, the schedulability of τ' implies that of τ''. Finally, by the sustainability property [11], the schedulability of τ'' yields that of τ', which proves the theorem. □

Then, the next challenge is how to set α_i for every $\tau_i \in \tau$ to make the task set τ' schedulable by Theorem 1, which is CH2. As explained from Example 1, we can utilize the properties of the task split of τ_i; (P1) increasing α_i helps to reduce the interference of τ_i to other tasks, but (P2) increasing α_i makes it difficult for τ_i itself to be schedulable. Therefore, we need to increase α_i as much as possible by considering P1, but as long as τ_i is schedulable by considering P2, which is achieved by Algorithm 1. Note that the priority of τ'_i (i.e., split task) is inherited from that of τ_i (i.e., original task), meaning that the change of T'_i does not affect the priority of τ'_i; this conserves the properties P1 and P2, and therefore the schedulability improvement by the proposed method solely comes from the task split, not from the priority change. Initially, we set every α_k to 1 in Line 1 (i.e., no task split by default). For each iteration, we check whether each $\tau'_k \in \tau'$ is schedulable or not (Lines 5–11), and return schedulable with $\{\alpha_k\}$ if every τ'_k is schedulable (Lines 12–14); also, if τ'_k is schedulable, we find the largest α_k without compromising the schedulability of τ'_k (Line 16). If there is update of any α_k in the current iteration, we repeat to perform the next iteration; otherwise, we return unschedulable.

Time Complexity. While the time-complexity of RTA for FPS with Eq. (4) is $O(n^2 \cdot \max_{\tau_i \in \tau} T_i)$ [9], that of Algorithm 1 is calculated as follows. Lines 5–14 are equivalent to RTA for FPS, yielding $O(n^2 \cdot \max_{\tau_i \in \tau} T_i)$ time complexity. Also, Line 16 takes $O(n \cdot \log \alpha_{max})$ by applying the binary search; therefore, Lines 15–21 take $O(n^2 \cdot \log \alpha_{max})$, which is no larger than $O(n^2 \cdot \max_{\tau_i \in \tau} T_i)$. Finally, since Lines 5–14 and Lines 15–21 can be repeated up to $n \cdot \alpha_{max}$ times, the time-complexity of Algorithm 1 is $O(n^3 \cdot \max_{\tau_i \in \tau} T_i \cdot \alpha_{max})$. Since we can perform Algorithm 1 offline, the time-complexity is affordable.

Algorithm 1 Assignment of the task split factor $\{\alpha_k\}$

```
1: For every τ'_k ∈ τ', α_k ← 1.
2: UPDATED← True
3: for UPDATED=True do
4:   UPDATED← False
5:   for every τ'_k ∈ τ' do
6:     if Eq. (4) holds for τ'_k then
7:       SCHE_k ← True
8:     else
9:       SCHE_k ← False
10:    end if
11:  end for
12:  if SCHE_k =True holds for every τ'_k ∈ τ' then
13:    Return schedulable with {α_k}
14:  end if
15:  for every τ'_k ∈ τ' | SCHE_k = True do
16:    Find the largest α'_k ∈ [1, α_max] that satisfies Eq. (4) for τ'_k
17:    if α'_k > α_k then
18:      α_k ← α'_k
19:      UPDATED← True
20:    end if
21:  end for
22: end for
23: Return unschedulable
```



(a) RM-RTA versus RM-RTA^{TS} (b) TCM-RTA versus TCM-RTA^{TS}

Fig. 1. The ratio of task sets proven schedulable by X-RTA^{TS} and that by X-RTA for $m = 2, 4, 8$ and 16, where X is RM or TCM.

Run-time Overhead. Once we calculate α_k for every $\tau_k \in \tau$ in Algorithm 1, we schedule each τ_k as if it were τ'_k with $T'_k = \lfloor \frac{T_k}{\alpha_k} \rfloor$ and $C'_k = \lceil \frac{C_k}{\alpha_k} \rceil$. Therefore, it requires little run-time overhead to apply the proposed method.

3. Evaluation

In this section, we evaluate the effectiveness of the task split method in finding additional schedulable task sets.

Task set generation. We have two parameters for task set generation: (i) the number of processors (i.e., $m = 2, 4, 8$ and 16), and (ii) the distribution of task utilization C_i/T_i (i.e., the bimodal distribution with parameters 0.1, 0.3, 0.5, 0.7 and 0.9, and the exponential distribution with the same parameters). For every combination of (i) and (ii), we randomly generate 1000 task sets based on the set generation method in [1,12], yielding 40,000 task sets in total. In each task set, the number of tasks is at least $m + 1$, and the task set utilization $\sum_{\tau_i \in \tau} C_i/T_i$ is at most m. To effectively apply the task split factor $1 \leq \alpha_i \leq 6$, we set T_i and C_i as a multiple of 60.

Evaluation settings. For each task's priority under FPS, we consider two representative priority assignment policies, known to be effective to a multiprocessor platform: (i) the rate monotonic algorithm

Table 1
The number of task sets proven schedulable by X−RTA^{TS}, divided by that by X-RTA, for $m = 8$ under different task utilization distributions, where X is RM or TCM.

