2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium

Maximizing Contention-Free Executions in Multiprocessor Scheduling

Jinkyu Lee*, Arvind Easwaran† and Insik Shin*‡

*Dept. of Computer Science, KAIST, South Korea
†Cister Research Unit, Polytechnic Institute of Porto, Portugal
jinkyu@cps.kaist.ac.kr; aen@isep.ipp.pt; insik.shin@cs.kaist.ac.kr

***Abstract*—It is widely assumed that scheduling real-time tasks becomes more difficult as their deadlines get shorter. With deadlines shorter, however, tasks potentially compete less with each other for processors, and this could produce more *contention-free* slots at which the number of competing tasks is smaller than or equal to the number of available processors. This paper presents a policy (called CF policy) that utilizes such contention-free slots effectively. 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 tasks is reduced when their executions are postponed to contention-free slots. Finally, using the properties of the CF policy, we derive a counter-intuitive claim that shortening of task deadlines can help improve schedulability of task systems. We present heuristics that effectively reduce task deadlines for better scheduability without performing any exhaustive search.**

I. Introduction

With the increasing popularity of multi-core architectures, real-time multiprocessor scheduling has been receiving a growing interest in the recent past. Some multiprocessor studies (e.g., [1], [2], [3]) have focused on adapting existing uniprocessor scheduling to multiprocessors, and some others have developed novel scheduling theories specific to multiprocessors (e.g., [4], [5], [6], [7], [8], [9], [10], [11]). While real-time scheduling on uniprocessor platforms has successfully matured over years so that Earliest Deadline First (EDF) [12] was developed as an optimal scheduler in this domain, the same cannot be said about scheduling theory for multiprocessors. In particular, though optimal schedulers are known for implicit deadline task systems (deadline equal to task period) [4], no such scheduler has yet been developed for constrained deadline task systems (deadline no larger than task period). This entails new scheduling concepts and methods that can utilize the characteristics of constrained deadline task systems.

It is intuitive to generally infer that meeting the deadline of a job gets harder as we make the deadline shorter. This implies that a constrained deadline task set is generally more difficult to schedule than the corresponding implicit deadline task set, with the same periods and execution times. The fact that no optimal scheduler can exist for certain constrained deadline systems [13], lends further credibility to this argument. However, a shorter deadline can sometimes be favorable to schedulability as we will now demonstrate. Consider a constrained deadline system consisting of tasks τ_1, τ_2 and τ_3, scheduled on a two processor platform as shown in Figure 1. In such a system, it is guaranteed that a task does not compete for processors between the deadline of any job of the task and the release time of the next job of the same task. For example, in Figure 1, it can be seen that task τ_1 does not compete for processors in the intervals $[t_1, t_2)$ and $[t_6, t_7)$. If we consider such time slots for all tasks in the example, we can see that there are at most two competing tasks in intervals $[t_1, t_2)$, $[t_3, t_4)$ and $[t_5, t_8)$. We call these time slots *contention-free*, because the number of competing tasks is at most the number of available processors. An interesting property of these slots is that all pending tasks are guaranteed to be scheduled in them, as long as any work-conserving algorithm is used. By work-conserving, we mean an algorithm that always schedules ready tasks if processors are available.

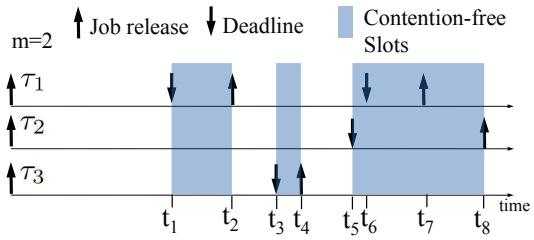


Figure 1. An example of contention-free slots

Using the notion of contention-free slots, we may move some job executions from contending 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 available 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 as the CF (Contention-

†Currently with Honeywell Inc., Minneapolis, USA
‡A corresponding author

1080-1812/11 $26.00 © 2011 IEEE
DOI 10.1109/RTAS.2011.30

Free) policy. Observe that the CF policy only postpones those job executions that are guaranteed to be scheduled before the job deadline by any work-conserving, preemptive algorithm. Thus, the policy reduces competing jobs by postponing some of them to contention-free slots, without imposing any penalties on the postponed jobs. This policy can be incorporated into any work-conserving, preemptive base algorithm *A*, and in this paper we denote such a CF-based algorithm as *A-CF*. The CF policy has an important property that Algorithm *A-CF* dominates[1] its base algorithm *A*.

