Contention-Free Executions for Real-Time Multiprocessor Scheduling

JINKYU LEE, University of Michigan
ARVIND EASWARAN, Nanyang Technological University
INSIK SHIN, KAIST

A time slot is defined as *contention-free* if the number of jobs with remaining executions in the slot is no larger than the number of processors, or *contending*, otherwise. Then an important property holds that in any contention-free slot, all jobs with remaining executions are guaranteed to be scheduled as long as the scheduler is work-conserving. This article aims at improving schedulability by utilizing the contention-free slots. To achieve this, this article presents a policy (called CF policy) that moves some job executions from contending slots to contention-free ones. This policy can be employed by any work-conserving, preemptive scheduling algorithm, and we show that any algorithm extended with this policy dominates the original algorithm in terms of schedulability. We also present improved schedulability tests for algorithms that employ this policy, based on the observation that interference from jobs is reduced when their executions are postponed to contention-free slots. Simulation results demonstrate that the CF policy, incorporated into existing algorithms, significantly improves schedulability of those existing algorithms.

Categories and Subject Descriptors: C.3 [**Computer Systems Organizations**]: Special-Purpose and Application-Based Systems—*Real-time and embedded systems*; D.4.1 [**Operating Systems**]: Process Management—*Scheduling*

General Terms: Algorithms, Design, Performance

Additional Key Words and Phrases: Contention-free slot, contention-free policy, multiprocessor scheduling, schedulability analysis, real-time embedded systems

ACM Reference Format:
Jinkyu Lee, Arvind Easwaran, and Insik Shin. 2014. Contention-free executions for real-time multiprocessor scheduling. ACM Trans. Embedd. Comput. Syst. 13, 2s, Article 69 (January 2014), 25 pages.
DOI: http://dx.doi.org/10.1145/2494530

1. INTRODUCTION

Embedded systems are computing systems designed for specialized hardware and/or software, and usually run with limited hardware resources for limited functions. These systems have been widely deployed in automobiles, avionics, aerospace industries, and medical devices. In particular, there is an increasing move towards implementing embedded systems upon multiprocessor (multicore) platforms, since the use of such

J. Lee is currently affiliated with Sungkyunkwan University.
This is an extended version of Lee et al. [2011a].

This work was supported in part by BRL (2009-0086964), BSRP (2010-0006650, 2012-R1A1A1014930), NIPA (H0503-12-1041), KEIT (2011-10041313), DGIST CPS Global Center, and KIAT (M002300089) funded by the Korea Government (MEST/MKE). This work was also supported by an NTU Start-Up Grant.

Authors' addresses: J. Lee, Department of Computer Science and Engineering, Sungkyunkwan University, South Korea; email: jinkyu.yi@gmail.com; A. Easwaran, School of Computer Engineering, Nanyang Technological University, Singapore; email: arvinde@ntu.edu.sg; I. Shin (corresponding author), Department of Computer Science, KAIST, South Korea; email: insik.shin@cs.kaist.ac.kr.

DOI: http://dx.doi.org/10.1145/2494530

a parallel architecture provides many benefits, including greater computing power at lower cost and greater energy efficiency. Multicore architecture has been supported by AUTOSAR in the automotive domain [Autosar 2009], and smartphones are getting equipped with multicore processors [van Berkel 2009].

Many embedded systems often involve time-critical control tasks in which their correctness depends not only on the logical correctness but also on the timeliness. In order to guarantee the timely-correctness, real-time scheduling has been studied extensively. In particular, real-time scheduling on uniprocessor platforms has successfully matured over the years. Key results include the EDF (earliest deadline first) [Liu and Layland 1973] scheduling algorithm that has been proved as optimal [Dertouzos 1974; Baruah et al. 1990]. However, real-time multiprocessor scheduling theories have not matured yet. Even though a great deal of scheduling algorithms have been proposed to improve scheduling performance on multiprocessors (see [Davis and Burns 2011b] for an excellent survey), there is no known scheduling algorithm that outperforms all other existing scheduling algorithms under general task models.

There has been a thread of research on the development of prioritization policies that can be incorporated into any existing scheduling algorithms orthogonally to improve schedulability on multiprocessors effectively. A successful example of such policies is the zero-laxity policy that promotes the priority of jobs with the zero-laxity, where a laxity of a job at any time is defined as the remaining time to the deadline minus the remaining execution time [Cho et al. 2002; Lee et al. 2011b]. Since the zero-laxity policy executes jobs that would otherwise inevitably miss their deadlines, a scheduling algorithm that employs the policy always produces better schedules than those by the original scheduling algorithm.

In this article, we develop a new policy that can be composed with any scheduling algorithms for schedulability improvement. To this end, we define and utilize a new notion of *contention-free* slots, which occur when the number of active jobs (i.e., jobs with remaining executions) is no larger than the number of processors. An interesting property of these slots is that all active jobs are guaranteed to be scheduled in the slots, as long as any work-conserving algorithm is used. By work-conserving, we mean an algorithm that always schedules active jobs if any processor is available. Using the notion of contention-free slots, we may move some job executions from *contending* slots (i.e., non-contention-free slots) to contention-free ones, so that we reduce the number of competing jobs in contending slots. To safely move executions, we introduce a policy under which a job postpones its remaining executions to contention-free slots whenever the number of contention-free slots up to its deadline is at least as many as remaining executions. We call this policy of deferring executions to contention-free slots the CF (Contention-Free) policy. Observe that the CF policy only postpones those job executions that are guaranteed to be scheduled before the job deadline. Thus, the policy reduces competing jobs by postponing some of them to contention-free slots, without imposing any penalties on the schedulability of the postponed jobs. This policy can be incorporated into any existing work-conserving preemptive algorithm A, and in this article, we denote such a CF-based algorithm as A-CF. The CF policy has an important property that algorithm A-CF dominates its base algorithm A, which means any task set schedulable by A is also schedulable by A-CF.

However, the CF policy requires a job to know the number of contention-free slots up to its deadline, which depends on job schedules of currently active jobs and jobs to be released in the future. We want to make the CF policy feasible without relying on such runtime and future information. Therefore, we introduce two methods for calculating lower bounds on the number of contention-free slots of each job, and each method only requires offline information. Then, the lower bounds can be given to each job offline, and then the CF policy operates based on the lower bounds.

Although just incorporating the CF policy is sufficient to improve the schedulability of any base algorithm, the true potential of the policy will only be realized when a corresponding improvement in the schedulability tests is achieved. That is, without schedulability tests, we cannot guarantee the predictability of the target system. Hence, using the notion of contention-free slots, we develop an interference reduction technique for the CF policy, which can be incorporated into existing interference-based schedulability tests for the base algorithms (e.g., [Bertogna et al. 2005, 2009; Bertogna and Cirinei 2007; Baruah 2007; Baker et al. 2008; Lee et al. 2010, 2011b]). As examples, we introduce schedulability tests for EDF-CF and EDZL-CF, based on existing tests for global EDF [Bertogna et al. 2009, 2005] and EDZL (earliest deadline first until zero laxity) [Baker et al. 2008], and demonstrate by simulations that the proposed tests significantly improve the existing tests. We want to emphasize that the principle of this interference reduction is generally applicable to any existing interference-based schedulability tests of other scheduling algorithms.

One potential drawback of the CF policy is it may incur additional preemptions by deferring executions. We derive an upper bound on the number of times each job is preempted under CF-based algorithms. Since the upper bound depends on base algorithms, we also choose EDF and EDZL and compare the upper bound under the base algorithms with that under EDF-CF and EDZL-CF, respectively. Simulation results of the upper bounds indicate that the CF policy, when incorporated into EDF and EDZL, does not incur more than 20% additional preemptions in any case. We also measure the actual number of preemptions incurred by each task set and demonstrate that the CF policy incurs only at most 1% additional actual preemptions on average, compared to their base algorithms.

In summary, this article makes the following contributions.

—It defines a novel notion of contention-free slots in multiprocessor real-time scheduling.
—It derives how to calculate a lower bound on the number of contention-free slots of each job, only using offline information.
—Using the lower bound, it proposes a novel concept of the CF policy, which can be incorporated into any work-conserving preemptive algorithm, and has a property that algorithm *A-CF* dominates algorithm *A* in terms of schedulability.
—It develops schedulability tests for CF-based algorithms by introducing an interference reduction technique and applying it to existing interference-based schedulability tests for corresponding base algorithms.
—It analyzes an upper bound on the number of times each job is preempted under CF-based algorithms and demonstrates that additional preemptions incurred by the CF policy itself are not considerable.

The rest of this article is organized as follows. Section 2 describes our system model. Section 3 defines the notion of contention-free slots and calculates a lower bound on the number of contention-free slots of each job in two ways. Section 4 proposes the CF policy with its properties. Section 5 develops schedulability tests for CF-based algorithms, and Section 6 analyzes the number of preemptions for CF-based algorithms. Section 7 demonstrates the effectiveness of the CF policy through simulations. Section 8 discusses the applicability of the CF policy to existing scheduling algorithms and analyses. Section 9 summarizes the related work, and finally Section 10 concludes.

2. SYSTEM MODEL

In this article, we focus on the sporadic task model [Mok 1983]. In this model, we specify task $\tau_i \in \mathcal{T}$ as (T_i, C_i, D_i), where T_i is the minimum separation, C_i is the worst-case execution time requirement, and D_i is the relative deadline. Further, we restrict our attention to a constrained deadline task system, that is, $C_i \leq D_i \leq T_i$ for each task $\tau_i \in \mathcal{T}$. Task τ_i invokes a series of jobs, each separated from its predecessor by at least T_i time units. We denote job $J_{i,q}$ as the q^{th} job of τ_i, and $r_{i,q}$ and $r_{i,q} + D_i$ as the release time and the deadline of job $J_{i,q}$, respectively. We assume that a single job of a task cannot be executed in parallel. We use $C_i(t)$ to denote the remaining execution time of a job of τ_i at time t, and this quantity is well defined, since we focus on constrained deadline task systems. Also, we call a job *active* at t if it has remaining executions at t (i.e., $C_i(t) > 0$). We denote the total number of tasks as n.

In this article, we assume that the platform comprises m identical unit-capacity processors. We also assume quantum-based time, and without loss of generality, let one time unit (or time slot) denote the quantum length. All task parameters are assumed to be specified as multiples of this quantum length.

3. THE CONTENTION-FREE SLOT

In this section, we define and identify contention-free slots. First, a new notion of a contention-free slot in real-time scheduling is defined as follows.

Definition 3.1. A time slot is *contention-free* if and only if the number of active jobs (i.e., jobs with remaining executions) in the slot is equal to or less than the number of processors (m). As opposed to a contention-free slot, a time slot is *contending* if and only if the slot is not contention-free.

Then, a contention-free slot has the following property.

LEMMA 3.2. *In any contention-free slot, all active jobs are guaranteed to be executed under any work-conserving scheduling algorithm.*

PROOF. The lemma holds by the definition of the contention-free slot. □

Our goal is to find better schedules by utilizing contention-free slots; a job delays its executions whenever the number of contention-free slots up to its deadline is at least as many as remaining executions (Section 4 will explain this in detail). To do this, job $J_{i,q}$ should know the number of contention-free slots in an interval between the current time t and its deadline $r_{i,q} + D_i$, and this requires the following information.

