Preempt a Job or Not in EDF Scheduling of Uniprocessor Systems

Jinkyu Lee, *Member*, *IEEE*, and Kang G. Shin, *Life Fellow*, *IEEE*

Abstract—The earliest-deadline-first (EDF) policy has been widely studied for the scheduling of real-time jobs for its effectiveness and simplicity. However, since each preemption incurs an additional delay to the execution of jobs, the effectiveness of EDF is affected greatly by the underlying preemption policy that determines if and when a higher-priority job is allowed to preempt a currently executing lower-priority job. To address this problem, we propose a new and better (in meeting job deadlines) preemption policy of EDF, given a non-zero preemption delay. Specifically, we propose a *controlled preemption* (CP) policy that controls the condition of *preempting* jobs, whereas existing approaches focus on that of *preempted* jobs. We define cp-EDF in which the CP policy is applied to EDF, and analyze its schedulability. This schedulability analysis is then utilized to develop an algorithm that assigns the optimal control parameters of cp-EDF. Our in-depth evaluation has demonstrated that cp-EDF with the optimal parameter assignment improves EDF schedulability over existing preemption policies by up to 7.4%.

Index Terms—Preemption policy, earliest-deadline-first (EDF) policy, real-time uniprocessor scheduling, schedulability analysis, real-time systems

1 Introduction

A real-time scheduling algorithm can be characterized by its prioritization and preemption policies. The prioritization policy determines each job's priority, and numerous prioritization policies have been proposed and deployed, such as EDF (Earliest Deadline First) [18], FP (Fixed Priority) [18], and LLF (Least Laxity First) [17]. The preemption policy determines if and when a higher-priority job can preempt a currently-executing lower-priority job. There are two types of preemption policies: (1) the non-preemptive policy that disallows preemption of an executing job, and (2) the fully-preemptive policy that always allows a higher-priority job to preempt a lower-priority executing job.

Of all prioritization policies, EDF has received significant attention due to the following optimal properties on a uniprocessor platform.

- The non-preemptive EDF (denoted by np-EDF) scheduling algorithm can schedule any task set schedulable by any other scheduling algorithm that employs the non-preemptive policy [11], if the system cannot be left idle if there is an unfinished, ready job, i.e., work-conserving job scheduling.
- The fully-preemptive EDF (denoted by fp-EDF) scheduling algorithm can schedule any task set schedulable by any other scheduling algorithm (that employs any prioritization and preemption policies) [10], and hence, fp-EDF always outperforms np-EDF in meeting job deadlines.

However, the optimality of fp-EDF holds only if each preemption does not incur any delay to the preempting and preempted jobs. In reality, however, when a job preempts another, it incurs a non-trivial preemption delay to the involved jobs, e.g., context switching delay [5], [7], [25], [27]. In the presence of this additional delay that occurs when a job is preempted and then resumes its execution later, fp-EDF is not always optimal, because the system "wastes" processor time whenever a preemption occurs. On the other hand, np-EDF—which is the opposite extreme to fp-EDF—is ineffective in that a higher-priority job may miss its deadline when it is blocked by a lower-priority executing job.

This observation calls for a new preemption policy that regulates preemptions to balance between the system utilization wasted by preemptions and the low-priority jobs' blocking of high-priority jobs. In other words, we would like to develop a new preemption policy for EDF, which can find more schedulable task sets that existing preemption policies cannot feasibly schedule. To meet this goal, we propose *controlled preemption* (denoted by CP), which controls the condition of *preempting jobs* whereas existing approaches [7], [9] regulate that of *preempted jobs*. In Section 2.3 we will elaborate why the control of preempting jobs can be more effective than controlling the preempted job. We then define cp-EDF that adopts EDF and CP as its prioritization and preemption policies, respectively. According to the CP policy, cp-EDF specifies whether or not a job can preempt any other lower-priority job using a simple parameter. By assigning the parameter for each job, cp-EDF can express various EDF scheduling algorithms under different preemption policies, including fp-EDF and np-EDF. Then, we analyze the schedulability of cp-EDF. This analysis also serves as a

• *J. Lee is with Department of Computer Science and Engineering, Sungkyunkwan University, Suwon, Gyeonggi-Do, South Korea. E-mail: jinkyu.lee@skku.edu.*
• *K. G. Shin is with Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI, USA. E-mail: kgshin@eecs.umich.edu.*

Manuscript received 05 July 2012; revised 29 Sep. 2012; accepted 01 Nov. 2012; published online 25 Nov. 2012; date of current version 29 Apr. 2014. Recommended for acceptance by D. Bader.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2012.279

generalization of existing schedulability tests for fp-EDF (assuming no preemption delay) [4] and np-EDF [12].

This schedulability analysis is then used to find an optimal preemption policy of cp-EDF for a given preemption delay, which is associated with assignment of the parameter for each job. For this, we first derive a property of the schedulability analysis and then use it to introduce an optimal assignment algorithm that reduces the search space. Since the algorithm still needs to explore all possible assignments in the worst case, we develop a heuristic algorithm that investigates only some of these assignments. Using in-depth simulations, we demonstrate that cp-EDF with the optimal or heuristic assignment is more effective than fp-EDF, np-EDF and existing approaches [7].

In summary, this paper makes the following contributions.

- Discovery of new preemption policies that control the condition of "preempting" (as opposed to "preempted") jobs for better EDF schedulability for a given preemption delay (in Section 2.3);
- Development of a CP policy for the first time that controls preempting (instead of preempted) jobs, and applies the policy to EDF, resulting in the cp-EDF that is a generalization of fp-EDF and np-EDF (Section 3.1);
- Derivation of a schedulability analysis for cp-EDF for a given preemption delay and a given task set, which also generalizes existing schedulability tests for fp-EDF [4] and np-EDF [12] (Section 3.2);
- Development of an algorithm by using the schedulability analysis for cp-EDF to find the optimal preemption policy associated with the assignment of a parameter (Section 4); and
- Demonstration of the superior scheduling performance of cp-EDF with the optimal preemption policy over the EDF with existing preemption policies (Section 5).

The rest of this paper is organized as follows. Section 2 presents our system model and discusses the related work and existing EDF schedulability tests. Section 3 develops the CP policy as well as cp-EDF and its schedulability analysis. Section 4 derives the optimal preemption policy of cp-EDF. Section 5 evaluates cp-EDF using simulation. Finally, Section 6 concludes the paper.

2 Background

In this section, we first introduce the system model and assumptions, and discuss the related work. Then, we describe how existing studies have incorporated preemption delays into schedulability tests for EDF and why we need to control preempting (not preempted) jobs.

2.1 System Model and Assumptions

We focus on a sporadic task model [20] in which a task $\tau_i \in \mathcal{T}$ is modeled as (T_i, C_i, D_i), where T_i is the minimum separation between two successive invocations, C_i is the worst-case execution time when it is executed *exclusively* (i.e., without preemption), and D_i is its relative deadline. Let n be the total number of tasks. Without loss of generality, tasks can be so arranged that a task with a shorter relative deadline has a smaller task index, i.e., $D_1 \leq D_2 \leq \ldots \leq D_n$. Each τ_i invokes a series of jobs, each separated from its predecessor/successor by at least T_i time units.

We assume that a job can be preempted at any time, but control preemptions for better schedulability. To do this, we introduce a new additional parameter X_i $(= 0$ or $1)$, for each $\tau_i \in \mathcal{T}$. This parameter controls preemptions such that a job invoked by task τ_i with $X_i = 1$ $(X_i = 0)$ can (cannot) preempt any other lower-priority job at any time. Section 3.1 will give a detailed account of this.