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. 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., [14], [15], [16], [17]). As an example, we introduce a schedulability test for EDF-CF based on an existing test for EDF [16], [14], and demonstrate that the proposed test significantly dominates the existing test.

Performance of the proposed CF policy depends largely on the number of available contention-free slots. A larger number implies that more competing jobs can be postponed, and thus overall schedulability can be improved. One simple way to increase contention-free slots is to reduce the deadline of some tasks in the system. Note that reducing a task deadline is also safe from the point of view of the task's original requirements, in that it is more restrictive. Reducing task deadlines to improve schedulability is very counter-intuitive however, mainly because of the perceived difficulty in meeting the shortened deadlines. In this paper we show that combining deadline reduction with the CF policy can have a surprisingly positive impact on schedulability. We introduce novel heuristics to reduce task deadlines, with an aim to increase the number of contention-free slots and thereby overall system schedulability. We show through simulations that a significant number of task systems that were not schedulable under either EDF or EDF-CF, are eventually deemed to be schedulable when deadlines of some tasks are reduced. This deadline reduction technique not only increases the number of contention-free slots in constrained deadline systems, but also introduces such slots in implicit deadline systems of which there were none to begin with. Thus, by combining deadline reduction with the CF policy, we are able to improve schedulability even for implicit deadline systems.

In summary, this paper makes the following contributions. It proposes a novel concept of the CF policy, which can be incorporated into any work-conserving, preemptive algorithm and has the property that it dominates the base algorithm in terms of schedulability (Section III). It introduces an interference reduction technique that can improve interference-based schedulability tests of algorithms, when they employ the CF policy (Section IV). It also presents how deadline reduction improves the schedulability of CF-based algorithms (Section V). Finally, it presents further improvement on both applicability and performance of the CF policy (Section VI), and demonstrates the effectiveness of the CF policy through simulations (Section VII).

[1]Algorithm *X dominates* Algorithm *Y* if any task set schedulable by Algorithm *Y* is also schedulable by Algorithm *X*.

II. System Model

Task model. In this paper we assume a sporadic task model [18]. In this model, we specify a 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 assume a constrained deadline task system, i.e., $C_i \leq D_i \leq T_i$ for each task τ_i. A task τ_i invokes a series of jobs, each separated from its predecessor by at least T_i time units. We denote $J_{i,q}$ 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 $J_{i,q}$, 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. We denote the total number of tasks as n.

In this paper we assume quantum-based time and without loss of generality let one time unit denote the quantum length. All task parameters are assumed to be specified as multiples of this quantum length.

Multiprocessor platform. We assume that the platform is comprised of m identical unit-capacity processors, and therefore restrict the system utilization U_{sys} $(= \sum_{\tau_j \in \mathcal{T}} \frac{C_j}{T_j})$ to at most m. It has been previously shown that $U_{sys} \leq m$ is a necessary condition for feasibility of the task system considered here [19].

III. The CF Policy

In this section, we present the CF policy. We first introduce the definition and property of the contention-free slot, and present how to calculate the minimum number of contention-free slots in an interval. Second, using the minimum number, we show how the CF policy operates without knowing future release patterns of jobs. Finally, we present and prove an important property of the CF policy, which is that of dominance.

A. The contention-free slot

We express that a job $J_{i,q}$ is *active* at a time instant t if the remaining execution time at t is positive, i.e., $C_i(t) > 0$. We also express that a job $J_{i,q}$ is *available* at a time instant t if the instant is included in the interval between its release time and deadline, i.e., $t \in [r_{i,q}, r_{i,q} + D_i)$. Since a job can be

active only when it is available[2], 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.

We denote $N(t)$ as the number of available jobs at t, and express that a time slot $[t, t+1)$ is *contention-free* (*contending*) if $N(t) \leq m$ ($N(t) > m$). Then, the following property of the contention-free slot always holds:

Lemma 1: All active jobs in any contention-free slot are executed under any work-conserving scheduling algorithm.