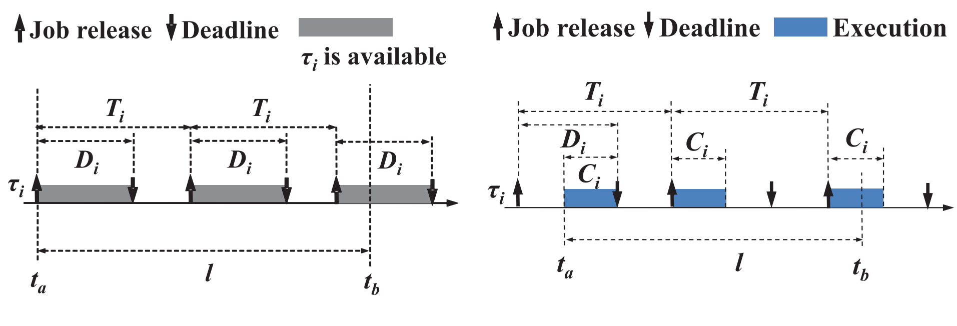
(a) Job parameters of all jobs active at t (deadline and remaining execution time).
(b) Job parameters of all jobs released in $[t, r_{i,q} + D_i)$ (release time, deadline, and execution time).
(c) Job schedules in $[t, r_{i,q} + D_i)$.

Note that (a) is tractable only at runtime; (b) cannot be obtained without knowing future release patterns, which is impossible for a sporadic task model; and (c) depends on scheduling algorithms and (a) and (b).

We want to calculate the number of contention-free slots of each job without relying on runtime and future information and without depending on scheduling algorithms. In the next sections, we introduce two methods for deriving a lower bound on the number of contention-free slots in an interval of length l, which only requires knowledge of task parameters $\{(T_i, C_i, D_i)\}_{\tau_i \in \mathcal{T}}$. Then, we present how to use the two methods to obtain the number of contention-free slots for each job.

3.1. A Lower Bound on the Number of Contention-Free Slots

In this section, we calculate a lower bound on the number of contention-free slots in an interval of length l, based on the observation that there is no execution between the deadline of a job of a task and the release time of the next job of the task. We express that job $J_{i,q}$ is *available* at a time instant t if the instant is included in the



(a) Available slots are maximized. (b) Executed slots are maximized.

Fig. 1. Situations when the time for jobs of τ_i to be available and executed is maximized in an interval of length l.

interval between its release time and deadline, that is, $t \in [r_{i,q}, r_{i,q} + D_i)$. Since a job can be active only when it is available, we observe the following relationship between the number of active and available jobs.

Observation 1. The number of active jobs at t does not exceed the number of available jobs at t.

Therefore, we can derive a lower bound on the number of contention-free slots in an interval of length l by counting the number of slots in which the number of available jobs is less than or equal to m. Now we explain how to lower bound the number of such slots by using task parameters only.

We first calculate the number of slots for jobs of a task to be available in a given interval. In an interval of length l, jobs of τ_i are available during at most $\zeta_i(l)$ slots, where

$$\zeta_i(l) = \left\lfloor \frac{l}{T_i} \right\rfloor D_i + \min\left(D_i, l - \left\lfloor \frac{l}{T_i} \right\rfloor T_i\right). \tag{1}$$

This is because the number of slots in which jobs of τ_i is available is maximized when the release time of the first job of τ_i in the interval $[t_a, t_b)$ of length l is the same as the beginning of the interval, as shown in Figure 1(a). Then, $\lfloor \frac{l}{T_i} \rfloor$ in Eq. (1) calculates how many jobs of τ_i have their release time and deadline within an interval of length l (e.g., $\lfloor \frac{l}{T_i} \rfloor = 2$ in Figure 1(a)); $\lfloor \frac{l}{T_i} \rfloor D_i$ represents the number of available slots of those jobs. The second term in the right-hand side of Eq. (1) calculates the number of available slots of a job whose release time is within the interval but whose deadline is not (e.g., the third job in Figure 1(a)).

By Definition 3.1 and Observation 1, at least $m + 1$ available jobs exist for a contending slot. Then, using $\zeta_i(l)$, we can calculate an upper bound on the number of contending slots in an interval of length l, and equivalently, we derive a lower bound on the number of contention-free slots in the interval as follows.

Lemma 3.3. *In an interval of length l, the number of contention-free slots is at least as many as* $\Phi(l)$, *where*

$$\Phi(l) = \max\left(0, l - \left\lfloor \frac{\sum_{\tau_i \in T} \zeta_i(l)}{m+1} \right\rfloor\right). \tag{2}$$

Proof. If there are at most m available jobs in a time slot, the slot has at most m active jobs by Observation 1, which means the slot is contention-free by Definition 3.1. This means at least $m + 1$ available jobs are required for a slot to be contending.

Thus, there exist at most $\min(l, \lfloor \frac{\sum_{\tau_i \in \mathcal{T}} \zeta_i(l)}{m+1} \rfloor)$ contending slots in an interval length of l, which is equivalent to saying that the number of contention-free slots is at least $l - \min(l, \lfloor \frac{\sum_{\tau_i \in \mathcal{T}} \zeta_i(l)}{m+1} \rfloor) = \Phi(l)$. □

3.2. An Improved Lower Bound on the Number of Contention-Free Slots

The lower bound derived in Lemma 3.3 is pessimistic in that it does not consider how many executions are required by each job; in other words, Lemma 3.3 only utilizes task parameters $\{T_i\}_{\tau_i \in \mathcal{T}}$ and $\{D_i\}_{\tau_i \in \mathcal{T}}$, and does not consider $\{C_i\}_{\tau_i \in \mathcal{T}}$. Now, we present another way of computing a lower bound on the number of contention-free slots in an interval of length l by considering how many executions are required in the interval.

We first calculate how many executions of jobs of a task can be performed in an interval of length l. An upper bound on the execution time of jobs of task τ_i in an interval of length l is depicted in Figure 1(b) [Bertogna and Cirinei 2007; Bertogna et al. 2009]. Given an interval $[t_a, t_b)$ of length l, the first job of τ_i begins its execution at t_a and completes the execution at $t_a + C_i$. Here, $t_a + C_i$ is also the deadline of the first job. Thereafter, jobs are released and scheduled as soon as possible. We let $\eta_i(l)$ denote the number of jobs of τ_i that can execute completely within the interval of interest (including the first job) as follows.

$$\eta_i(l) = \left\lfloor \frac{l - (C_i + T_i - D_i)}{T_i} \right\rfloor + 1 = \left\lfloor \frac{l + D_i - C_i}{T_i} \right\rfloor. \tag{3}$$

The execution time of the last job of τ_i can then be upper bounded by $\min(C_i, l + D_i - C_i - \eta_i(l) \cdot T_i)$. Then, an upper bound on the execution time of jobs of τ_i during an interval of length l is therefore [Bertogna and Cirinei 2007; Bertogna et al. 2009]

$$\zeta'_i(l) = \eta_i(l) \cdot C_i + \min(C_i, l + D_i - C_i - \eta_i(l) \cdot T_i). \tag{4}$$

Using $\zeta'_i(l)$, we can derive another lower bound on the number of contention-free slots in an interval of length l as follows.

LEMMA 3.4. *Assume the scheduling algorithm is work-conserving. In an interval of length l, the number of contention-free slots is at least as many as $\Phi'(l)$, where*

$$\Phi'(l) = \max\left(0, l - \left\lfloor \frac{\sum_{\tau_i \in \mathcal{T}} \zeta'_i(l)}{m} \right\rfloor\right). \tag{5}$$

PROOF. Since the scheduling algorithm is work-conserving, a time slot is contention-free if at most $m-1$ jobs are executed in the time slot. Note that if m jobs are executed, we do not know whether there are exactly m active jobs or more than m active jobs. This means, for a time slot to be contending, at least m jobs should be executed in the time slot. Thus, there exist at most $\min(l, \lfloor \frac{\sum_{\tau_i \in \mathcal{T}} \zeta'_i(l)}{m} \rfloor)$ contending slots in an interval length of l, which is equivalent to saying that the number of contention-free slots is at least $l - \min(l, \lfloor \frac{\sum_{\tau_i \in \mathcal{T}} \zeta'_i(l)}{m} \rfloor) = \Phi'(l)$. □

3.3. The Number of Contention-Free Slots of Each Job

So far, we developed Lemmas 3.3 and 3.4 to calculate lower bounds on the number of contention-free slots in an interval of length l. Now, we explain how to use the lemmas to compute a lower bound on the number of contention-free slots between the release time and deadline of a job invoked by τ_k (denoted by ϕ_k).

By Lemmas 3.3 and 3.4, we may simply calculate ϕ_k by choosing the maximum of the numbers by Eqs. (2) and (5) with $l = D_k$ (i.e., $\phi_k = \max(\Phi(D_k), \Phi'(D_k))$), but we can improve $\Phi'(D_k)$. That is, since we focus on an interval between the release time and deadline of a job invoked by τ_k, the execution time of the job is exactly C_k, not $\zeta'_k(D_k)$ (note that $C_k \leq \zeta'_k(D_k)$). Therefore, we can improve $\Phi'(D_k)$ by replacing $\zeta'_k(D_k)$ with C_k in Eq. (5), and the following lemma records this.

LEMMA 3.5. *Assume the scheduling algorithm is work-conserving. In an interval between the release time and deadline of a job invoked by τ_k, the number of contention-free slots is at least as many as $\phi_k = \max(\Phi(D_k), \Phi''(D_k))$, where $\Phi(D_k)$ is defined in Eq.* (2), *and*

$$\Phi''(D_k) = \max\left(0, D_k - \left\lfloor \frac{C_k + \sum_{\tau_i \in \mathcal{T}-\{\tau_k\}} \zeta'_i(D_k)}{m} \right\rfloor\right). \tag{6}$$

PROOF. The lemma holds by Lemmas 3.3 and 3.4. □

Note that for τ_k, neither $\Phi(D_k)$ nor $\Phi''(D_k)$ dominates the other, and the following properties hold for them.

—In the numerator of the floor function, $\zeta'_i(D_k) \leq \zeta_i(D_k)$ (for $i = k$, $C_k \leq \zeta_i(D_k)$) for all $\tau_i \in \mathcal{T}$.
—In the denominator of the floor function, $m < m + 1$.

As m gets larger, the ratio $m/(m+1)$ converges to one, and therefore, the difference between the numerators for $\Phi(D_k)$ and $\Phi''(D_k)$ is more pronounced. In addition, given $\{D_i\}_{\tau_\in\mathcal{T}}$ and $\{T_i\}_{\tau_\in\mathcal{T}}$, $\Phi(D_k)$ is invariant, but $\Phi''(D_k)$ gets larger as $\{C_i\}_{\tau_\in\mathcal{T}}$ gets smaller. Therefore, in most cases (not all cases), $\Phi''(D_k)$ is equal to or smaller than $\Phi(D_k)$, which means $\Phi''(D_k)$ is more useful. Here are examples of calculating $\phi_k = \min(\Phi(D_k), \Phi''(D_k))$.

We consider seven tasks on a four-processor platform: $\tau_1 = \tau_2 = \tau_3 = \tau_4 = (10, 1, 6)$, $\tau_5 = \tau_6 = (10, 6, 7)$, and $\tau_7 = (10, 6, 10)$. Then $\Phi(D_7) = 10 - \lfloor\frac{6+6+6+6+7+7+10}{4+1}\rfloor = 1$, but $\Phi''(D_7) = 10 - \lfloor\frac{2+2+2+2+7+7+6}{4}\rfloor = 3$. Therefore. $\phi_7 = \max(1, 3) = 3$, which means a job of τ_7 has at least three contention-free slots between its release time and deadline.