In this paper, we focus on a uniprocessor platform, which can serve at most one job at a time. We introduce a new delay parameter α associated with each preemption where α is the worst-case delay in preempting a job. We consider predictable processors which do not have any cache [23], [27], [29], e.g., ARM2. These processors have already been deployed in embedded systems for their low manufacturing cost and power usage. Therefore, α simply represents the time required for a context switch for each preemption as assumed in [27].

We assume a quantum-based time and let the length of a quantum be one time unit, without loss of generality. All task parameters are specified in multiples of the quantum length or time unit.

2.2 Related Work

Numerous studies have been done to explore preemption policies which are more general than the fully-preemptive and non-preemptive policies [3], [6], [8], [24], [26]. A few of them have tried to improve the schedulability of FP [8], [24] since the fully-preemptive policy is not optimal for FP even without preemption delay. On the other hand, several studies have focused on the accommodation of non-preemptive execution parts in each job [3], [6], [26]. In particular, Baruah [3] applied the limited preemption policy to EDF, which specifies the maximum length of non-preemptive execution for each job. However, all these studies assumed that each preemption does not incur any additional delay. They also specified conditions of preempted (not preempting) jobs.

Another category of the related work is the preemption delay analysis (with the analysis incorporated into schedulability tests) [1], [13]–[15], [21], [27]. However, they have focused on the fully-preemptive policy with FP [1], [14], [15], [21], [27] and EDF [13], and have not aimed at improving schedulability by developing more suitable preemption policies.

Recently, researchers have started search for the best preemption policies for given preemption delays [7], [9], with the same goal as this paper. Bertogna et al. [7] adopted the limited preemption policy [3], and determined the number of non-preemptive chunks for each job where each non-preemptive chunk is executed atomically. Then, for better schedulability, they solved the problem of finding the "optimal" number of chunks for each job. This solution was then extended to environments where preemption delays were specified for given preemption points [9]. In these studies, preemption delays were incorporated into preempted (not preempting) jobs, and this may result in a pessimistic schedulability test due to over-estimation of the number of preemptions. In the next subsection, we will detail this by presenting how existing studies incorporate preemption delays into schedulability tests.

2.3 Incorporating Preemption Delays into EDF Schedulability Tests

Baruah et al. [4] defined the demand bound function of τ_i for an interval of length l (denoted by $\mathsf{DBF}(\tau_i, l)$) that computes the cumulative maximum execution requirements of all jobs invoked by τ_i, each of whose release time and deadline are within the interval as

$$\mathsf{DBF}(\tau_i, l) \triangleq \max\left(0, \left\lfloor \frac{l - D_i}{T_i} \right\rfloor + 1\right) \cdot C_i. \qquad (1)$$

Using the demand bound function, a necessary and sufficient schedulability analysis for fp-EDF has been developed in [4] as follows. Task set $\mathcal{T}$ is schedulable by fp-EDF on a uniprocessor platform if and only if the following is true for all $l > 0$ [4]:

$$\sum_{\tau_i \in \mathcal{T}} \mathsf{DBF}(\tau_i, l) \leq l. \qquad (2)$$

The above analysis is valid only when each preemption does not incur any additional delay. If we consider the delay α for each preemption, extra delays should be added to the LHS of Eq. (2). Since a job can preempt any other job at most once under fp-EDF [19], we incorporate preemption delays into *preempting* jobs, and then calculate $\mathsf{DBF_P}(\tau_i, l)$ for the cumulative preemption delays by *preempting* jobs invoked by τ_i, each of whose release time and deadline are within an interval of length l as

$$\mathsf{DBF_P}(\tau_i, l) \triangleq \max\left(0, \left\lfloor \frac{l - D_i}{T_i} \right\rfloor + 1\right) \cdot \alpha. \qquad (3)$$

Then, a task set is schedulable by fp-EDF on a uniprocessor platform in the presence of the preemption delay α if the following inequality holds for all $l > 0$:

$$\sum_{\tau_i \in \mathcal{T}} (\mathsf{DBF}(\tau_i, l) + \mathsf{DBF_P}(\tau_i, l)) \leq l. \qquad (4)$$

On the other hand, Bertogna et al. [7] incorporated preemption delays into *preempted* jobs. Since known upper-bounds on the number of times for a job to be preempted under fp-EDF are pessimistic, they adopted the limited preemption policy [3] to EDF (denoted by lp-EDF). Since lp-EDF specifies the number of non-preemptive chunks of τ_i (denoted by p_i), the number of times for a job of τ_i to be preempted is upper-bounded by $p_i - 1$. In this case, the total execution time of each job of τ_i including preemption delays in case the job is *preempted*, is upper-bounded by $C_i + (p_i - 1) \cdot \alpha$, and then the length of each non-preemptive chunk with its preemption delay is set to $\frac{C_i + (p_i - 1)\cdot\alpha}{p_i}$. Then, under lp-EDF, we compute $\mathsf{DBF_{lp}}(\tau_i, l)$ for the cumulative preemption delays by *preempted* jobs invoked by τ_i, each of whose release time and deadline are within an interval of length l as:

$$\mathsf{DBF}_{lp}(\tau_i, l) \triangleq \max\left(0, \left\lfloor \frac{l - D_i}{T_i} \right\rfloor + 1\right) \cdot (p_i - 1) \cdot \alpha. \qquad (5)$$

Then, task set $\mathcal{T}$ is schedulable by lp-EDF on a uniprocessor platform in the presence of the preemption delay α if the following inequality holds for all $l > 0$ [7]:

$$\mathsf{Blocking} + \sum_{\tau_i \in \mathcal{T}} \big(\mathsf{DBF}(\tau_i, l) + \mathsf{DBF_{lp}}(\tau_i, l)\big) \leq l, \qquad (6)$$

where $\mathsf{Blocking} \geq 0$ is the maximum blocking time of higher-priority jobs by the execution of non-preemptive chunks of lower-priority jobs, and it gets larger as p_i gets smaller (or as $\frac{C_i + (p_i - 1)\cdot\alpha}{p_i}$ gets larger). Details can be found in [7].

The authors of [7] solved the problem of finding $\{p_i\}$ that satisfies Eq. (6) for all $l > 0$. However, this solution could be ineffective due to over-estimation of the number of preemptions in Eq. (5), as we discuss next.

Observe that the total demand for fp-EDF in the LHS of Eq. (4) is equal to that for lp-EDF in the LHS of Eq. (6) when $p_i = 2$ for all tasks and $\mathsf{Blocking} = 0$. This means, in order for lp-EDF to have better schedulability than that of fp-EDF, that p_i should be less than 2.0 on average. However, in many cases it is not possible for p_i to be small. This is because a smaller p_i results in a larger $\mathsf{Blocking}$, and a higher-priority job may then miss its deadline due to the execution of a larger non-preemptive chunk of a lower-priority job. We demonstrate this with the following example.

Example 1. Suppose that there are two tasks, $\mathcal{T} = \{\tau_i = (10, 3, 5), \tau_2 = (10, 5, 10)\}$, and the preemption delay is $\alpha = 1$. Then, if $p_2 \leq 3$, the length of each non-preemptive chunk of τ_2 $\left(\frac{C_2 + (p_2 - 1)\cdot\alpha}{p_2}\right)$ is strictly larger than 2, and then the execution of a non-preemptive chunk of τ_2 causes a job of τ_1 to miss its deadline since τ_1's worst-case execution time (3) plus blocking time (>2) is strictly larger than its relative deadline (5). Therefore, the smallest possible value of p_2 is 4, and then lp-EDF with $p_2 = 4$ cannot satisfy Eq. (6) for $l = 10$ (i.e., $\mathsf{DBF}(\tau_1, 10) + \mathsf{DBF}(\tau_2, 10) + \mathsf{DBF_{lp}}(\tau_2, 10) = 3 + 5 + 3 > 10$). This means that when $\alpha = 1$, task set $\mathcal{T}$ is unschedulable by lp-EDF with any $\{p_i\}$ [by Eq. (6)] while it is schedulable by fp-EDF [by Eq. (4)]. This is because Eq. (5) over-estimates the number of times for a job of τ_2 to be preempted; this job seems to be preempted $p_2 - 1 = 3$ times, but it can actually be preempted only once.