Proof: It is clear that all active jobs at t are executed under any work-conserving algorithm if there are at most m active jobs at t. From Observation 1, the number of active jobs is upper-bounded by the number of available jobs, and thus this lemma holds. ■

In constrained deadline task systems, at most one active job per task exists in any time slot, and hence, for simplicity of presentation, we use the term "task" also to refer to "active job of a task" in the rest of this paper. Based on this observation, the above definitions of availability and contention-free slot can be easily extended from jobs to tasks. We define that a task τ_i is *available* at t if there exists an available job of that task at t. Likewise, we let $N(t)$ denote the number of available tasks at t, and define that a slot $[t, t+1)$ is contention-free if $N(t) \leq m$.

Note that it does not depend on scheduling algorithms whether a time slot is contending or not. Instead, it only depends on the release patterns of jobs. Based on this observation, we may consider using a policy to shift executions from contending slots to contention-free slots, orthogonally to any scheduling algorithm. This policy is beneficial to schedulability in that such shifting reduces the number of competing executions in the contending slots while the shifted executions are successfully performed in the contention-free slots. When we apply this policy, we need to know which slot is contention-free, meaning that we should know the future release patterns of all jobs (i.e., the policy should be clairvoyant). To make the policy applicable to more general models, such as the sporadic model considered in this paper, we do not count the actual number of contention-free slots depending on release patterns. Instead, we analyze the minimum number of contention-free slots that each job encounters in the interval between its release time and deadline, regardless of release patterns. We will explain how to determine the shifting policy using this number in Section III-B.

For this policy, we need to calculate the minimum number of contention-free slots in any interval of length l, which in turn can be obtained by computing the maximum number of contending slots in the same interval. By definition, whether a time slot is contending or not depends on the number of available tasks in that slot. We first calculate the number of slots for a task to be available in a given interval. In an interval of length l, τ_i is available during at most $\zeta_i(l)$ slots, where

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

This is because the number of slots where a task is available is maximized either 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 2 or when the deadline of the last job of τ_i in the interval $[t_c, t_d)$ of length l is the same as the end of the interval as shown in the same figure.

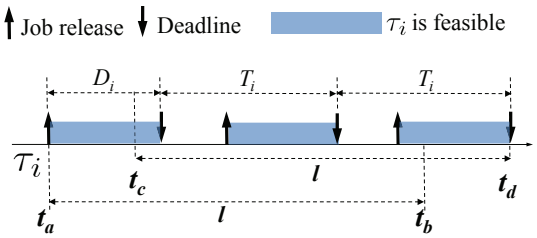


Figure 2. Situation when the number of slots where task τ_i is available is maximized

Using $\zeta_i(l)$, we can calculate the maximum number of contending slots in an interval of length l, and equivalently, we derive the minimum number of contention-free slots in the interval as follows:

Lemma 2: In an interval of length l, the number of contention-free slots is at least as many as $\Phi(l)$, where

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

Proof: If there are at most m available tasks in a time slot, the slot is contention-free. This means at least $m+1$ available tasks are required for a slot to be contending. Thus, there exist at most $\min(l, \left\lfloor \frac{\sum_{\tau_i \in \mathcal{T}} \zeta_i(l)}{m+1} \right\rfloor)$ contending slots in an interval length of l. Since any slot which is not a contending slot is contention-free by definition, we conclude the number of contention-free slots is at least $\Phi(l)$. An example of this calculation when $m = 5$ and $n = 9$ is shown in Figure 3. ■

B. Description of the CF Policy

From Lemma 2 we know that there are at least $\Phi(D_i)$ contention-free slots between the release time and deadline of any job of task τ_i. We denote $\Phi(D_i)$ as ϕ_i. When a job of τ_i is released, it knows the minimum number of

[2]We assume the situation where there is no job missing its deadline so that there is no job that is active after its deadline.