Task utilization distribution (avg. # of tasks in each task set)	$\frac{\text{RM−RTA}^{TS}}{\text{RM−RTA}}$	$\frac{\text{TCM−RTA}^{TS}}{\text{/TCM−RTA}}$
Bino. with 0.1 (17.7)	123.2%	121.4%
Bino. with 0.3 (14.6)	115.3%	115.7%
Bino. with 0.5 (12.7)	114.4%	114.1%
Bino. with 0.7 (11.1)	114.5%	114.3%
Bino. with 0.9 (10.2)	114.9%	105.0%
Exp. with 0.1 (44.1)	123.6%	119.2%
Exp. with 0.3 (20.0)	118.4%	120.0%
Exp. with 0.5 (16.4)	113.8%	114.7%
Exp. with 0.7 (14.9)	111.8%	116.7%
Exp. with 0.9 (14.6)	112.9%	116.5%
All distributions (17.6)	117.1%	116.6%

(denoted by RM) [13] that assigns a higher priority to a smaller period T_i, and (ii) the $(T_i - C_i)$ monotonic algorithm (denoted by TCM) that assigns a higher priority to a smaller $(T_i - C_i)$ [14]. For the schedulability analysis, we compare the vanilla RTA (i.e., Lemma 1, denoted by RTA), to the RTA with the task split (i.e., Theorem 1 in which $\{\alpha_i\}$ is assigned by Algorithm 1, denoted by RTA^{TS}). For the generated task sets, we count the number of task sets deemed schedulable by the following pairs of a priority assignment of FPS and a schedulability analysis: RM-RTA, RM-RTA^{TS}, TCM-RTA and TCM-RTA^{TS}.

Performance in covering schedulable task sets. We make the following three important observations. First, the proposed task split method significantly improves the schedulability performance. As shown in Fig. 1, for every pair of the priority assignment policy of FPS and the number of processors, RTA^{TS} finds 16.1%–19.4% additional schedulable task sets, compared to RTA. This indicates the proposed task split method is very effective in covering schedulable task sets, achieving the goal of this paper.

Second, The proposed task split method is effective to both RM and TCM. If we focus on the evaluation results of Fig. 1(a) and those of Fig. 1(b), the schedulability performance improvement of RTA^{TS} over RTA is 16.1%–17.1% and 16.3%–19.4%, respectively for RM and TCM. Hence, the proposed task split method is robust to the priority assignment policy of FPS.

Third, the schedulability performance improvement by the proposed method varies with the task utilization distribution, and there exists a tendency that the method is more effective to task sets in each of which the number of tasks is sufficiently large. As shown in Table 1 for the case of $m = 8$, the exponential distribution with 0.1 and that with 0.3, which result in the largest average number of tasks in each task set (i.e., 44.1 and 20.0, respectively), yield 18.4%–23.6% schedulability improvement. On the other hand, the binomial distribution with 0.9 and that with 0.7, which result in the smallest average number (i.e., 10.2 and 11.1, respectively) yield only 5.0%–14.9% schedulability improvement. This is because, if a task set consists of only a few tasks, the number of tasks which can reduce interference to the target task (that is unschedulable without the task split method) is small, disallowing the target task to be schedulable by the task split. Note that we observe a similar trend to other m.

4. Conclusion

In this paper, we developed a systematic method of how to improve schedulability performance via the task split. We demonstrated that the method finds 16.1%–19.4% (on average) additional schedulable task sets. In the future, we would like to efficiently generalize the method to constrained-deadline tasks in which the relative deadline is the same as or smaller than the period.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (NRF-2021R1A2B5B02001758)

References

[1] M. Bertogna, M. Cirinei, Response-Time Analysis for globally scheduled Symmetric Multiprocessor Platforms, in: Proceedings of IEEE Real-Time Systems Symposium, 2007, pp. 149–160.
[2] M. Bertogna, M. Cirinei, G. Lipari, Schedulability analysis of global scheduling algorithms on multiprocessor platforms, IEEE Trans. Parallel Distrib. Syst. 20 (2009) 553–566.
[3] N. Guan, M. Sitgge, W. Yi, G. Yu, New Response Time Bounds for Fixed Priority Multiprocessor Scheduling, in: Proceedings of IEEE Real-Time Systems Symposium, 2009, pp. 387–397.
[4] J. Lee, New response time analysis for global EDF on a multiprocessor platform, J. Syst. Archit. 65 (2016) 59–63.
[5] S. Chang, J. Sun, Z. Hao, Q. Deng, N. Guan, Computing exact WCRT for typed DAG tasks on heterogeneous multi-core processors, J. Syst. Archit. 124 (2022) 102385: 1–11.
[6] S. Kato, N. Yamasaki, Real-Time Scheduling with Task Splitting on Multiprocessors, in: Proceedings of IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, 2007, pp. 441–450.
[7] A. Burns, R.I. Davis, P. Wang, F. Zhang, Partitioned EDF scheduling for multiprocessors using a c=d task splitting scheme, Real-Time Syst. 48 (2012) 3–33.
[8] D. Casini, A. Biondi, G. Buttazzo, Task splitting and load balancing of dynamic real-time workloads for semi-partitioned EDF, IEEE Trans. Comput. 70 (12) (2021) 2168–2181.
[9] R.I. Davis, A. Burns, Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems, Real-Time Syst. 47 (2011) 1–40.
[10] L. Sha, J.P. Lehoczky, R. Rajkumar, Solutions for Some Practical Problems in Prioritized Preemptive Scheduling, in: Proceedings of IEEE Real-Time Systems Symposium, 1986, pp. 181–191.
[11] S. Baruah, A. Burns, Sustainable Scheduling Analysis, in: Proceedings of IEEE Real-Time Systems Symposium, 2006, pp. 159–168.
[12] J. Lee, A. Easwaran, I. Shin, Maximizing Contention-Free Executions in Multiprocessor Scheduling, in: Proceedings of IEEE Real-Time Technology and Applications Symposium, 2011, pp. 235–244.
[13] C.L. Liu, J. Layland, Scheduling algorithms for multi-programming in a hard-real-time environment, J. ACM 20 (1) (1973) 46–61.
[14] Marko Bertogna, Real-Time Scheduling for Multiprocessor Platforms, (Ph.D. thesis), Scuola Superiore Sant'Anna, Pisa, 2007.