The above example shows the limitation of a preemption policy, which incorporates preemption delays into *preempted* jobs. Therefore, instead of controlling the condition of *preempted* jobs, we propose a new preemption policy that controls that of *preempting* jobs. This policy can then be closely associated with the schedulability analysis with a structure similar to Eq. (4) in that we can totally remove the term $\mathsf{DBF_P}(\tau_i, l)$ when we decide to disallow jobs of τ_i to preempt any other job. In Section 3, we will detail this policy and analyze its schedulability.

3 CP-EDF Scheduling Algorithm and Its Schedulability Analysis

Our goal is to find a better preemption policy of EDF for a given preemption delay. As shown in Section 2.3, we need a new preemption policy that controls preemptions at the side of preempting jobs. We first present a preemption policy that meets this need, and then describe cp-EDF in which this policy is applied to EDF. Second, we develop the schedulability test of cp-EDF for a given preemption delay and a given task set. The analysis will be used in Section 4 to find the best preemption policy associated with a per-task control parameter for cp-EDF.

3.1 The CP Policy and cp-EDF Scheduling Algorithm

We consider a preemption policy under which preemptions are controlled by a parameter X_i, i.e., whether or not a job of τ_i is allowed to preempt any other lower-priority job. That is, if $X_i = 1$ ($X_i = 0$), a job of τ_i can (cannot) preempt any other lower-priority job. We call this the *controlled preemption* (CP) policy. The CP policy is concise but effective in that it controls the number of preemptions allowed, and this control is closely associated with its schedulability analysis.

Let cp-EDF denote a scheduling algorithm that adopts controlled preemption (CP) and Earliest-Deadline-First (EDF) as its preemption and prioritization policies, respectively. Algorithm 1 shows a formal description of the cp-EDF scheduling algorithm. Note that as you can see in Step 9 of job release, J_{new} does not start its execution immediately, although its priority is higher than that of J_{curr} and it is allowed to preempt other lower-priority jobs (i.e., $d_{new} < d_{curr}$ and $X_i = 1$). Instead, it checks whether there is a job with an earlier deadline than that of J_{new} in the wait queue. This prevents a job with $X_i = 0$ from blocking by more than one lower-priority job while the job itself does not trigger any preemption (where the preemption is activated by J_{new} with $X_i = 1$, not by a higher-priority job with $X_i = 0$ in the wait queue).

Algorithm 1 cp-EDF scheduling algorithm

Job release: when a job J_{new} of task τ_i is released,

```
1: Set the absolute deadline of $J_{new}$: $d_{new} \leftarrow t + D_i$.
2: Check whether there is a currently executing job $J_{curr}$
   (which has its absolute deadline of $d_{curr}$).
3: if $J_{curr}$ does not exist then
4:     Execute $J_{new}$.
5: else
6:     if $d_{new} \geq d_{curr}$ or $X_i = 0$ then
7:         Put $J_{new}$ into the wait queue.
8:     else
9:         Stop executing $J_{curr}$ and put $J_{curr}$ and $J_{new}$ into the
           wait queue, and perform the job completion
           process.
10:    end if
11: end if
```

Trivially, cp-EDF can represent both np-EDF and fp-EDF, as stated in the following lemma.

Job completion: when the currently executing job J_{curr} finishes its execution,

1: Start to execute a job with the earliest deadline in the wait queue.

Lemma 1. *The cp-EDF scheduling algorithm subsumes both np-EDF and fp-EDF.*

Proof. The proof is trivial as follows.

- np-EDF is equivalent to cp-EDF with $X_i = 0 \quad \forall \tau_i \in \mathcal{T}$; and
- fp-EDF is equivalent to cp-EDF with $X_i = 1 \quad \forall \tau_i \in \mathcal{T}$. □

3.2 Schedulability Analysis of cp-EDF

To determine if a given task set is schedulable with a scheduling algorithm, most schedulability tests find necessary conditions for a job to miss its deadline. Then, a task is deemed schedulable if the task set avoids such conditions. We also follow this to develop a schedulability test of cp-EDF for a given preemption delay and a given task set.

Let us consider what happens if there is a deadline miss under cp-EDF. Let t_2 be the first time instant at which a job misses its deadline, and t_1 ($<t_2$) be the latest time instant at which any job whose deadline is no later than t_2, does not have any remaining execution. By definition of t_1 and t_2, $[t_1, t_2)$ is a busy interval and there should be at most two types of jobs occupying the processor during $[t_1, t_2)$: a set, $\mathcal{J}_1$, of jobs whose release times and deadlines are within $[t_1, t_2)$; and another set, $\mathcal{J}_2$, of jobs which do not belong to $\mathcal{J}_1$. Then, each job in $\mathcal{J}_2$ should meet the following conditions.

1) The release time is before t_1 and the deadline is after t_2 (otherwise, the job cannot occupy the processor in $[t_1, t_2)$ or belongs to $\mathcal{J}_1$);
2) By condition 1, the priority of the job is lower than that of any job in $\mathcal{J}_1$, and therefore, the job can occupy the processor only in $[t_1, t_1 + b]$ where $t_1 + b$ is a time instant at which the job is preempted or finishes its execution; and
3) By condition 2, there exists only one job in $\mathcal{J}_2$. Then, we calculate the amount of system occupancy by jobs in $\mathcal{J}_1$ and $\mathcal{J}_2$, and this amount is necessarily larger than $t_2 - t_1 (\triangleq l)$ for a job to miss its deadline. Therefore, if this amount is not larger than l, $\forall l > 0$, then the task set avoids the necessary condition for missing deadlines, i.e., the task set is schedulable.

We first calculate the amount of system occupancy by the job in $\mathcal{J}_2$. By condition 1, the job should be invoked by τ_i whose relative deadline is strictly larger than l (i.e., $D_i > l$). Then, we can calculate the longest duration for the job in $\mathcal{J}_2$ occupying the system in $[t_1, t_2)$ (an interval of length $l \triangleq t_2 - t_1$) by choosing the maximum execution times among all tasks that satisfy $D_i > l$ as

$$\mathsf{B} \triangleq \begin{cases} \min(l, \max_{\{\tau_i \in \mathcal{T} | D_i > l\}} C_i), & \text{if } D_1 \leq l < D_n, \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

Note that the duration is upper-bounded by the interval length l. The duration is zero unless $D_1 \leq l < D_n$: when $l < D_1$, there is no job in $\mathcal{J}_1$ (i.e., no deadline miss at t_2), and when $l > D_n$, there is no job in $\mathcal{J}_2$.