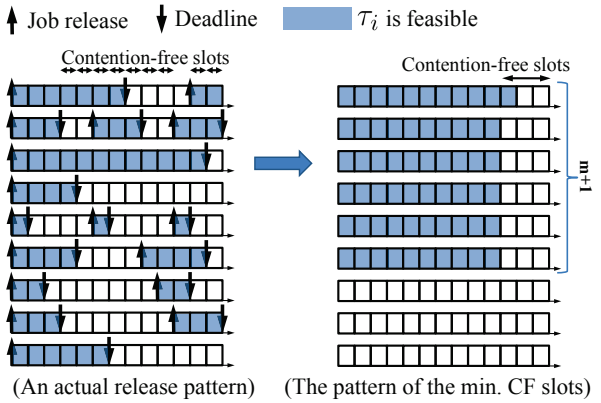


Figure 3. An example of calculating contention-free slots when $m = 5$ and $n = 9$

contention-free slots ϕ_i, 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 $\phi_i(t)$). When a job is released, $\phi_i(t)$ is set to ϕ_i. And 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 $\phi_i(t) = C_i(t)$, 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 work-conserving, preemptive base scheduling algorithm, and we call such an extended base algorithm as a CF-based scheduling algorithm. Also, we denote such a CF-based algorithm, derived from a base algorithm *A*, as *A-CF*. In the rest of the paper, we only consider work-conserving, preemptive base algorithms.

Figure 4 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 1-b), 1-c), 4), and 5), are required even when the CF policy is not incorporated in the base algorithm. The CF policy itself only requires Steps 1-a), 2), and 3), and thus has time complexity $O(n)$ for each t; at most one comparison and two updates for each job are performed.

C. Properties of the CF Policy

In the previous subsection, 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 base scheduling

1) For each job $J_{i,q}$ released at t,
   a) Set $\phi_i(t) \leftarrow \phi_i$ $(= \Phi(D_i))$.
   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 $\phi_i(t) = C_i(t)$, 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 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 satisfies $C_i(t) = 0$, remove the job from its queue.

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

Figure 4. Description of *A-CF* scheduling algorithm

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 3: When a 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 k-th execution of the job is performed in $[t, t+1)$ under Algorithm *A*. Suppose the k-th execution is not performed until t under Algorithm *A-CF*. We claim that only those active jobs, which have higher priority than $J_{i,q}$ under Algorithm *A* at time t, can have higher priority than $J_{i,q}$ under Algorithm *A-CF*.

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 the algorithms, the claim holds because active jobs in Q_H are prioritized by *A* and those in Q_L have 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: When a job $J_{i,q}$ is in the lower priority queue Q_L, the remaining executions are successfully performed within 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 remaining number of 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 above two lemmas.

Theorem 1: If a task set is schedulable by Algorithm *A*, it is also schedulable by Algorithm *A-CF*.

Proof: By Lemmas 3 and 4, 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 1, we know that the CF policy is beneficial to schedulability. In the next section, we introduce a technique to improve schedulability analysis of existing algorithms when the CF policy is incorporated.

IV. Schedulability Analysis of the CF policy

In this section, we present schedulability analysis of the CF policy. We first describe how the CF policy reduces interference from higher priority tasks. Then, we show how the interference reduction improves existing schedulability tests. While we can apply this interference reduction technique to any interference-based schedulability test (e.g., [14], [15], [16], [17]), in this paper we demonstrate it on the EDF test presented in [16], [14].

A. Interference reduction

Many existing schedulability tests (e.g., [14], [15], [16], [17]) decide schedulability using the concept of *interference*. Interference of a task τ_i on a task τ_k during some interval $[t_a, t_b)$, denotes the time duration in this interval when jobs of τ_k are waiting to execute while jobs of τ_i are executing. If the maximum interference from all tasks during an interval of length D_k is enough to block τ_k's executions by more than $D_k - C_k$, the task is deemed 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 the contending slots. We now introduce a lemma and a theorem that bound the maximum interference that a single job of task τ_i can cause on any other job.

Lemma 5: A job $J_{i,q}$ can interfere with any job in the higher priority queue Q_H 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: the job is in Q_H until its deadline.) In this case, it should hold that $\phi_i(t) < C_i(t)$ for any t. This means $\phi_i(t) = 0$ after some t' where $C_i(t') = 1$. Then, this job encounters at least ϕ_i contention-free slots while it is active. In other words, at least ϕ_i executions of this job are performed in contention-free slots. Since no job interferes with other jobs in contention-free slots, we can conclude that job $J_{i,q}$ interferes with other jobs during at most $C_i - \phi_i$ time slots.