If we consider another task set with reducing the execution time of τ_7 by one (i.e., $\tau'_7 = (10, 5, 10)$), then $\Phi(D_7)$ is still 1, but $\Phi''(D_7)$ is $10 - \lfloor\frac{2+2+2+2+7+7+5}{4}\rfloor = 4$. Then, $\phi_7 = \max(1, 4) = 4$, which means a job of τ_7 has at least four contention-free slots between its release time and deadline.

We want to emphasize that ϕ_k in Lemma 3.5 depends only on task parameters (T_i, C_i, D_i) of each task $\tau_i \in \mathcal{T}$, and therefore, ϕ_k is independent of underlying scheduling algorithms. Due to this property, contention-free slots can be broadly utilized for better schedulability regardless of underlying scheduling algorithms, which will be detailed in the next section.

4. THE CONTENTION-FREE POLICY

In this section, we present the contention-free (CF) policy. We first give a formal description of the CF policy. Then, we present and prove important properties of the CF policy.

4.1. Description of the CF Policy

From Lemma 3.5, when a job of τ_k is released, it knows there are at least ϕ_k contention-free slots between its release time and deadline, but it does not know when it will encounter those slots. Under this limited information, we want to shift executions from contending slots to contention-free slots for better schedulability. Also, we want to do this shifting under an important principle that it does not make any previously schedulable job unschedulable. For this purpose, we maintain a variable that stores the remaining number of contention-free slots for a job of τ_i at time t (denoted by $\phi_i(t)$).

ALGORITHM 1: *A-CF* Scheduling Algorithm

```
(1) For each job $J_{i,q}$ released at $t$,
    (a) Set $\phi_i(t) \leftarrow \phi_i$ $(= \max(\Phi(D_i), \Phi''(D_i)))$.
    (b) Set $C_i(t) \leftarrow C_i$.
    (c) Put the job in $Q_H$.
(2) For each job $J_{i,q}$ in $Q_H$,
    (a) If $\phi_i(t) \geq C_i(t)$, move the job to $Q_L$.
(3) If $N(t) = |Q_H| + |Q_L| \leq m^*$, for each job $J_{i,q}$ in $Q_H$,
    (a) Update $\phi_i(t+1) \leftarrow \max(0, \phi_i(t) - 1)$.
(4) Prioritize jobs in $Q_H$ separately according to the base algorithm $A$.
(5) For each job $J_{i,q}$ chosen among the $m$ highest-priority jobs (considering any job in $Q_H$ has a
    higher priority than any job in $Q_L$),
    (a) Execute the job.
    (b) Update $C_i(t) \leftarrow C_i(t) - 1$.
    (c) If the job finishes its execution, remove the job from its queue.
```

*Here $|Q|$ denotes the number of jobs in Q.

When a job is released, $\phi_i(t)$ is set to ϕ_i $(= \max(\Phi(D_i), \Phi''(D_i)))$. Also whenever the number of active jobs is not larger than m (i.e., contention-free slot), $\phi_i(t)$ is reduced by one. Once the job satisfies $\phi_i(t) \geq C_i(t)$, its priority becomes the lowest, because the remaining executions can be successfully performed in contention-free slots (as long as the scheduling algorithm is work-conserving and preemptive). We denote this shifting policy as the CF (contention-free) policy. This policy can be incorporated into any existing work-conserving, preemptive scheduling algorithm, and we call such an extended algorithm a CF-based scheduling algorithm. Also, we denote such a CF-based algorithm, derived from a base algorithm A, as *A-CF*.

Algorithm 1 gives a formal description of how the CF policy operates with its base algorithm A at each time slot t. Here, two queues are maintained: the higher-priority queue (Q_H) and the lower-priority queue (Q_L). Note that Steps (1b), (1c), (4), and (5) can be required even when the CF policy is not incorporated in the base algorithm. The CF policy itself requires Steps (1a), (2), and (3), and thus additional time complexity is $O(n)$ for each t; at most one comparison and two updates for each job are additionally needed.

4.2. Properties of the CF Policy

In the previous section, the CF policy is designed under the principle that it does not make any schedulable job unschedulable. Based on this principle, we can easily derive a dominance relationship between any work-conserving, preemptive base scheduling algorithm A and the corresponding CF-based algorithm *A-CF*. To prove the dominance relationship, we introduce two properties of the CF policy in the following two lemmas: one holds when a job is in the higher-priority queue Q_H, and the other holds when a job is in the lower-priority queue Q_L.

LEMMA 4.1. *When job $J_{i,q}$ is in the higher-priority queue Q_H, executions of the job under algorithm* A-CF *are performed no later than the corresponding executions under algorithm* A.

PROOF. We consider the case where the kth execution of the job is performed in $[t, t+1)$ under algorithm A. Suppose the kth execution is not performed until t under algorithm *A-CF*. We claim that only those active jobs, which have a higher priority than $J_{i,q}$ under algorithm A at t, can have a higher priority than $J_{i,q}$ under algorithm *A-CF*. If the claim is true, then the kth execution of the job is performed in $[t, t+1)$.

We consider two types of active jobs at t under algorithm *A-CF*: jobs which are active under both A and *A-CF*, and jobs which are active only under *A-CF*. For jobs which are active under both algorithms, the claim holds, because active jobs in Q_H are prioritized by A and those in Q_L have a lower priority than $J_{i,q}$. For jobs which are active under *A-CF* but not under A, it holds that they are all in Q_L and hence have a lower priority than $J_{i,q}$. This follows from the fact that only a job in Q_L can have its executions postponed under *A-CF* in comparison to its executions under A. Thus, the claim is true, and this proves the lemma. □

LEMMA 4.2. *When job $J_{i,q}$ is in the lower-priority queue Q_L, the remaining executions are successfully performed before its deadline under algorithm* A-CF.

PROOF. Job $J_{i,q}$ migrates from Q_H to Q_L only when the number of remaining executions is no more than the number of remaining contention-free slots for this job up to its deadline. Since any active job is always executed in contention-free slots, it follows that all the remaining executions of $J_{i,q}$ will be successfully scheduled. Hence the lemma holds. □

Now, we prove a dominance property of the CF policy using the preceding two lemmas.

THEOREM 4.3. *If a task set is schedulable by algorithm* A *(which is work-conserving and preemptive), it is also schedulable by algorithm* A-CF.

PROOF. By Lemmas 4.1 and 4.2, any job meeting its deadline under algorithm A also finishes its executions within its deadline under algorithm *A-CF*. This proves the theorem. □

By Theorem 4.3, we know that the CF policy is not harmful to schedulability. In the next section, we introduce a technique to improve schedulability analysis of existing algorithms when the CF policy is incorporated.

5. SCHEDULABILITY ANALYSIS FOR CF-BASED SCHEDULING ALGORITHMS

In this section, we present schedulability analysis of CF-based scheduling algorithms. We first describe how the CF policy reduces interference from higher-priority jobs. Then, we show how the interference reduction improves existing schedulability tests. While we can apply this interference reduction technique to any existing interference-based schedulability tests (e.g., [Bertogna et al. 2005, 2009; Bertogna and Cirinei 2007; Baruah 2007; Baker et al. 2008; Lee et al. 2010, 2011b]), in this article, we demonstrate it on the EDF and EDZL tests presented in Bertogna et al. [2009, 2005] and Baker et al. [2008].

5.1. Interference Reduction

Many existing schedulability tests (e.g., [Bertogna et al. 2005, 2009; Bertogna and Cirinei 2007; Baruah 2007; Baker et al. 2008; Lee et al. 2010, 2011b]) decide schedulability using the concept of *interference*. Interference of job J_A on job J_B denotes the time duration in the interval between the release time and deadline of J_B when J_B is waiting to execute while J_A is executing. If the interference from all jobs during an interval of length D_k is enough to block the execution of a job of τ_k by more than $D_k - C_k$, the job is unschedulable.

When the CF policy is employed, we may reduce interference using the property of contention-free slots. That is, a job cannot interfere with other jobs in contention-free slots, because all active jobs in contention-free slots are always scheduled. Since the CF policy shifts some executions to contention-free slots, it reduces interference in contending slots. We now introduce a theorem that bounds the maximum interference that a single job of task τ_i can cause on any other job.

THEOREM 5.1. *Job $J_{i,q}$ can interfere with any job during at most $C_i - \phi_i$ time slots.*

PROOF. When the job is released, $\phi_i(t)$ is set to ϕ_i. We consider the following two cases.

Case 1. $J_{i,q}$ is in Q_H until its deadline. In this case, it should hold that $\phi_i(t) < C_i(t)$ for all t. This means $\phi_i(t) = 0$ after some t', where $C_i(t') = 1$. Then, $J_{i,q}$ encounters at least ϕ_i contention-free slots while it is active. In other words, at least ϕ_i executions of $J_{i,q}$ are performed in contention-free slots. Since no job interferes with other jobs in contention-free slots, we can conclude that $J_{i,q}$ interferes with other jobs during at most $C_i - \phi_i$ time slots.

Case 2. $J_{i,q}$ moves to Q_L at t', that is, $\phi_i(t') = C_i(t')$. Before t', $J_{i,q}$ encounters exactly $\phi_i - \phi_i(t')$ contention-free slots. After t', the job cannot interfere with other jobs in Q_H because $J_{i,q}$ is in Q_L. Hence, $C_i - C_i(t') - (\phi_i - \phi_i(t')) = C_i - \phi_i$ executions of $J_{i,q}$ can interfere with other jobs. □

5.2. Schedulability Analysis of EDF-CF and EDZL-CF

Theorem 5.1 indicates that we can reduce interference when the CF policy is employed. We now describe how this reduction can be applied to the existing EDF and EDZL schedulability tests presented in Bertogna et al. [2009, 2005] and Baker et al. [2008]. In other words, we introduce new schedulability tests for both EDF-CF and EDZL-CF, respectively.

Before introducing new schedulability tests, we briefly explain how individual scheduling algorithms operate. Global EDF (earliest deadline first) [Liu and Layland 1973] schedules jobs according to their deadlines; jobs with m earliest deadlines are chosen to execute. Global EDZL (earliest deadline first until zero laxity) [Lee 1994] assigns the highest priority to zero-laxity jobs and schedules remaining jobs using global EDF, where the laxity of a job at any time instant is defined as the remaining time to deadline minus the amount of remaining execution time. Then, EDF-CF and EDZL-CF schedule jobs according to Algorithm 1, prioritizing jobs in Q_H by EDF and EDZL, respectively. One may wonder how EDZL-CF operates, since EDZL itself involves two priority queues (for zero-laxity jobs and others), and the CF policy also yields two priority queues Q_H and Q_L. However, as shown in Algorithm 1, EDZL-CF maintains CF-related priority-queues Q_H and Q_L at a higher level; for prioritizing jobs in Q_H, two additional priority-queues for EDZL exist at a lower level for identifying zero-laxity jobs.

We first revisit the EDF schedulability test in Bertogna et al. [2009, 2005]. This test checks whether a job of τ_k has enough interference from other jobs to miss its deadline. Since finding the exact interference is difficult, it uses an upper bound based on the job release pattern depicted in Figure 2. Since a job with a later deadline cannot interfere with another job with an earlier deadline under EDF, an upper bound on the amount of interference of jobs of τ_i on a job of τ_k (denoted by $I_{k\leftarrow i}$) occurs when the deadlines of two jobs are aligned, as shown in the figure. Then $I_{k\leftarrow i}$ is calculated as follows.

$$I_{k\leftarrow i} = \left\lfloor \frac{D_k}{T_i} \right\rfloor C_i + \min\left(C_i, D_k - \left\lfloor \frac{D_k}{T_i} \right\rfloor T_i\right). \tag{7}$$

Using $I_{k\leftarrow i}$, the following lemma introduces a schedulability test for EDF.