We now calculate how long jobs in $\mathcal{J}_1$ occupy the system. Suppose that the amount of system occupancy by the job in $\mathcal{J}_2$ in $[t_1, t_2)$ is b $(0 \leq b \leq \mathsf{B})$. Since each job invoked by τ_i in $\mathcal{J}_1$ has higher priority than the job in $\mathcal{J}_2$ by condition 2, any job invoked by τ_i with $X_i = 1$ in $\mathcal{J}_1$ cannot be released in $[t_1, t_1 + b)$. Otherwise, the job in $\mathcal{J}_2$ is preempted before $t_1 + b$. Considering each job can cause at most one preemption, the amount of system occupancy by jobs invoked by τ_i with $X_i = 1$ in $[t_1, t_2)$ is upper-bounded by $\mathsf{DBF}(\tau_i, l - b) + \mathsf{DBF}_{\mathsf{P}}(\tau_i, l - b)$, where $l \triangleq t_2 - t_1$, and $\mathsf{DBF}(\cdot)$ and $\mathsf{DBF}_{\mathsf{P}}(\cdot)$ are defined in Eqs. (1) and (3). On the other hand, if $X_i = 0$, a job invoked by τ_i cannot preempt any other job, and it can be released any time before t_1 (by definition of $\mathcal{J}_1$) without triggering any preemption. Therefore, the amount of system occupancy by jobs of τ_i with $X_i = 0$ in $[t_1, t_2)$ is upper-bounded by $\mathsf{DBF}(\tau_i, l)$. Therefore, if the amount of system occupancy by the job in $\mathcal{J}_2$ is b $(0 \leq b \leq \mathsf{B})$, the total amount of system occupancy by jobs in $\mathcal{J}_1$ is upper-bounded by

$$\sum_{\tau_i\in\mathcal{T}|X_i=1} \mathsf{DBF}(\tau_i, l-b) + \sum_{\tau_i\in\mathcal{T}|X_i=1} \mathsf{DBF_P}(\tau_i, l-b) + \sum_{\tau_i\in\mathcal{T}|X_i=0} \mathsf{DBF}(\tau_i, l). \quad (8)$$

Finally, we complete the calculation of the amount of system occupancy by jobs in $\mathcal{J}_1$ and $\mathcal{J}_2$ in $[t_1, t_2)$. Using the information, the following theorem formally describes a schedulability test of cp-EDF for a given preemption delay and a given task set.

Theorem 1. *A task set $\mathcal{T}$ is schedulable by cp-EDF on a uniprocessor platform in the presence of the preemption delay α if the following inequality holds for all $l > 0$:*

$$\max_{0\le b\le B}\left(b + \sum_{\tau_i\in\mathcal{T}|X_i=1} \mathsf{DBF}(\tau_i, l-b) + \sum_{\tau_i\in\mathcal{T}|X_i=1} \mathsf{DBF_P}(\tau_i, l-b)\right) + \sum_{\tau_i\in\mathcal{T}|X_i=0} \mathsf{DBF}(\tau_i, l) \le l, \quad (9)$$

where B, $\mathsf{DBF}(\cdot)$ and $\mathsf{DBF_P}(\cdot)$ are defined as in Eqs. (7), (1) and (3), respectively.

Proof. We prove the contraposition: if $\mathcal{T}$ is not schedulable by cp-EDF, Eq. (9) does not hold. We follow the definitions of $t_1, t_2, \mathcal{J}_1$ and $\mathcal{J}_2$ in the second paragraph of Section 3.2.

For a job to miss its deadline, the amount of occupancy by jobs in $\mathcal{J}_1$ and $\mathcal{J}_2$ should be larger than l ($\triangleq t_2 - t_1$). Since the amount is upper-bounded by the LHS of Eq. (9), the LHS is necessarily larger than l. Thus, the contraposition is true. □

Compared to the classic schedulability analysis of fp-EDF in Eq. (4), Theorem 1 requires investigating all $0 \le b \le B$ for given l. Due to the investigation, one may wonder whether it is tractable to perform a schedulability analysis in Theorem 1. The following lemma states that the time-complexity is pseudo-polynomial, which is regarded tractable in real-time schedulability analyses [4].

Lemma 2. *Theorem 1 for a task set with given $\{X_i\}$ can be tested in pseudo-polynomial time in the task parameters, if $\sum_{\tau_i\in\mathcal{T}|X_i=0} C_i/T_i + \sum_{\tau_i\in\mathcal{T}|X_i=1}(C_i+\alpha)/T_i$ is upper-bounded by a constant that is strictly smaller than 1.*

Proof. The proof is given in Appendix, which can be found in the Computer Society Digital Library at https://doi.ieeecomputersociety.org/10.1109/TC.2012.279. □

The above schedulability analysis for cp-EDF in the presence of a given preemption delay generalizes the existing schedulability analyses for both fp-EDF and np-EDF. This is formalized in the following lemma.

Lemma 3. *The schedulability analysis for cp-EDF in Theorem 1 is a generalization of the schedulability analyses for both fp-EDF [4] (assuming no preemption delay) and np-EDF [12].*

Proof. The proof is trivial as follows.

- The schedulability analysis for fp-EDF [4] under the assumption of no preemption delay is equivalent to that for cp-EDF in Theorem 1 with $X_i = 1, \forall\tau_i \in \mathcal{T}$ and $\alpha = 0$; and
- The schedulability analysis for np-EDF [12] is equivalent to that for cp-EDF in Theorem 1 with $X_i = 0\ \forall\tau_i \in \mathcal{T}$.

Note that under np-EDF (or cp-EDF with $X_i = 0$ $\forall\tau_i \in \mathcal{T}$), the preemption delay α does not affect the schedulability of each task set since no preemption is allowed under np-EDF. □

4 Optimal Preemption for cp-EDF

We now present how to find an "optimal" preemption policy for cp-EDF for given preemption delay and task set, where an optimal preemption policy for task set $\mathcal{T}$ represents an instance of $\{X_i\}$ that satisfies Eq. (9) $\forall l > 0$.

A naive approach to finding an optimal preemption policy for cp-EDF is to test Eq. (9) for the entire search space $\Gamma \triangleq \{[X_1,\ldots,X_n] | X_i = 0 \text{ or } 1, \forall\tau_i \in \mathcal{T}\}$, in which there are 2^n candidates. To reduce the search space, we first explore the property of Eq. (9).

Lemma 4. *X_k does not affect the LHS of Eq. (9) for $l < D_k$.*

Proof. If $l < D_k$, $\max\left(0, \left\lfloor\frac{l-b-D_i}{T_i}\right\rfloor\right) = 0$ for any $b \ge 0$. Therefore, τ_k does not contribute anything to the LHS of Eq. (9) via $\mathsf{DBF}(\cdot)$ or $\mathsf{DBF_P}(\cdot)$ terms. Thus, the LHS is independent of X_k. □

By the above lemma, the examination of $D_1 \le l < D_2$ shows that the LHS of Eq. (9) only depends on X_1. So, we investigate whether Eq. (9) holds for $D_1 \le l < D_2$ when $X_1 = 0$ and $X_1 = 1$, respectively. If the equation does not hold for given X_1, we can rule out all the combinations that include the given X_1 in Γ. Generally, the LHS of Eq. (9) for $D_k \le l < D_{k+1}$ only depends on $X_1,\ldots,X_k$. Algorithm 2 shows an algorithm that finds an optimal preemption policy associated with $\{X_i\}$, and the following lemma proves its correctness.

Algorithm 2 Optimal algorithm ($\mathcal{T}$)

```
 1: The search space is $\Gamma \triangleq \{[X_1,\ldots,X_n] | X_i = 0 \text{ or } 1, \forall\tau_i \in \mathcal{T}\}$.
 2: for $k = 1$; $k \le n-1$; $k$++ do
 3:     Find all combinations of $\{[X_1,\ldots,X_k]\}$ that do not
        satisfy Eq. (9) for any $D_k \le l < D_{k+1}$, and remove
        any instance that contains these combinations in the
        search space $\Gamma$.
 4:     if there is no remaining instance in $\Gamma$ then
 5:         return INFEASIBLE.
 6:     end if
 7: end for
 8: if there exists at least one instance in $\Gamma$ that satisfies Eq. (9)
    $\forall l \ge D_n$ then
 9:     return FEASIBLE with one of the instances.
10: else
11:     return INFEASIBLE.
12: end if
```

Lemma 5 (Correctness of the Optimal Algorithm). *The optimality of Algorithm 2 is correct.*

Proof. Since Steps 2-7 and 8 in Algorithm 2 check whether the resulting instance $\{X_i\}$ satisfies Eq. (9) $\forall l < D_n$ and $\forall l \geq D_n$, respectively, the algorithm is correct if the task set is deemed feasible.

Now, we show that there exists no $\{X_i\}$ which satisfies Eq. (9) $\forall l > 0$ if the algorithm finds the task set infeasible.

Suppose that there is an instance $\{X_i\}$ that satisfies Eq. (9) $\forall l > 0$ while the algorithm finds $\mathcal{T}$ infeasible. We have to consider two cases: the feasible instance is excluded by Step 3 and Step 8. By Lemma 4, a feasible instance cannot be excluded by Step 3, and it is trivial that a feasible instance cannot be excluded by Step 8. This is a contradiction. □

Even though Algorithm 2 can significantly reduce the search space on average, it may still require consideration of all combinations (i.e., 2^n) in the worst case. In other words, the total time-complexity for Algorithm 2 is $O(P \cdot 2^n)$, which is exponential, where P denotes pseudo-polynomial time-complexity of testing Theorem 1 for a given task set with given $\{X_i\}$ (see Eq. (11) in the Appendix).

Algorithm 3 Heuristic algorithm ($\mathcal{T}$)

```
1: $X_i \leftarrow 0 \ \forall \tau_i \in \mathcal{T}$.
2: for $k = 1$; $k \leq n - 1$; $k++$ do
3:     for $j = k$; $j \geq 1$; $j--$ do
4:         if $X_j = 0$ and Eq. (9) does not hold for any $D_k \leq l < D_{k+1}$ then
5:             $X_j \leftarrow 1$.
6:         else
7:             $j \leftarrow 0$, i.e., exit the inner-loop of Step 3.
8:         end if
9:     end for
10: end for
11: if Eq. (9) is satisfied $\forall l > 0$ then
12:     return FEASIBLE.
13: else
14:     return INFEASIBLE.
15: end if
```

To reduce the number of $\{X_i\}$ to be considered in the worst case, we may use a heuristic as shown in Algorithm 3. Initially, X_i is set to 0 for all $\tau_i \in \mathcal{T}$. Whenever Eq. (9) for $D_k \leq l < D_{k+1}$ is not satisfied, the algorithm chooses one of the tasks such that $X_i = 0$ and $i \leq k$ (tasks that can affect the LHS of Eq. (9) by Lemma 4), and set $X_i = 1$ for the task. Once X_i is set to 1, it does not change any more. Therefore, the heuristic algorithm explores at most $n + 1$ combinations. Then, the total time-complexity of Algorithm 3 is pseudo-polynomial in the task parameters (i.e., $O(P \cdot n)$). Since schedulability analysis can be performed offline, a pseudo-polynomial time-complexity is regarded tractable in real-time schedulability analyses [4].

To demonstrate the effectiveness of the heuristic algorithm, we will compare the performance of the heuristic and optimal algorithms in Section 5. For better understanding, we now give an example how Algorithm 3 works.

Example 2. We consider a task set with three tasks $\mathcal{T} = \{\tau_1 = (7, 1, 2), \tau_2 = (6, 1, 4), \tau_3 = (7, 2, 6)\}$, with the preemption delay of $\alpha = 1$.

By Step 1 in Algorithm 3, $X_i, \forall \tau_i \in \mathcal{T}$ is initially set to 0. Then, when $k = 1$ in Step 2, the inner-loop is examined only for $j = 1$. The if-phrase in Step 4 is true since $X_1 = 0$ and Eq. (9) does not hold for $l = 2$ (i.e., $3 > 2$). Therefore, X_1 is set to 1 in Step 5. Next, when $k = 2$ in Step 2, the inner-loop is checked for $j = 2$ and then $j = 1$. When $j = 2$, the if-phrase in Step 4 holds, i.e., $X_2 = 0$ and the LHS of Eq. (9) is equal to 5 for $l = 4$. Therefore, X_2 is also set to 1 in Step 5. For $j = 1$, the if-phrase in Step 4 does not hold. After escaping the outer-loop, Step 11 is performed; Eq. (9) is satisfied for all $l > 0$ when $X_1 = 1, X_2 = 1$ and $X_3 = 0$, and therefore the final $\{X_i\}$ is feasible.

5 Evaluation

In this section, we evaluate the scheduling performance of cp-EDF with $\{X_i\}$ assigned by the optimal and heuristic algorithms. We first describe how task sets are generated, and then compare cp-EDF with EDF under different preemption policies.

5.1 Generation of Task Sets

We generate task sets based on the technique proposed in [2], which has been widely used before [16], [22]. There are three input parameters: (a) the task system (constrained or implicit deadlines), (b) individual task utilization (C_i/T_i) distributions (bimodal with parameter[1]: 0.1, 0.3, 0.5, 0.7, or 0.9, or exponential with parameter[2]: 0.1, 0.3, 0.5, 0.7, or 0.9), and (c) individual task period (T_i) distribution (uniform distribution in $[1, 1000)$ or trimodal distribution in which T_i is uniformly selected in $[1, 10)$, $[10, 100)$, and $[100, 1000)$ with equal probability $\frac{1}{3}$). For each task, T_i is chosen based on the given period distribution, C_i is chosen based on the given bimodal or exponential parameter for C_i/T_i, and D_i is uniformly distributed in $[C_i, T_i]$ for constrained deadline task systems or D_i is equal to T_i for implicit deadline task systems.

For each combination of (a), (b) and (c), we repeat the following procedure and generate 10,000 task sets.

1) Initially, we generate a set of 2 tasks.
2) In order to exclude unschedulable sets, we check whether the generated task set can satisfy the necessary and sufficient feasibility condition [4]. This condition means the schedulability of a task set under fp-EDF in the absence of preemption delay.
3) If it fails to pass the feasibility test, we discard the generated task set and return to Step 1. Otherwise, we include this set for evaluation. Then, this task set serves as a basis for the next new set; we create a new set by

1. For a given bimodal parameter p, a value for C_i/T_i is uniformly chosen in $[0, 0.5)$ with probability p, and in $[0.5, 1]$ with probability $1 - p$.

2. For a given exponential parameter $1/\lambda$, a value for C_i/T_i is chosen according to an exponential distribution whose probability density function is $\lambda \cdot \exp(-\lambda \cdot x)$.

adding a new task into an already created and tested set, and return to Step 2.

For any given task system, 10,000 task sets are created for each task utilization and period model, thus resulting in 100,000 task sets in total.

5.2 Scheduling Performance

The scheduling performance of cp-EDF is evaluated in comparison with EDF under other preemption policies. For this, we test 100,000 constrained deadline task sets and another 100,000 implicit task sets over varying preemption delays ($\alpha = 0, 1, 2, 4, 8, 16, 32, 64, 128, 256, 512$ and 1024) and the two different period distributions using the following schedulability tests: (i) cp-EDF with the optimal $\{X_i\}$ assignment (Algorithm 2), (ii) cp-EDF with the heuristic $\{X_i\}$ assignment (Algorithm 3), (iii) lp-EDF with its optimal $\{p_i\}$ assignment in [7], (iv) fp-EDF schedulability analysis [Eq. (4)], and (v) np-EDF schedulability analysis in [12]. These are respectively annotated as cp-EDF-o, cp-EDF-h, lp-EDF-o, fp-DEF, and np-EDF. For simplicity, a schedulability test or a scheduling algorithm A is said to *dominate* another test or algorithm B if any task set schedulable by B is schedulable by A. To represent average performance, we also say that a schedulability test or scheduling algorithm A is *better* than B if the number of schedulable task sets by A is larger than that by B.