LEMMA 5.2 (THEOREM 7 IN [BERTOGNA ET AL. 2009]). *A task set is schedulable under EDF if the following inequality holds for each task τ_k.*

$$\sum_{\tau_i \in \mathcal{T} - \{\tau_k\}} \min(I_{k\leftarrow i}, D_k - C_k + 1) < m \cdot (D_k - C_k + 1). \tag{8}$$

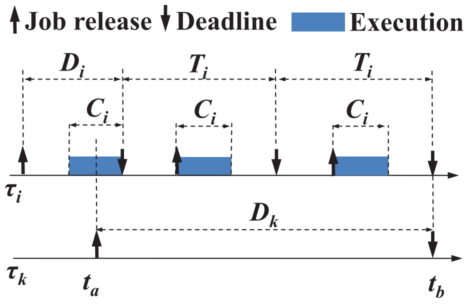


Fig. 2. Situation when the interference of jobs of τ_i on a job of τ_k is maximized under EDF or EDZL.

PROOF. The proof is given in Theorem 5 in Bertogna et al. [2009]. To summarize, for a job of τ_k to miss its deadline, the job is executed at most $C_k - 1$ time units. At each time slot, interference from at least m other jobs is needed to block τ_k's execution. Hence, if the total interference of other jobs on a job of τ_k is less than $m \cdot (D_k - (C_i - 1))$, the job cannot miss its deadline. □

By the dominance property of the CF policy (i.e., Theorem 4.3), we can use Lemma 5.2 as a schedulability test for EDF-CF. However, we now introduce a better schedulability test using the interference reduction technique presented earlier.

Theorem 5.1 indicates that at most $\max(0, C_i - \phi_i)$ executions of job $J_{i,q}$ interfere with other jobs. Instead of counting the whole execution time C_i as interference as in Eq. (7), we count only $\max(0, C_i - \phi_i)$ executions as interference. With this modification, Lemma 5.2 can be modified for EDF-CF as follows.

THEOREM 5.3. *A task set is schedulable under EDF-CF if the following inequality holds for each task τ_k.*

$$\sum_{\tau_i \in \mathcal{T} - \{\tau_k\}} \min\left(I^{\mathsf{CF}}_{k \leftarrow i}, D_k - C_k + 1\right) < m \cdot (D_k - C_k + 1), \tag{9}$$

where

$$I^{\mathsf{CF}}_{k \leftarrow i} = \left\lfloor \frac{D_k}{T_i} \right\rfloor C'_i + \min\left(C'_i, D_k - \left\lfloor \frac{D_k}{T_i} \right\rfloor T_i\right), \tag{10}$$

and $C'_i = \max(0, C_i - \phi_i)$.

PROOF. The theorem holds from Theorem 5.1 and Lemma 5.2. □

Similar to EDF-CF, we now introduce a new schedulability test for EDZL-CF based on the EDZL schedulability test presented in Baker et al. [2008]. The EDZL schedulability test proposed in Baker et al. [2008] uses the following observation: when a job misses its deadline under EDZL at t', there exist at least $m + 1$ jobs which have a zero laxity at t ($< t'$). Therefore, while the EDF test judges whether there is a single job with a deadline miss, the EDZL test judges whether there are at least $m + 1$ jobs with a zero-laxity state. Then, similar to Lemma 5.2, the schedulability test for EDZL [Baker et al. 2008] is given as follows.

LEMMA 5.4 (THEOREM 7 IN [BAKER ET AL. 2008]). *A task set is schedulable under EDZL if the following inequality holds for at least $n - m$ tasks.*

$$\sum_{\tau_i \in \mathcal{T} - \{\tau_k\}} \min(I_{k \leftarrow i}, D_k - C_k) < m \cdot (D_k - C_k). \tag{11}$$

Proof. The proof is given in Theorem 7 in Baker et al. [2008]. In summary, for a job of τ_k to reach a zero-laxity state, time to deadline should be the same as the remaining execution time. Hence, if the total interference of other jobs on a job of τ_k is less than $m \cdot (D_k - C_i)$, the job cannot reach a zero-laxity state. Therefore, any job invoked by tasks that satisfy Eq. (11) cannot have a zero laxity. If at least $n - m$ tasks satisfy Eq. (11), at most m jobs have a zero laxity, which means any job cannot miss its deadline under EDZL. □

Similar to the EDF-CF test, we derive a schedulability test for EDZL-CF by replacing C_i with $\max(0, C_i - \phi_i)$ in the following theorem.

Theorem 5.5. *A task set is schedulable under EDZL if the following inequality holds for at least $n - m$ tasks.*

$$\sum_{\tau_i \in \mathcal{T} - \{\tau_k\}} \min\left(I^{\mathsf{CF}}_{k \leftarrow i}(D_k), D_k - C_k\right) < m \cdot (D_k - C_k). \tag{12}$$

Proof. The theorem holds from Theorem 5.1 and Lemma 5.4. □

It is easy to see that Theorems 5.3 and 5.5 dominate Lemmas 5.2 and 5.4, respectively. Although we have chosen EDF-CF and EDZL-CF to demonstrate how the interference reduction technique can be applied to existing schedulability analyses, the idea can be generally extended to any existing schedulability analysis that calculates interference similar to Eq. (7), such as the ones for any work-conserving algorithm [Bertogna et al. 2009], fixed-priority scheduling [Bertogna et al. 2009], any ZL-based algorithm (that gives the highest priority to zero-laxity jobs) [Lee et al. 2011b], and LLF (least laxity first) [Lee et al. 2010]. Once the CF policy is incorporated into those schedulability tests, the principle of interference reduction of each job (i.e., $\max(0, C_i - \phi_i)$ instead of C_i) significantly improves the schedulability of base algorithms. We will discuss the applicability of the CF policy for existing schedulability analyses in Section 8.1.

Note that existing EDF and EDZL schedulability analyses (in Lemmas 5.2 and 5.4) as well as the proposed EDF-CF and EDZL-CF analyses (in Theorems 5.3 and 5.5) are sustainable [Burns and Baruah 2008] with respect to the minimum separations, the worst-case execution times, and the relative deadlines (i.e., $\{T_i, C_i, D_i\}_{\tau_i \in \mathcal{T}}$).

6. PREEMPTION ANALYSIS FOR CF-BASED SCHEDULING ALGORITHMS

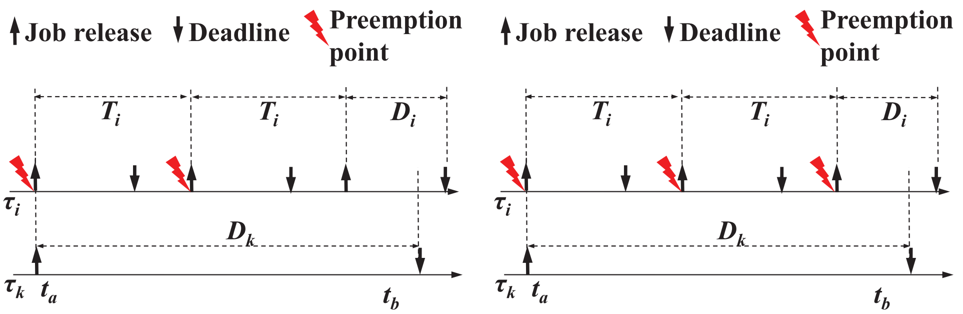
While the CF policy improves schedulability by moving executions in contending slots to contention-free ones, it may introduce additional preemptions. In this section, we analyze upper bounds of the number of times a job of a given task is preempted when the CF policy is incorporated into base algorithms. Since such an upper bound depends on base algorithms, we compare the upper bound under EDF-CF and EDZL-CF, with that under their corresponding base algorithms EDF and EDZL, respectively.

6.1. Preemption Analysis for EDF and EDF-CF

Under EDF, we observe the following relationship between jobs to be preempted and to preempt.

Observation 2. Under *EDF*, job $J_{i,q}$ can preempt job $J_{k,p}$ only if the release time of $J_{i,q}$ is later than that of $J_{k,p}$ and the deadline of $J_{i,q}$ is before that of $J_{k,p}$.

By Observation 2, if $D_k \leq D_i$, any job of τ_i cannot preempt a job of τ_k. On the other hand, if $D_k > D_i$, the number of times a job of τ_k is preempted by jobs of τ_i is maximized when a job of τ_i is released just after a job of τ_k is released, as shown in Figure 3(a). Then, the number is calculated by $\lceil \frac{D_k - D_i}{T_i} \rceil$. To summarize, we can express an upper



(a) In the case of EDF. (b) In the case of EDF-CF or EDZL-CF (assuming $\phi_i < C_i$ and $\phi_k > 0$).

Fig. 3. Situations when the number of times a job of τ_k is preempted by jobs of τ_i is maximized under EDF and under EDF-CF or EDZL-CF.

bound on the number of times a job of τ_k is preempted by jobs of τ_i (denoted by $N^{\mathsf{EDF}}_{k\leftarrow i}$) as follows [Ju et al. 2007].

$$N^{\mathsf{EDF}}_{k\leftarrow i} \triangleq \begin{cases} \left\lceil \frac{D_k - D_i}{T_i} \right\rceil, & \text{if } D_k > D_i, \\ 0, & \text{if } D_k \le D_i. \end{cases} \tag{13}$$

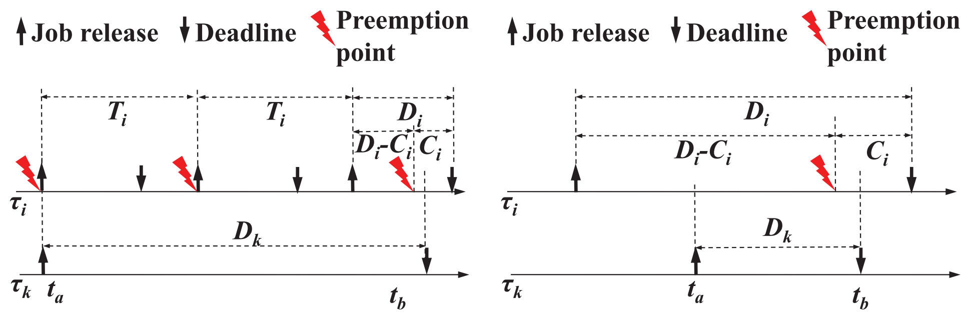
We note that under EDF, a job can be preempted at t only if some job is released at t. Therefore, we can calculate an upper bound on the number of times a job of τ_k is preempted under EDF by summing $N^{\mathsf{EDF}}_{k\leftarrow i}$ for all $\tau_i \in \mathcal{T} - \{\tau_k\}$ as follows.

$$\sum_{\tau_i \in \mathcal{T} - \{\tau_k\}} N^{\mathsf{EDF}}_{k\leftarrow i}. \tag{14}$$

Under EDF-CF, we may consider three cases for calculating an upper bound on the number of times a job of τ_k is preempted by jobs of τ_i (denoted by $N^{\mathsf{EDF\text{-}CF}}_{k\leftarrow i}$). The first case is $\phi_i \ge C_i$, where any job of τ_i is executed only in contention-free slots. Then, no preemption occurs by any job of τ_i. In the second case of $\phi_k = 0$ (and $\phi_i < C_i$), a job of τ_k cannot be in the lower-priority queue, and then the upper bound is the same as $N^{\mathsf{EDF}}_{k\leftarrow i}$. In the rest of the cases, a job of τ_k can be in the lower-priority queue. Once the job is in the lower-priority queue, a job of τ_i can preempt the job of τ_k even if the deadline of the job of τ_i is later than that of the job of τ_k. In this case, $N^{\mathsf{EDF\text{-}CF}}_{k\leftarrow i}$ is maximized when a job of τ_i is released just after a job of τ_k is released, as shown in Figure 3(b), and it can be calculated by $\lceil \frac{D_k}{T_i} \rceil$. In summary, $N^{\mathsf{EDF\text{-}CF}}_{k\leftarrow i}$ is calculated by

$$N^{\mathsf{EDF\text{-}CF}}_{k\leftarrow i} \triangleq \begin{cases} 0, & \text{if } \phi_i \ge C_i, \\ N^{\mathsf{EDF}}_{k\leftarrow i}, & \text{else if } \phi_k = 0, \\ \left\lceil \dfrac{D_k}{T_i} \right\rceil, & \text{otherwise.} \end{cases} \tag{15}$$

While EDF allows a job to be preempted only when a job is released, EDF-CF may incur one more self-preemption when each job moves from the higher-priority queue to the lower-priority one. Note that this self-preemption of a job of τ_k occurs only when $0 < \phi_k < C_k$. Otherwise, a job of τ_k is always in either the lower-priority queue or the higher-priority queue. Using this property and $N^{\mathsf{EDF\text{-}CF}}_{k\leftarrow i}$, the following lemma presents an upper bound on the number of times a job is preempted under EDF-CF.