One may think that some values for α are unrealistic since the preemption delay is too high compared to the task execution time. However, our setting from $\alpha = 0$ to $\alpha = 1024$ generalizes the degree of preemption allowance of each task. While $\alpha = 0$ means a job can be preempted anytime without any overhead, $\alpha = 1024$ represents the other extreme case—each task cannot be preempted due to its transactional operations. Through the case of $\alpha = 1024$, we want to verify that cp-EDF-o as well as cp-EDF-h correctly chooses $X_i = 0$ for all $\tau_i \in \mathcal{T}$, i.e., np-EDF, in case that the non-preemptive policy is the best.

Figs. 1(a)–1(d) show the number of constrained and implicit deadline task sets deemed schedulable by (ii), (iii), (iv) and (v) while varying preemption delays. In the figures, we commonly observe the behavior of fp-EDF as follows: it deems all task sets schedulable when $\alpha = 0$, reflecting the optimality of fp-EDF in the absence of preemption delay; the number of schedulable task sets gets smaller as α becomes larger; and it deems no task sets schedulable when preemptions are disallowed, i.e., when $\alpha = 1024$. On the other hand, the number of task sets are deemed schedulable by np-EDF is independent of α. Then, when the preemption delay is small ($\alpha \leq 32$), fp-EDF is better than np-EDF. However, a large preemption delay ($\alpha \geq 64$) entails the opposite behavior. This implies an intuitive remark: if we have only two extreme preemption policies, we should apply fp-EDF (np-EDF) in some environments where the preemption delay is small (large).

If we focus on cp-EDF-o and cp-EDF-h, we cannot distinguish between them, and therefore, we do not draw the line for cp-EDF-o in all the figures. This means, the heuristic in Algorithm 3 is so effective that it is comparable to the optimal one, Algorithm 2. More specifically, the difference between schedulable task sets by cp-EDF-o and cp-EDF-h is less than 0.01% in any case, and the following example shows a task set for such a rare case.

Example 3. Consider a task set with three tasks $\mathcal{T} = \{\tau_1 = (10, 1, 3), \tau_2 = (3, 1, 3), \tau_3 = (5, 2, 5)\}$, and the preemption delay is $\alpha = 1$. While cp-EDF-o finds the task set schedulable with $X_1 = 1$, $X_2 = 0$ and $X_3 = 0$, cp-EDF-h deems the task set unschedulable because it does not investigate the feasible $\{X_i\}$ to reduce time-complexity.

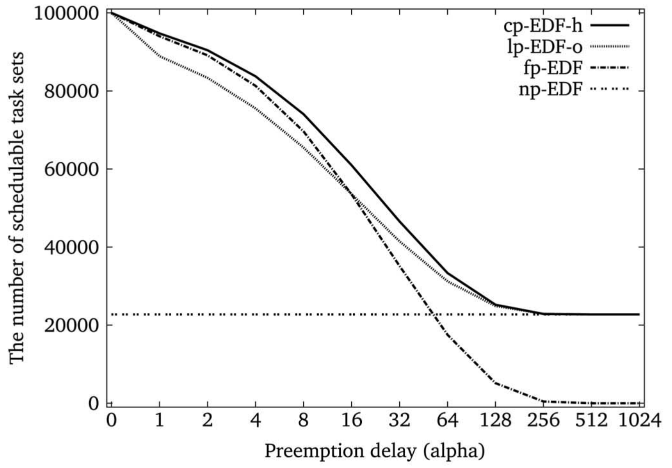
Now, we compare cp-EDF-h with lp-EDF-o. One of the important observations is that they are incomparable; while cp-EDF-h is better for constrained deadline task sets in Figs. 1(a) and 1(c), lp-EDF-o is better for implicit deadline task sets in Figs. 1(b) and 1(d). This is because, while scheduling a new task set, in which the relative deadline of each task is reduced, is in general more difficult than scheduling the original task set, such a difficulty is more pronounced for lp-EDF-o. Under lp-EDF-o, a smaller relative deadline causes a shorter length of each non-preemptive chunk as shown in Example 1, which entails a larger number of non-preemptive chunks and then a more pessimistic upper-bound on the number of preemptions. Therefore, the performance of lp-EDF-o is degraded more for constrained deadline task sets.

This way, cp-EDF-h (lp-EDF-o) is favorable for constrained (implicit) deadline tasks, as the following numeric values demonstrate. In Fig. 1(a), cp-EDF-h finds up to 11.4% more schedulable constrained deadline task sets which are schedulable by neither fp-EDF nor np-EDF. In addition, cp-EDF-h deems at most 7.4% more constrained deadline task sets schedulable, which are not schedulable by any other schedulability tests. We also observe a similar trend in Fig. 1(c). Likewise, in Figs. 1(b) and 1(d), lp-EDF-o covers up to 6.7% more schedulable implicit deadline task sets, which are not schedulable by fp-EDF, np-EDF and cp-EDF-h.

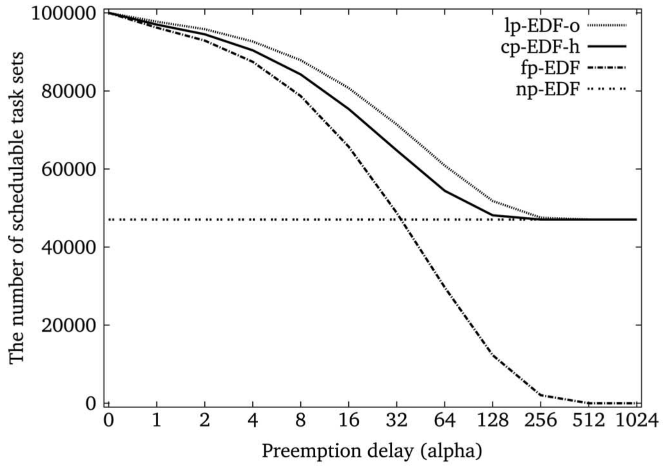
To see how the number of tasks affects the improvement by cp-EDF-h, we plot schedulable task set ratio according to the different number of tasks in Figs. 2(a) and 2(b). The two figures correspond to Fig. 1(a) for $\alpha = 1$ and $\alpha = 4$, respectively. As shown in the figures, we cannot say that there is a clear relationship between the number of tasks and the amount improvement by cp-EDF-h; instead, cp-EDF-h successfully finds additional schedulable task sets that are not covered by lp-EDF-o, regardless of the number of tasks in each task set. We do not plot figures for other settings, but this trend for constrained deadline task sets is observed for other α values and the trimodal period distribution.

Another important observation is that cp-EDF-h is always better than both fp-EDF and np-EDF in Figs. 1(a)–1(d). In fact, we can show that cp-EDF-h dominates them. This is because cp-EDF and its schedulability test are generalizations of np-EDF and fp-EDF, and their schedulability tests, respectively, as we stated in Lemmas 1 and 3, and cp-EDF-o finds an optimal $\{X_i\}$ assignment based on the cp-EDF schedulability analysis. Unlike cp-EDF-o, we observe that lp-EDF-o can be inferior to fp-EDF for some α in Fig. 1(a). This is because the number of preemptions can be over-estimated in lp-EDF-o as mentioned in Section 2.3. In fact, lp-EDF-o dominates np-EDF, but does not dominate fp-EDF due to such over-estimation.

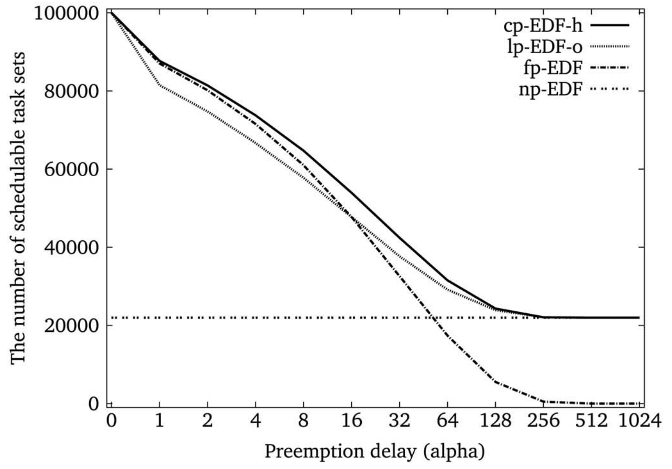
In summary, from the performance evaluation thus far, we make the following observations. First, cp-EDF with the optimal $\{X_i\}$ assignment can always be an alternative to both



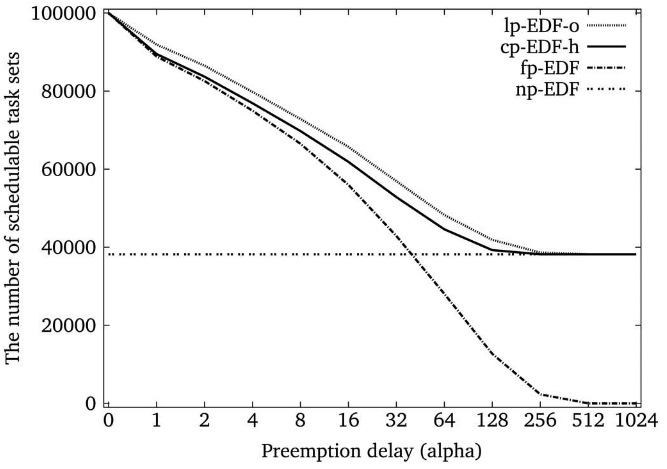
(a) Constrained deadline task sets with uniform period distribution



(b) Implicit deadline task sets with uniform period distribution

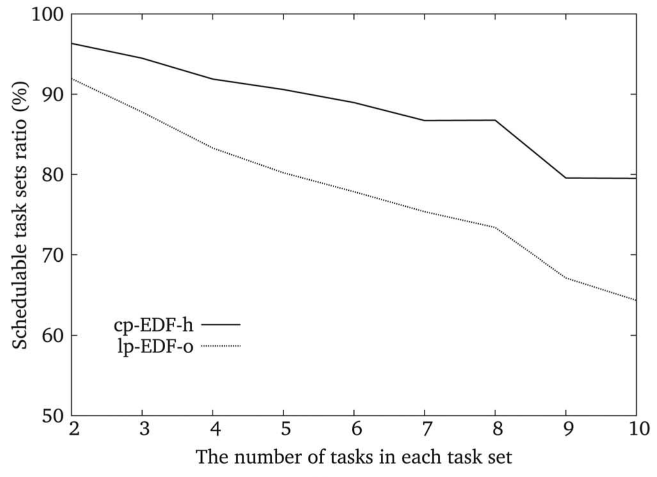


(c) Constrained deadline task sets with trimodal period distribution

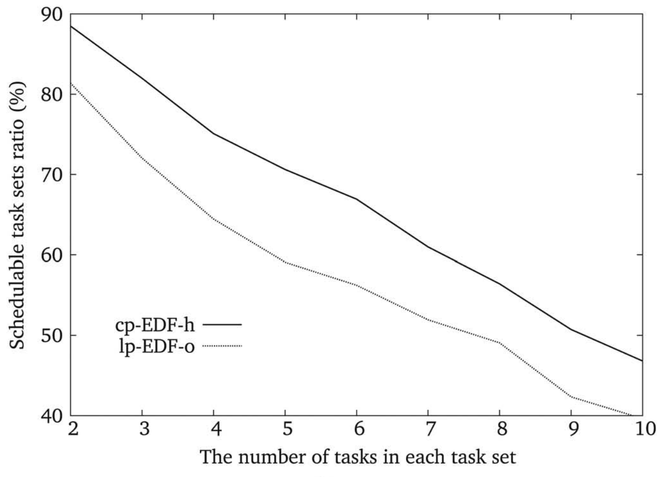


(d) Implicit deadline task sets with trimodal period distribution

Fig. 1. The number of schedulable sets by EDF under different preemption policies.



(a) $\alpha = 1$



(b) $\alpha = 4$

Fig. 2. Schedulable task set ratio according to the different number of tasks (for constrained task sets with uniform period distribution).

fp-EDF and np-EDF, and to lp-EDF with the optimal $\{p_i\}$ assignment in some environments. Second, since cp-EDF and lp-EDF with the optimal assignment of the corresponding parameters do not dominate each other, depending on task sets, we improve the scheduling performance by testing offline each task set and then selecting the best. Finally, at only a negligible performance loss, cp-EDF with the optimal parameter assignment can be replaced by that with the heuristic one, which requires a lower time-complexity—pseudo polynomial in the task parameters.

6 Conclusion

In this paper, we have identified the need of new preemption policies that control preempting jobs for better EDF schedulability under a given preemption delay, and developed the CP policy. Then, we have demonstrated that cp-EDF finds additional schedulable task sets, which are deemed unschedulable by EDF under existing preemption policies.

We have focused on predictable processors without any cache, which have already been deployed in embedded computing systems, and thus abstracted the preemption delay as a unified value α. However, if we consider general-purpose processors, the preemption delay includes the time required to save and reload cache, and then can be differentiated by (i) which job preempts and (ii) which job is preempted, or (iii) both of (i) and (ii) [5], [7], [13], [25]. As of now, cp-EDF and lp-EDF can immediately accommodate a preemption delay specified by (i) and (ii), respectively, but each of them is to deal with (iii) in a pessimistic manner in that cp-EDF and lp-EDF do not differentiate a preemption delay specified by (ii) and (i), respectively. In future, we plan to develop a new preemption policy that controls both preempting and preempted jobs so effective that it dominates EDF under all preemption policies including cp-EDF and lp-EDF.

Acknowledgment

Jinkyu Lee is the corresponding author. This work was supported in part by the NSF under Grants CNS-0930813 and CNS-1138200 and in part by the National Research Foundation of Korea Grant funded by the Korean Government (Ministry of Education, Science and Technology) [NRF-2011-357-D00186].

References

[1] S. Altmeyer, R. Davis, and C. Maize, "Cache related preemption delay aware response time analysis for fixed priority preeptive systems," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, 2011, pp. 261–271.

[2] T. P. Baker, "Comparison of empirical success rates of global vs. paritioned fixed-priority and EDF scheduling for hand real time," Dept. Comput. Sci., Florida State Univ., Tallahasee, FL, Tech. Rep. TR-050601, 2005.

[3] S. Baruah, "The limited-preemption uniprocessor scheduling of sporadic task systems," in *Proc. Euromicro Conf. Real-Time Syst. (ECRTS)*, 2005, pp. 137–144.

[4] S. Baruah, A. Mok, and L. Rosier, "Preemptively scheduling hard-real-time sporadic tasks on one processor," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, 1990, pp. 182–190.

[5] A. Bastoni, B. B. Brandenburg, and J. H. Anderson, "Cache-related preemption, and migration delays: Empirical approximation, and impact on schedulability," in *Proc. 6th Int. Workshop Oper. Syst. Platforms Embedded Real-Time Appl.*, 2010, pp. 33–44.

[6] M. Bertogna and S. Baruah, "Limited preemption EDF scheduling of sporadic task systems," *IEEE Trans. Ind. Informat.*, vol. 6, no. 4, pp. 579–591, Nov. 2010.