(a) In the case of $D_k > D_i - C_i$. (b) In the case of $D_k \leq D_i - C_i$.

Fig. 4. Situations when the number of times a job of τ_k is preempted by jobs of τ_i is maximized under EDZL.

LEMMA 6.1. *Under EDF-CF, the number of times a job of τ_k is preempted is upper bounded by*

$$\begin{cases} 1 + \sum_{\tau_i \in \mathcal{T} - \{\tau_k\}} N_{k \leftarrow i}^{\textsf{EDF-CF}}, & \textit{if } 0 < \phi_k < C_k, \\ \sum_{\tau_i \in \mathcal{T} - \{\tau_k\}} N_{k \leftarrow i}^{\textsf{EDF-CF}}, & \textit{if } \phi_k = 0 \textit{ or } \phi_k \geq C_k. \end{cases} \tag{16}$$

PROOF. The lemma holds by the calculation of $N_{k \leftarrow i}^{\textsf{EDF-CF}}$ and the preceding explanation of self-preemption. □

6.2. Preemption Analysis for EDZL and EDZL-CF

We can calculate an upper bound on the number of times a job of τ_k is preempted by jobs of τ_i under EDZL (denoted by $N_{k \leftarrow i}^{\textsf{EDZL}}$), similar to that under EDF. While EDF allows $J_{i,q}$ to preempt $J_{k,p}$ only if the deadline of $J_{i,q}$ is earlier than that of $J_{k,p}$, EDZL also allows such a preemption if $J_{i,q}$ encounters a zero-laxity state. Therefore, if $D_k > D_i - C_i$, the number of times a job of τ_k is preempted by jobs of τ_i is maximized when a job of τ_i is released just after a job of τ_k is released, as shown in Figure 4(a). In the figure, the last job of τ_i may preempt a job of τ_k if it is in a zero-laxity state, and a job of τ_i can encounter a zero-laxity state at least $D_i - C_i$ time units after its release time. Therefore, $N_{k \leftarrow i}^{\textsf{EDZL}}$ is calculated by $\lceil \frac{D_k - D_i + C_i}{T_i} \rceil$ if $D_k > D_i - C_i$. In the case of $D_k \leq D_i - C_i$, $N_{k \leftarrow i}^{\textsf{EDZL}}$ is equal to one because we can make a release pattern such that a job of τ_i preempts a job of τ_k by encountering a zero-laxity state, as shown in Figure 4(b). Therefore, we summarize $N_{k \leftarrow i}^{\textsf{EDZL}}$ as follows.

$$N_{k \leftarrow i}^{\textsf{EDZL}} \triangleq \begin{cases} \left\lceil \dfrac{D_k - D_i + C_i}{T_i} \right\rceil, & \text{if } D_k > D_i - C_i, \\ 1, & \text{if } D_k \leq D_i - C_i. \end{cases} \tag{17}$$

Then, similar to EDF, we can calculate an upper bound on the number of times a job of τ_k is preempted under EDZL as follows.

$$\sum_{\tau_i \in \mathcal{T} - \{\tau_k\}} N_{k \leftarrow i}^{\textsf{EDZL}}. \tag{18}$$

We also calculate an upper bound on the number of times a job of τ_k is preempted by jobs of τ_i under EDZL-CF (denoted by $N_{k \leftarrow i}^{\textsf{EDZL-CF}}$), similar to EDF-CF. That is, in the case of $\phi_i \geq C_i$, there is no preemption; in the case of $\phi_k = 0$, the upper bound is the same as $N_{k\leftarrow i}^{\text{EDZL}}$; and otherwise, the upper bound is $\lceil \frac{D_k}{T_i} \rceil$. In summary, $N_{k\leftarrow i}^{\text{EDZL-CF}}$ is calculated as follows.

$$N_{k\leftarrow i}^{\text{EDZL-CF}} \triangleq \begin{cases} 0, & \text{if } \phi_i \geq C_i, \\ N_{k\leftarrow i}^{\text{EDZL}}, & \text{else if } \phi_k = 0, \\ \left\lceil \dfrac{D_k}{T_i} \right\rceil, & \text{otherwise.} \end{cases} \tag{19}$$

Similar to EDF-CF, EDZL-CF may incur one more self-preemption without any job release, when each job moves from the higher-priority queue to the lower-priority one. Therefore, similar to EDF-CF, we calculate an upper bound on the number of times a job is preempted under EDZL-CF in the following lemma.

LEMMA 6.2. *Under EDZL-CF, the number of times a job of τ_k is preempted is upper bounded by*

$$\begin{cases} 1 + \sum_{\tau_i \in \mathcal{T} - \{\tau_k\}} N_{k\leftarrow i}^{\text{EDZL-CF}}, & \text{if } 0 < \phi_k < C_k, \\ \sum_{\tau_i \in \mathcal{T} - \{\tau_k\}} N_{k\leftarrow i}^{\text{EDZL-CF}}, & \text{if } \phi_k = 0 \text{ or } \phi_k \geq C_k. \end{cases} \tag{20}$$

PROOF. The proof is similar to that of Lemma 6.1. □

7. PERFORMANCE EVALUATION

This section presents simulation results to evaluate the performance of the CF policy. As examples, we compare the performance of EDF-CF and EDZL-CF with that of their corresponding base algorithms, EDF and EDZL, in terms of schedulability and the number of preemptions.

7.1. Task Set Generation

We generate task sets based on a technique proposed earlier [Baker 2005], which has also been used in many previous studies (e.g., [Bertogna et al. 2009; Andersson et al. 2008]). We have three input parameters: (a) the number of processors m (1, 2, 4, 8, 16, 32, 48 or 64), (b) the type of task sets (constrained or implicit deadline), and (c) individual task utilization (C_i/T_i) distribution (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). For each task, T_i is uniformly chosen in $[1, T_{max} = 1000]$, C_i is chosen based on the bimodal or exponential parameter, and D_i is uniformly chosen in $[C_i, T_i]$ for constrained deadline task sets or D_i is equal to T_i for implicit deadline task sets.

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 $m + 1$ tasks.
(2) In order to exclude unschedulable sets, we check whether the generated task set can pass a necessary feasibility condition [Baker and Cirinei 2006].
(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 set serves as a

[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 the exponential distribution whose probability density function is $\lambda \cdot \exp(-\lambda \cdot x)$.

basis for the next new set; we create a new set by adding a new task into the old set and return to Step 2.

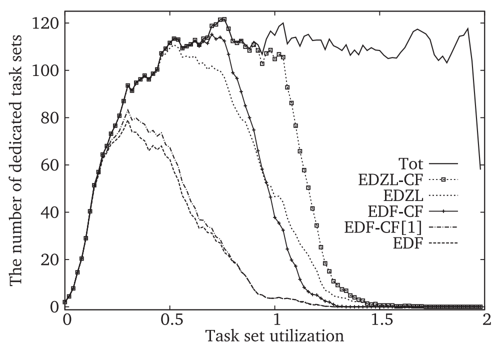
For any given m and given task set type, 10,000 task sets are created for each task utilization model, thus resulting in 100,000 task sets in total.

7.2. Comparison of Schedulability

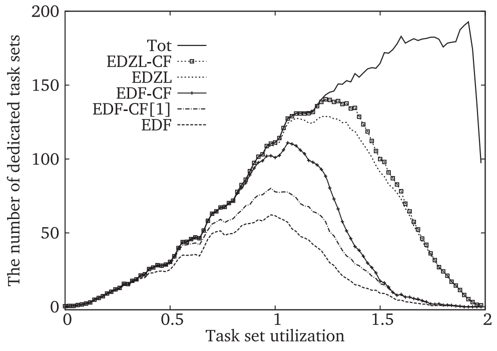
In this section, we evaluate the performance of five schedulability tests: (i) the EDF test in Lemma 5.2, (ii) the EDF-CF test in Lee et al. [2011a] (the best result with deadline reduction), (iii) our proposed EDF-CF test in Theorem 5.3, (iv) the EDZL test in Lemma 5.4, and (v) our EDZL-CF test in Theorem 5.5. These tests are, respectively, annotated as "EDF", "EDF-CF[1]", "EDF-CF", "EDZL", and "EDZL-CF". Note that in this article, we do not apply the deadline reduction technique presented in our preliminary paper [Lee et al. 2011a] to both EDF-CF and EDZL-CF. This is because our improved lower bound on the number of contention-free slots (i.e., $\Phi''(D_k)$) significantly improves the schedulability of EDF-CF compared to the lower bound presented in Lee et al. [2011a] (i.e., $\Phi(D_k)$ in this article), and the deadline reduction technique achieves only marginal improvement when it is incorporated into EDF-CF or EDZL-CF. However, for comparison, we include EDF-CF[1], which employs $\Phi(D_k)$ and the best deadline reduction heuristic presented in Lee et al. [2011a].

Figure 5 shows schedulability test results for implicit deadline task sets over varying task utilization models and different numbers of processors. Among the ten task utilization models, we choose to show exponential distribution with 0.1, exponential distribution with 0.9, and bimodal distribution with 0.9, since they correspond to the cases where the average task utilization ($\overline{C_i/T_i}$) is the smallest, medium, and the largest, respectively (i.e., the average number of tasks ($\overline{n}$) is the largest, medium, and the smallest). Among the different m, we choose to show for $m = 2$ and 8, since simulation results with different values of m have similar behaviors. Each figure comprises six lines, each plot showing the number of task sets proven schedulable by each test, with task set utilization ($U_{sys} \triangleq \sum_{\tau_i \in \mathcal{T}} C_i/T_i$) in $[U_{sys} - 0.01 \cdot m, U_{sys} + 0.01 \cdot m)$. Here, "Tot" means the total number of task sets with each task set utilization.

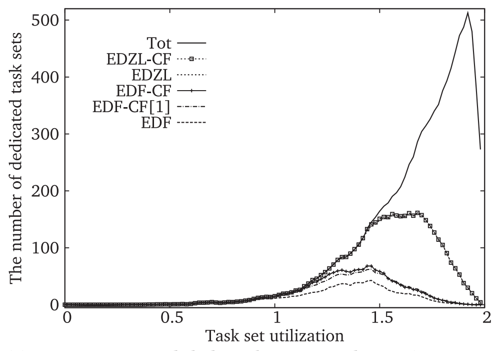
As shown in Figure 5, the CF policy improves schedulability of implicit deadline task sets, but the degree of improvement varies with individual task utilization. That is, while EDF-CF and EDZL-CF significantly improve schedulability of task sets with exponential distribution with 0.1 (see Figures 5(a) and 5(d)), compared to EDF and EDF-CF[1], and EDZL, respectively, such improvement is fair for task sets with exponential distribution with 0.9 (see Figures 5(b) and 5(e)) and marginal for task sets with bimodal distribution with 0.9 (see Figures 5(c) and 5(f)). This is related to the property of schedulability tests EDF and EDZL. As task set utilization gets larger, the number of contention-free slots of a job gets smaller, since the number of active jobs in each time slot is likely to become larger. Therefore, we can obtain more benefit from the CF policy in terms of schedulability analysis when task set utilization is lower. In turn, since EDF and EDZL rely on an upper bound on interference from individual tasks (i.e., Eq. (7)), more tasks in a task set pronounce the pessimism in the interference bound, resulting in lower schedulability. Further credibility to this argument is lent by the observation that if we focus on task set utilization of 1.0 in Figures 5(a), (b), and (c), we can check that no task set, about half of task sets, and almost all task sets are schedulable by EDF, respectively. Therefore, once the CF policy is incorporated, it can find more contention-free slots with lower task set utilization and then results in more improvement for task sets with exponential distribution with 0.1, in which EDF and EDZL have poor performance with low task set utilization. Such an improvement for low task set utilization is so significant that EDF-CF is comparable to EDZL in



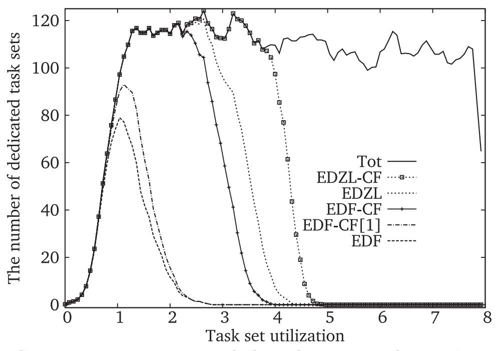
(a) $m = 2$. Exponential distribution with 0.1 ($\overline{n} = 11.6$, $\overline{C_i/T_i} = 0.09$).



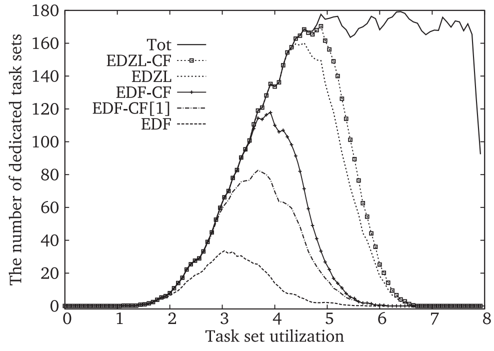
(b) $m = 2$. Exponential distribution with 0.9 ($\overline{n} = 4.3$, $\overline{C_i/T_i} = 0.32$).



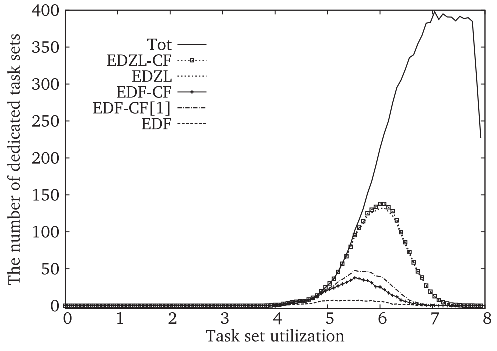
(c) $m = 2$. Bimodal distribution with 0.9 ($\overline{n} = 3.1$, $\overline{C_i/T_i} = 0.55$).



(d) $m = 8$. Exponential distribution with 0.1 ($\overline{n} = 43.1$, $\overline{C_i/T_i} = 0.10$).



(e) $m = 8$. Exponential distribution with 0.9 ($\overline{n} = 14.5$, $\overline{C_i/T_i} = 0.39$).



(f) $m = 8$. Bimodal distribution with 0.9 ($\overline{n} = 10.1$, $\overline{C_i/T_i} = 0.68$).

Fig. 5. Schedulability tests for implicit deadline task sets.

Figures 5(a) and 5(d), while there is a huge schedulability gap between EDF-CF and EDZL in Figures 5(c) and 5(f).

Now we present the numerical value for overall improvement by the CF policy for implicit deadline task systems. Table I shows the number of schedulable task sets by the five schedulability tests. Here 100,000 task sets are tested for each m and the type of task sets (which are generated according to Section 7.1). When $m = 2$ for implicit deadline task sets, while EDF-CF[1] deems additional 31.0% task sets schedulable compared

Table I. Number of Schedulable Task Sets among 100,000 Task Sets

Implicit deadline task sets					
	EDF	EDF-CF[1]	EDF-CF	EDZL	EDZL-CF
$m = 2$	20,999	27,516	36,929	55,882	59,396
$m = 8$	6,261	16,438	23,637	40,182	44,839

Constrained deadline task sets					
	EDF	EDF-CF[1]	EDF-CF	EDZL	EDZL-CF
$m = 2$	9,705	20,767	27,736	48,655	55,355
$m = 8$	2,177	12,272	16,801	29,572	36,673

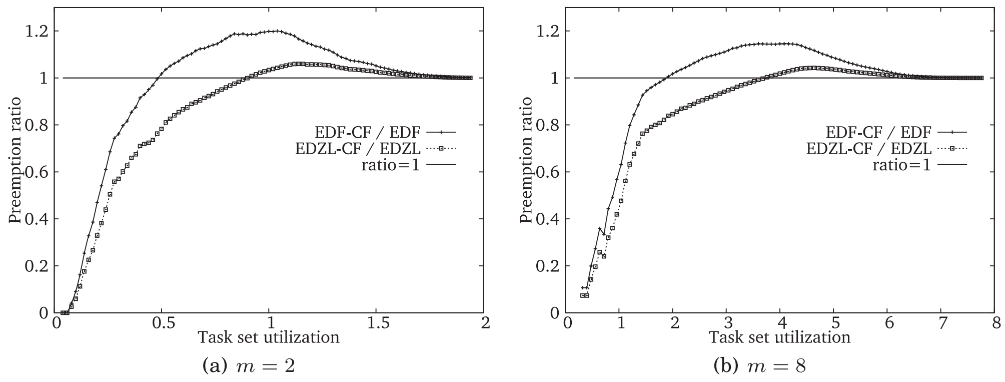


Fig. 6. Preemption ratio between EDF-CF and EDF, and EDZL-CF and EDZL for implicit deadline task systems.

to EDF, EDF-CF deems additional 75.9% task sets schedulable. When it comes to EDZL-CF, its improvement is 6.3% over EDZL. Note that the degree of the improvement by EDF-CF is larger than that by EDZL-CF, because EDZL produces better schedules than EDF in multiprocessor scheduling so that EDF has more room to improve. As the number of processors gets larger, the improvement by the CF policy is more pronounced. For example, when $m = 8$ for implicit deadline task sets, EDF-CF[1] and EDF-CF, respectively, deem additional 162.5% and 277.5% task sets, compared to EDF, and EDZL-CF deems additional 11.6% task sets, compared to EDZL.

For constrained deadline task sets, the behavior of schedulability improvement is similar to that for implicit deadline task sets, and therefore we do not show the corresponding figures to Figure 5. However, as shown in Table I, we check that the CF policy successfully improves schedulability of base algorithms, even for constrained deadline task sets for $m = 2$ and $m = 8$.

7.3. Comparison of the Number of Preemptions

In this section, we evaluate the number of times each job is preempted under EDF-CF and EDZL-CF compared to EDF and EDZL.

Figure 6 shows preemption results for $m = 2$ and $m = 8$ for implicit deadline task sets. Each figure consists of three lines: (i) the ratio between an upper bound on the number of times each job is preempted under EDF-CF (i.e., Eq. (16)) and that under EDF (i.e., Eq. (14)), (ii) the ratio between an upper bound on the number of times each job is preempted under EDZL-CF (i.e., Eq. (20)) and that under EDZL (i.e., Eq. (18)), and (iii) a base line in which the ratio is uniformly one. Each plot for (i) and (ii) shows the average ratio of each task, with task set utilization ($U_{sys} \triangleq \sum_{\tau_i \in \mathcal{T}} C_i/T_i$) in $[U_{sys} - 0.01 \cdot m, U_{sys} + 0.01 \cdot m)$.

Table II. Average Actual Number of Preemptions Incurred by Each Task Set During 100,000 Time Units

Implicit deadline task sets						
	EDF	EDF-CF	EDF-CF/EDF	EDZL	EDZL-CF	EDZL-CF/EDZL
$m = 2$	1,068.3	1,071.1	100.26%	1,079.7	1,082.4	100.25%
$m = 8$	2,319.8	2,321.1	100.06%	2,324.8	2,326.1	100.06%

Constrained deadline task sets						
	EDF	EDF-CF	EDF-CF/EDF	EDZL	EDZL-CF	EDZL-CF/EDZL
$m = 2$	752.9	757.6	100.62%	771.1	774.5	100.44%
$m = 8$	1,554.7	1,556.3	100.10%	1,576.4	1,577.8	100.09%

In Figure 6, we observe three behaviors of the two ratios as task set utilization becomes larger: (i) the two ratios are very low at the beginning; (ii) the two ratios become larger up to some point; and (iii) after the point, the two ratios become smaller and converge to one. When task set utilization is low, many tasks may have the maximum number of contention-free slots (i.e., $\phi_i \geq C_i$), and then any job invoked by such tasks cannot incur any preemption, as shown in Eqs. (15) and (19). As task set utilization gets larger, less tasks satisfy $\phi_i \geq C_i$, which means more tasks belong to the condition of "otherwise" in Eqs. (15) and (19). Since it is trivial, $\lceil \frac{D_k}{T_i} \rceil$ is always equal to or larger than both $N_{k \leftarrow i}^{\mathsf{EDF}}$ and $N_{k \leftarrow i}^{\mathsf{EDZL}}$, and the two ratios will be larger than one in some task set utilization. However, if total set utilization converges to m, it is difficult for tasks to get any contention-free slots, and then ϕ_k for each task τ_k also converges to zero. Then, it holds that $N_{k \leftarrow i}^{\mathsf{EDF\text{-}CF}} = N_{k \leftarrow i}^{\mathsf{EDF}}$ and $N_{k \leftarrow i}^{\mathsf{EDZL\text{-}CF}} = N_{k \leftarrow i}^{\mathsf{EDZL}}$, as shown in Eqs. (15) and (19), and thus the two ratios equal to one. Note that the three behaviors also hold for constrained deadline task sets.