[7] M. Bertogna, G. Buttazzo, M. Marinoni, and M. Caccamo, "Preemption points placement for sporadic task sets," in *Proc. Euromicro Conf. Real-Time Syst. (ECRTS)*, 2010, pp. 251–260.

[8] M. Bertogna, G. Buttazzo, and G. Yao, "Improving feasibility of fixed priority tasks using non-preemptive regions," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, 2011, pp. 251–260.

[9] M. Bertogna, O. Xhani, M. Marinoni, F. Esposito, and G. Buttazzo, "Optimal selection of preemption points to minimize preemption overhead," in *Proc. Euromicro Conf. Real-Time Syst. (ECRTS)*, 2011, pp. 217–227.

[10] M. Dertouzos, "Control robotics: The procedural control of physical processors," in *Proc. IFIP Congr.*, 1974, pp. 807–813.

[11] L. George, P. Muhlethaler, and N. Rivierre, "Optimality and non-preemptive real-time scheduling revisited," Tech. Rep.: RR-2516, INRIA, 1995.

[12] L. George, N. Rivierre, and M. Spuri, "Preemptve and non-preemptive real-time uniprocessor scheduling," Tech. Rep.: RR-2966, INRIA, 1996.

[13] L. Ju, S. Chakraborty, and A. Roychoudhury, "Accounting for cache-related preemption delay in dynamic priority schedulability analysis," in *Proc. Des. Autom. Test Eur. Conf. Exhib. (DATE)*, 2007, pp. 1623–1628.

[14] C.-G. Lee, J. Hahn, Y.-M. Seo, S. L. Min, R. Ha, S. Hong, C. Y. Park, M. Lee, and C. S. Kim, "Analysis of cache-related preemption delay in fixed-priority preemptive scheduling," *IEEE Trans. Comput.*, vol. 47, no. 6, pp. 700–713, Jun. 1998.

[15] C.-G. Lee, K. Lee, J. Hahn, Y.-M. Seo, S. L. Min, R. Ha, S. Hong, C. Y. Park, M. Lee, and C. S. Kim, "Bounding cache-related preemption delay for real-time systems," *IEEE Trans. Softw. Eng.*, vol. 27, no. 9, pp. 805–826, Sep. 2001.

[16] J. Lee, A. Easwaran, and I. Shin, "Maximizing contention-free executions in multiprocessor scheduling," in *Proc. IEEE Real-Time Technol. Appl. Symp. (RTAS)*, 2011, pp. 235–244.

[17] J. Y.-T. Leung, "A new algorithm for scheduling periodic, real-time tasks," *Algorithmica*, vol. 4, pp. 209–219, 1989.

[18] C. L. Liu and J. Layland, "Scheduling algorithms for multi-programming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[19] J. Liu. *Real-Time Systems*. Englewood Cliffs, NJ: Prentice-Hall, 2000.

[20] A. Mok, "Fundamental design problems of distributed systems for the hard-real-time environment," Ph.D. dissertation, Massachusetts Instit. Technol., Cambridge, MA, 1983.

[21] J. Staschulat, S. Schliecker, and R. Ernst, "Scheduling analysis of real-time systems with precise modeling of cache related preemption delay," in *Proc. Euromicro Conf. Real-Time Syst. (ECRTS)*, 2005, pp. 41–48.

[22] M. Bertogna, M. Cirinei, and G. Lipari, "Schedulability analysis of global scheduling algorithms on multiprocessor platforms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 4, pp. 553–566, Apr. 2009.

[23] L. Thiele and R. Wilhelm, "Design of systems with predictable behavior," in *Proc. Abstracts Collect. Perspectives Workshop 03471 (Dagstuhl Seminars)*, 2004, pp. 1–8.

[24] Y. Wang and M. Saksena, "Scheduling fixed-priority tasks with preemption threshold," in *Proc. IEEE Int. Conf. Embedded Real-Time Comput. Syst. Appl. (RTCSA)*, 1999, pp. 318–335.

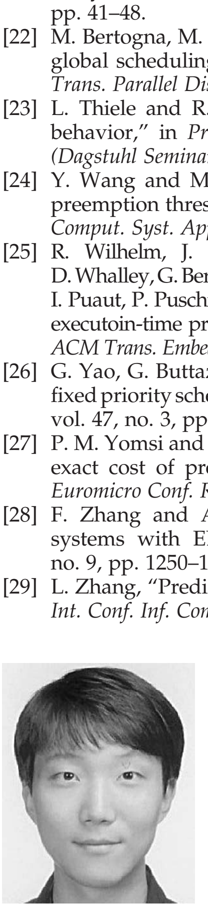
[25] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The worst-case executoin-time problem - overview of methods and survey of tools," *ACM Trans. Embedded Comput. Syst.*, vol. 7, no. 3, pp. 36:1–36:53, 2008.

[26] G. Yao, G. Buttazzo, and M. Bertogna, "Feasibility analysis under fixed priority scheduling with limited preemptions," *Real-Time Syst.*, vol. 47, no. 3, pp. 198–223, 2011.

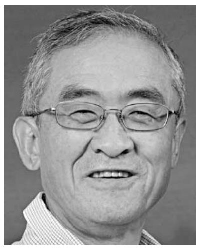
[27] P. M. Yomsi and Y. Sorel, "Extending rate monotonic analysis with exact cost of preemptions for hard real-time systems," in *Proc. Euromicro Conf. Real-Time Syst. (ECRTS)*, 2007, pp. 280–290.

[28] F. Zhang and A. Burns, "Schedulability analysis for real-time systems with EDF scheduling," *IEEE Trans. Comput.*, vol. 58, no. 9, pp. 1250–1258, Sep. 2009.

[29] L. Zhang, "Predictable architecture for real-time systems," in *Proc. Int. Conf. Inf. Commun. Signal Process.*, 1997, pp. 1761–1765.



Jinkyu Lee received the BS, MS, and PhD degrees in computer science from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, in 2004, 2006, and 2011, respectively. He is an assistant professor in Department of Computer Science and Engineering, Sungkyunkwan University, South Korea, where he joined in 2014. He has been a research fellow/visiting scholar in the Department of Electrical Engineering and Computer Science, University of Michigan until 2014. His research interests include system design and analysis with timing guarantees, QoS support, and resource management in real-time embedded systems and cyber-physical systems. He won the best student paper award from the 17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) in 2011, and the Best Paper Award from the 33rd IEEE Real-Time Systems Symposium (RTSS) in 2012.



Kang G. Shin is the Kevin & Nancy O'Connor professor of Computer Science with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor. His current research interests include QoS-sensitive computing and networking as well as embedded real-time and cyber-physical systems. He has supervised the completion of 74 PhDs, and authored/coauthored about 800 technical articles (about 300 of these are in archival journals), a textbook and more than 20 patents or invention disclosures, and received numerous best paper awards, including the Best Paper Awards from the 2011 ACM International Conference on Mobile Computing and Networking (MobiCom'11), the 2011 IEEE International Conference on Autonomic Computing, the 2010 and 2000 USENIX Annual Technical Conferences, as well as the 2003 IEEE Communications Society William R. Bennett Prize Paper Award and the 1987 Outstanding IEEE Transactions of Automatic Control Paper Award. He has also received several institutional awards, including the Research Excellence Award in 1989, Outstanding Achievement Award in 1999, Distinguished Faculty Achievement Award in 2001, and Stephen Attwood Award in 2004 from The University of Michigan (the highest honor bestowed to Michigan Engineering faculty); a Distinguished Alumni Award of the College of Engineering, Seoul National University in 2002; 2003 IEEE RTC Technical Achievement Award; and 2006 Ho-Am Prize in Engineering (the highest honor bestowed to Korean-origin engineers).