In summary, an upper bound on the number of times a job is preempted under EDF-CF, and EDZL-CF is, respectively, compared with that by EDF and EDZL as follows. When task set utilization is low, the CF-based algorithms incur less preemptions than the base algorithms; when task set utilization is high, the CF-based algorithms and the base algorithms incur a similar number of preemptions; and even when task set utilization is medium, the CF-based algorithms incur up to 20% (EDF-CF) and 6% (EDZL-CF) additional preemptions compared to the base algorithms, regardless of m and the type of task sets. Note that the ratio between EDF-CF and EDF is larger than the ratio between EDZL-CF and EDZL, because the gap between $N_{k \leftarrow i}^{\mathsf{EDF}}$ and $\lceil \frac{D_k}{T_i} \rceil$ is larger than that between $N_{k \leftarrow i}^{\mathsf{EDZL}}$ and $\lceil \frac{D_k}{T_i} \rceil$.

Such upper bounds on the number of preemptions, although pessimistic, are useful when the stringent guarantees on the number of preemptions are required. However, we also need to analyze how many actual additional preemptions the CF policy incurs, which demonstrates average preemption behaviors. To do this, we perform actual simulation of EDF, EDF-CF, EDZL, and EDZL-CF over all task sets generated in Section 7.1 and count the total number of preemptions incurred by each task set during the first 100,000 time units for tractability.

Table II shows the average actual number of preemptions incurred by each task set during 100,000 time units. As shown in the table, if we compare EDF with EDF-CF and EDZL with EDZL-CF, the actual additional preemptions incurred by the CF policy is not significant regardless of m and the task system, that is, less than 1% additional preemptions in any case. This is because the CF policy itself is not frequently triggered as follows. If a task set utilization is low, the number of CF slots of each job is large, resulting in many jobs in the lower-priority queue Q_L; however, a low task set utilization yields a small number of active jobs, and therefore, the jobs in

Q_L do not frequently delay their executions owing to the CF policy. On the other hand, if a task set utilization is high, few jobs are in Q_L, and therefore few jobs delay their executions due to the CF policy. Therefore, the CF policy does not significantly change the original schedules by base algorithms; instead, the CF policy improves predictability by reducing a considerable pessimism in schedulability analyses of base algorithms, as presented in Section 7.2.

8. DISCUSSION

In this article, we have defined a new notion of contention-free slots, analyzed the number of contention-free slots in a given interval, and finally developed the CF policy that takes advantage of contention-free slots. To demonstrate how the CF policy improves schedulability of existing scheduling algorithms, we have chosen simple algorithms and their analyses as examples, that is, global EDF and EDZL, and their deadline-based analyses. One may wonder that the applicability of the CF policy and its analysis is restricted only to the presented algorithms and analyses. To clarify this, this section discusses the applicability of the CF policy and its analysis towards other existing schedulability analyses and scheduling algorithms. Then, this section also discusses runtime overhead of the CF policy.

8.1. Applicability of the CF Analysis Technique for Schedulability Analyses

To guarantee schedulability of individual scheduling algorithms, many schedulability analyses have investigated per-task behaviors, that is, per-task schedulability tests check for individual tasks whether any job invoked by a task can miss its deadline, subject to no deadline miss for any other job, and the tests operate as follows. First, they calculate the amount of higher-priority job interference in a given interval, which can block the job of interest. Then, they deem a job of a task cannot trigger the first deadline miss if the job of interest can finish its execution within the deadline in the presence of higher-priority job interference.

We have introduced how to calculate the number of contention-free slots of each job regardless of underlying scheduling algorithms (i.e., ϕ_i for τ_i), and, as demonstrated in Theorems 5.3 and 5.5, the CF analysis technique presented in this article can reduce the amount of interference (i.e., $\max(0, C_i - \phi_i)$ instead of C_i). Therefore, the CF analysis technique can be immediately incorporated into most (if not all) per-task schedulability tests (e.g., [Bertogna et al. 2005, 2009; Baker et al. 2008; Lee et al. 2010; Baruah 2007; Guan et al. 2009]) for EDF, fixed-priority, EDZL, and LLF scheduling algorithms. As an example, we now demonstrate how to incorporate the CF analysis technique into the limited carry-in[3] technique for EDF [Baruah 2007].

The schedulability analysis technique presented in Section 5 calculates how long jobs of other tasks can interfere with a job of τ_k (whose interval is $[t_a, t_a + D_k)$) assuming that any task can have its carry-in job in $[t_a, t_a + D_k)$. To derive a tighter upper bound on the carry-in job interference, the limited carry-in technique investigates a set of extended intervals $[t_a - A_k, t_a + D_k)$ with arbitrary $A_k \geq 0$, provided that $[t_a - A_k, t_a)$ is the longest busy interval with at least one idle core in $[t_a - A_k - 1, t_a - A_k)$ but no idle core in $[t_a - A_k, t_a)$. By definition, the number of carry-in jobs in $[t_a - A_k, t_a + D_k)$ is upper bounded by $m - 1$. Then, we can calculate an upper bound on the maximum execution of jobs of τ_i in $[t_a - A_k, t_a + D_k)$ depending on whether τ_i has a carry-in job or

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

Table III. Number of Schedulable Task Sets Among 100,000 Task Sets

Implicit deadline task sets					Constrained deadline task sets				
	EDF	EDF*	EDF-CF	EDF-CF*		EDF	EDF*	EDF-CF	EDF-CF*
$m = 2$	20,999	55,776	36,929	57,929	$m = 2$	9,705	34,027	27,736	39,423
$m = 8$	6,261	16,677	23,637	27,762	$m = 8$	2,177	4,525	16,801	17,427

not (denoted by $I^{\mathsf{CI}}_{k\leftarrow i}$ and $I^{\mathsf{NC}}_{k\leftarrow i}$) as follows.

$$I^{\mathsf{CI}}_{k\leftarrow i} = \left\lfloor \frac{D_k + A_k}{T_i} \right\rfloor C_i + \min\left(C_i, D_k + A_k - \left\lfloor \frac{D_k + A_k}{T_i} \right\rfloor T_i\right), \tag{21}$$

$$I^{\mathsf{NC}}_{k\leftarrow i} = \max\left(0, \left(\left\lfloor \frac{D_k + A_k - D_i}{T_i} \right\rfloor + 1\right) \cdot C_i\right). \tag{22}$$

Then, for a given A_k, if the total amount of execution of tasks τ_i ($\neq \tau_k$) with any combination of at most $m-1$ carry-in jobs is not larger than $m \cdot (A_k + D_k - C_k)$, the job of interest of τ_k can successfully finish C_k amount of execution in $[t_a, t_a + D_k)$. By repeating this process for all $0 \leq A_k \leq \mathsf{MAX}_k$, we can guarantee τ_k's schedulability under EDF. See Baruah [2007] for more details, including how to calculate MAX_k.

Now, we briefly discuss how to adapt $I^{\mathsf{CI}}_{k\leftarrow i}$ and $I^{\mathsf{NC}}_{k\leftarrow i}$ for EDF-CF. Depending on whether the job of interest of τ_k is ready to execute or not, we divide $[t_a - A_k, t_a + D_k)$ into two disjoint subintervals. In $[t_a, t_a + D_k)$, no job in contention-free slots interferes with the job of interest of τ_k, and therefore, we need to know how many executions of a job of τ_i are performed in contention-free slots, that is, ϕ_i. On the other hand, $[t_a - A_k, t_a)$ is a busy interval. Since the amount of executions in Eqs. (21) and (22) is not clearly divided into the two intervals $[t_a - A_k, t_a)$ and $[t_a, t_a + D_k)$, we use the following interesting property: by definition, a non-busy slot (in which there are at most $m-1$ active jobs) is a contention-free slot. Therefore, once we calculate how many executions of a job of τ_i are performed in non-busy slots (denoted by ϕ_i'), such executions can neither contribute to make each slot in $[t_a - A_k, t_a)$ a busy slot, nor interfere with the job of interest of τ_k. Thus, for EDF-CF, we can use Eqs. (21) and (22) for replacing C_i with $C_i'' = \max(0, C_i - \phi_i')$. Note that by definition, ϕ_i' can be calculated by applying $m - 1$ instead of m to the calculation of ϕ_i in Lemma 3.5.

Table III shows the number of schedulable task sets among 100,000 task sets generated in Section 7.1 according to two different schedulability tests, EDF* and EDF-CF*; they, respectively, represent the schedulability tests of EDF and EDF-CF when the limited carry-in technique is combined using Eqs. (21) and (22), and using those with C_i''. As shown in the table, EDF-CF* covers a number of additional schedulable task sets in comparison with EDF* and EDF-CF. Specifically, EDF-CF* is shown to improve schedulability by 3.9–15.9% when $m = 2$, and by 66.5–285.1% when $m = 8$, compared to EDF*. This means, the CF policy significantly improves schedulability of EDF even with the limited carry-in technique.

8.2. Applicability of the CF Policy for Scheduling Algorithms

Although we have demonstrated the applicability of the CF policy for global EDF and EDZL, the CF policy can be incorporated into most (if not all) global, partitioned, and semi-partitioned scheduling algorithms[4] as long as the scheduling algorithms are work-conserving and preemptive. For example, there is a class of optimal global scheduling

[4]A global algorithm allows each task to migrate from one processor to another, while a partitioned algorithm restricts each task to run only on a designated processor. A semi-partitioned algorithm applies either global or partitioned approach to each task.

algorithms for implicit-deadline task systems, for example, PFair [Baruah et al. 1996]. However, those algorithms are no longer optimal for constrained-deadline task systems. The CF policy can be combined with those algorithms to improve schedulability by covering additional schedulable constrained deadline task sets, which are not found schedulable by those algorithms.

As of now, the CF policy and its analysis described in Algorithm 1 and Lemma 3.5 have been derived for global scheduling algorithms, but we can easily adapt them to partitioned algorithms by dividing a multiprocessor system into a set of uniprocessor subsystems. When it comes to semi-partitioned scheduling algorithms, it may not be straightforward to calculate the number of contention-free slots due to some migratory tasks; hence, here we introduce a brief idea. If we focus on a popular semi-partitioned algorithm, such as EDF-WM [Kato et al. 2009], migratory tasks have local deadline and execution times in each processor. Using the local deadline and execution time, we can calculate an upper bound on the amount of execution of a migratory task in an interval of given length on a given processor, and therefore, we can calculate the number of CF slots in an interval of given length on each processor. Once we know the number of CF slots, we can apply the CF policy similar to Algorithm 1.

8.3. Runtime Overhead of the CF Policy

One potential drawback of the CF policy is that the system should perform additional computation for each time slot; as shown in Algorithm 1, each task keeps track of the remaining execution ($C_i(t)$) and remaining contention-free slots ($\phi_i(t)$) for each time slot. While all scheduling algorithms, including EDF, EDZL, and the CF policy itself, require $O(1)$ computation for basic operations whenever each job is released or finished, the CF policy requires additional $O(m)$ and $O(n)$ computations every time slot for $C_i(t)$ and $\phi_i(t)$ updates, respectively. Note that the $O(m)$ computation is also needed for some scheduling algorithms, such as EDZL. Then, assuming $n > m$ (otherwise, the task set is trivially schedulable by any work-conserving algorithm), the total time complexity that EDF-CF and EDZL-CF incur every time slot is $O(1/f) + O(n)$, where f is average job release/finish frequency, while EDF and EDZL incur $O(1/f)$ and $O(1/f) + O(m)$, respectively.

To show actual runtime overhead, we measure the average time for the CF policy to incur during each time slot, using our simulation framework described in Section 7. Under Intel Xeon CPU E31230 with a single thread, the additional time for the CF policy to incur during each time slot is less than $0.01\mu s$ with any simulation environment (i.e., with EDF and EDZL, and different m). Even if the CPU employed in embedded systems is generally much slower than the one we used, the runtime overhead is still negligible, compared to the time quantum used in embedded systems, for example, $2ms$ in ARINC-653 [Airlines Electronic Engineering Committee 2003]. This means that if an embedded system employs a 200-times-slower CPU, the ratio of the runtime overhead to the time quantum is $2\mu s/2ms = 1/1000$, which is marginal.

To explore a trade-off between runtime overhead and schedulability improvement, let us discuss a variant of the CF policy. The update frequency X can be defined as a multiple of a time slot, and the variant of the CF policy, called CF$_X$, performs the tracking of the remaining execution and remaining contention-free slots for every X time slots instead of every single time slot; during $X - 1$ time slots out of X time slots, the tracking is not performed, and then the CF$_X$ policy may hold some jobs in Q_H, which are supposed to be transferred to Q_L. This delayed job-transfer results in decreasing the number of contention-free slots, and then also decreasing schedulability improvement from contention-free slots. On the other hand, increasing X can reduce runtime overhead. Note that the CF$_X$ policy is a generalization of the CF policy. If $X = 1$, the CF$_X$ policy is equivalent to the CF policy; on the other hand, if $X = \infty$, a base algorithm with the CF_X policy is the same as the base algorithm itself (no contention-free slots are utilized).

We can interpret the time complexity of the CF and CF_X policies as follows. In the case that n/m (the ratio of the number of cores, to the number of tasks) is not large, the order of time complexity the CF policy incurs is marginal; for example, the order of time complexity EDZL-CF incurs is similar to that at its base algorithm EDZL ($O(n) \approx O(m)$). On the other hand, in the case of large n/m, the overhead of the CF policy is not negligible, so we may consider applying the CF_X policy. Then, if we set X to n/m, the order of time complexity EDZL incurs is exactly the same as that of EDZL-CF_X, that is, $O(n) \cdot m/n = O(m)$.

The development of the CF_X policy raises many issues, including how to calculate new lower bounds of contention-free slots under the CF_X policy and how to find a suitable X for a given system. Those issues are beyond the scope of this article, while this article focuses on the original CF policy. However, we believe that the CF_X policy with a proper X effectively explores a trade-off between time complexity and schedulability improvement.

9. RELATED WORK

For real-time multiprocessor systems, many scheduling algorithms have been developed for better schedulability (see [Davis and Burns 2011b] for a comprehensive survey). Priority-based global scheduling algorithms can be broadly classified into three categories: (i) task-level fixed-priority (TFP) algorithms that assign a single static priority to all the jobs belonging to a single task, (ii) job-level fixed-priority (JFP) algorithms that assign a static priority to an individual job independently, and (iii) job-level dynamic-priority (JDP) algorithms under which a priority of job can dynamically change over time. Typical examples of TFP, JFP, and JDP algorithms are DM (deadline monotonic) [Leung and Whitehead 1982], EDF (earliest deadline first) [Liu and Layland 1973], and LLF (least laxity first) [Leung 1989], which are optimal preemptive scheduling algorithms over uniprocessors in their own categories, respectively.

Unlike the uniprocessor case, it has been reported that many existing TFP and JFP algorithms, including DM and EDF, are not quite effective on multiprocessors [Cho et al. 2002; Davis and Burns 2011b]. This leads to investigating variation in prioritization methods. For example, EDF-US [Srinivasan and Baruah 2002] and fpEDF [Baruah 2004] algorithms have been introduced as a hybrid of TFP and JFP. Both algorithms assign the highest priorities to some designated tasks (i.e., tasks of utilization greater than some threshold) and schedule the other tasks according to EDF. This way, it helps to improve a lower bound on utilization of schedulable task sets, compared to their base scheduling algorithm, EDF. However, EDF-US and fpEDF do not dominate EDF.

There has been another thread of research on prioritization methods that can be orthogonally applicable to different base scheduling algorithms, holding a dominance relation over the base algorithms. A successful example is the zero-laxity (ZL) policy, which gives the highest priority to jobs in the zero-laxity state. Different from EDF-US and fpEDF, the ZL policy can be incorporated into any (work-conserving and preemptive) scheduling algorithm. Also, since the policy promotes the priority of jobs that would otherwise inevitably miss their deadlines, any scheduling algorithm extended with the ZL policy dominates the base scheduling algorithm [Lee et al. 2011b]. So far, it has been demonstrated that the policy improves TFP and JFP algorithms, for example, EDZL (earliest deadline first until zero laxity) [Lee 1994; Cho et al. 2002; Park et al. 2005; Baker et al. 2008], and FPZL (task-level fixed priority until zero laxity) [Davis and Burns 2011a]. Besides, it could potentially improve JDP scheduling algorithms.

While the zero-laxity policy improves schedulability of the existing scheduling algorithms by promoting the priority of some jobs that should be executed to avoid a deadline miss, the CF policy achieves the same goal in the opposite way—demoting the priority of jobs that never miss their deadlines. Like the ZL policy, the CF policy not only can be incorporated into any (work-conserving and preemptive) scheduling algorithm, but also holds dominance relationship. One more important property of the ZL and CF policies is that they can be incorporated together into the existing scheduling algorithms simultaneously, for example, EDF can be extended with ZL and then CF, resulting in EDZL-CF. Therefore, we can gain an improvement by both policies at the same time.

10. CONCLUSIONS

In this article, we have introduced a novel notion of contention-free slots in real-time multiprocessor scheduling, and developed the CF policy that takes advantage of contention-free slots. We have shown that the CF policy can be incorporated into any work-conserving preemptive algorithm, and this improves existing algorithms. To demonstrate the effectiveness of the CF policy, we have developed schedulability analysis of algorithms that employ the CF policy and showed that the CF policy significantly improves the schedulability of corresponding original algorithms. With this improvement, we have showed that the CF policy can be of practical use in that additional preemptions incurred by the CF policy itself are not significant.

As we discussed in Section 8, the CF policy can be applied to any work-conserving, preemptive scheduling algorithm and any per-task schedulability test. The remaining challenge is to develop schedulability tests when the CF policy is incorporated into the existing scheduling algorithms that do not have per-task schedulability tests, for example, global algorithms with high utilization bounds and many existing semi-partitioned algorithms, which is our future work.

REFERENCES

AUTOSAR. 2009. AUTOSAR release 4.0 specification. http://www.autosar.org.

Airlines Electronic Engineering Committee. 2003. ARINC specification 653-1. Aeronautical Radio, INC., Annapolis, MD.

Björn Andersson, Konstantinos Bletsas, and Sanjoy Baruah. 2008. Scheduling arbitrary-deadline sporadic task systems on multiprocessor. In *Proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. 197–206.

Theodore P. Baker. 2005. Comparison of empirical success rates of global vs. partitioned fixed-priority and EDF scheduling for hand real time. Tech. rep. TR-050601. Department of Computer Science, Florida State University, Tallahasee.

Theodore P. Baker and Michele Cirinei. 2006. A necessary and sometimes sufficient condition for the feasibility of sets of sporadic hard-deadline tasks. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*. 178–190.

Theodore P. Baker, Michele Cirinei, and Marko Bertogna. 2008. EDZL scheduling analysis. *Real-Time Syst.* 40, 264–289.

Sanjoy Baruah. 2004. Optimal utilization bounds for the fixed-priority scheduling of periodic task systems on identical multiprocessors. *IEEE Trans. Comput.* 53, 6, 781–784.

Sanjoy Baruah. 2007. Techniques for multiprocessor global schedulability analysis. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*. 119–128.

S. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. 1996. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica* 15, 6, 600–625.

Sanjoy Baruah, Aloysius Mok, and Louis Rosier. 1990. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*. 182–190.

Marko Bertogna and Michele Cirinei. 2007. Response-time analysis for globally scheduled symmetric multiprocessor platforms. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*. 149–160.

Marko Bertogna, Michele Cirinei, and Giuseppe Lipari. 2005. Improved schedulability analysis of EDF on multiprocessor platforms. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*. 209–218.

Marko Bertogna, Michele Cirinei, and Giuseppe Lipari. 2009. Schedulability analysis of global scheduling algorithms on multiprocessor platforms. *IEEE Trans. Parallel Distrib. Syst.* 20, 553–566.

Alan Burns and Sanjoy Baruah. 2008. Sustainability in real-time scheduling. *J. Comput. Sci. Eng.* 2, 1, 74–97.

S. Cho, S.-K. Lee, S. Ahn, and K.-J. Lin. 2002. Efficient real-time scheduling algorithms for multiprocessor systems. *IEICE Trans. Commun.* E85–B, 12, 2859–2867.

Robert I. Davis and Alan Burns. 2011a. FPZL schedulability analysis. In *Proceedings of the IEEE Real-Time Technology and Applications Symposium (RTAS)*. 245–256.

Robert I. Davis and Alan Burns. 2011b. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv.* 43, 35:1–35:44.

M. Dertouzos. 1974. Control robotics: The procedural control of physical processors. In *Proceedings of the IFIP Congress*. 807–813.

Nan Guan, Martin Sitgge, Wang Yi, and Ge Yu. 2009. New response time bounds for fixed priority multiprocessor scheduling. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*. 387–397.

Lei Ju, Samarjit Chakraborty, and Abhik Roychoudhury. 2007. Accounting for cache-related preemption delay in dynamic priority schedulability analysis. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 1623–1628.

Shinpei Kato, Nobuyoki Yamasaki, and Yutaka Ishikawa. 2009. Semi-partitioned scheduling of sporadic task systems on multiprocessors. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*. 249–258.

Jinkyu Lee, Arvind Easwaran, and Insik Shin. 2010. LLF schedulability analysis on multiprocessor platforms. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*. 25–36.

Jinkyu Lee, Arvind Easwaran, and Insik Shin. 2011a. Maximizing contention-free executions in multiprocessor scheduling. In *Proceedings of the IEEE Real-Time Technology and Applications Symposium (RTAS)*. 235–244.

Jinkyu Lee, Arvind Easwaran, Insik Shin, and Insup Lee. 2011b. Zero-laxity based real-time multiprocessor scheduling. *J. Syst. Soft.* 84, 12, 2324–2333.

Suk Kyoon Lee. 1994. On-line multiprocessor scheduling algorithms for real-time tasks. In *Proceedings of the IEEE Region 10's 9th Annual International Conference*. 607–611.

J. Y. T. Leung and J. Whitehead. 1982. On the complexity of fixed-priority scheduling of periodic real-time tasks. *Perform. Eval.* 2, 237–250.

J. Y.-T. Leung. 1989. A new algorithm for scheduling periodic, real-time tasks. *Algorithmica* 4, 209–219.

C. L. Liu and James Layland. 1973. Scheduling algorithms for multi-programming in a hard-real-time environment. *J. ACM* 20, 1, 46–61.

Aloysius Mok. 1983. Fundamental design problems of distributed systems for the hard-real-time environment. Ph.D. Dissertation, Massachusetts Institute of Technology.

Minkyu Park, Sangchul Han, Heeheon Kim, Seongje Cho, and Yookun Cho. 2005. Comparison of deadline-based scheduling algorithms for periodic real-time tasks on multiprocessor. *IEICE Trans. Info. Syst.* E88-D, 658–661.

Anand Srinivasan and Sanjoy Baruah. 2002. Deadline-based scheduling of periodic task systems on multiprocessors. *Info. Process. Lett.* 84, 2, 93–98.

C. H. (Kees) van Berkel. 2009. Multi-core for mobile phones. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 1260–1265.

Received July 2012; revised December 2012, March 2012; accepted July 2013