(Case 2: the job moves to Q_L at t', i.e., $\phi_i(t') = C_i(t')$.) Before t', the job uses at least $\phi_i - \phi_i(t')$ contention-free slots. After t', the job cannot interfere with other jobs in Q_H because the job is in Q_L. Hence, at most $C_i - C_i(t') - (\phi_i - \phi_i(t')) = C_i - \phi_i$ executions of the job can interfere with other jobs in Q_H. ■

Using the above lemma and Lemma 4, we now derive in how many time slots a job can cause interference to other jobs under the CF policy.

Theorem 2: A job $J_{i,q}$ can interfere with any job in either Q_H or Q_L during at most $C_i - \phi_i$ time slots.

Proof: From Lemma 5 we guarantee that a job can interfere with any job in Q_H during at most $C_i - \phi_i$ time slots. The remaining part is to analyze the case that a job interferes with other jobs in Q_L. Lemma 4 guarantees that the remaining executions of any job in Q_L are scheduled by its deadline in contention-free slots. This means no job can cause interference to any job in Q_L. This proves the theorem. ■

B. Schedulability Analysis of EDF-CF

Theorem 2 indicates that we can reduce interference when the CF policy is employed. We now describe how this reduction can be applied to the EDF schedulability test presented in [16], [14]. In other words, we introduce a new schedulability test for EDF-CF.

We first re-visit the test in [16], [14]. This test checks whether a task τ_k has enough interference from other tasks to miss its deadline. Since finding the maximum interference is hard, it uses an upper bound based on the task release pattern depicted in Figure 5. Since a task with a later deadline cannot interfere with another task with an earlier deadline under EDF, the maximum interference of τ_i on τ_k occurs when the deadlines of two tasks are aligned as shown in the figure. This interference during an interval of length l (denoted by $I_{k,i}^{\mathsf{EDF}}(l)$) is given by

$$I_{k,i}^{\mathsf{EDF}}(l) = \left\lfloor \frac{l}{T_i} \right\rfloor C_i + \min\left(C_i, l - \left\lfloor \frac{l}{T_i} \right\rfloor T_i\right). \quad (3)$$

Using $I_{k,i}^{\mathsf{EDF}}(l)$, the following lemma introduces a schedulability test for EDF.

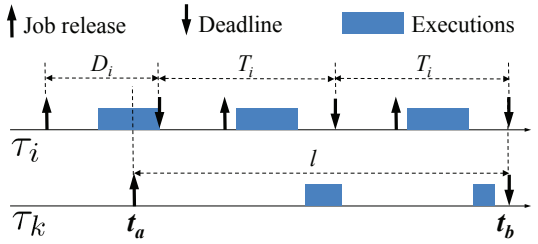


Figure 5. Situation when the maximum interference occurs under EDF

Lemma 6 (Theorem 7 in [16]): A task set is schedulable under EDF if the following inequality holds for each task τ_k:

$$\sum_{i \neq k, \tau_i \in \mathcal{T}} \min(I_{k,i}^{\mathsf{EDF}}(D_k), D_k - C_k + 1) < m \cdot (D_k - C_k + 1). \tag{4}$$

Proof: The proof is given in Theorem 5 in [16]. To summarize, for τ_k to miss its deadline, τ_k is executed at most $C_i - 1$ 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 tasks 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 (Theorem 1), we can use Lemma 6 as a schedulability test for EDF-CF. However, we now introduce a tighter schedulability analysis using the interference reduction technique presented earlier.

Theorem 2 indicates that at most $C_i - \phi_i$ executions of a job $J_{i,q}$ interferes with other jobs. Instead of counting the whole execution time C_i as interference as in Eq. (3), we only count $C_i - \phi_i$ executions as interference. With this modification, Lemma 6 can be modified as follows:

Theorem 3: A task set is schedulable under EDF-CF if the following inequality holds for each task τ_k:

$$\sum_{i \neq k, \tau_i \in \mathcal{T}} \min(I_{k,i}^{\mathsf{EDF\text{-}CF}}(D_k), D_k - C_k + 1) < m \cdot (D_k - C_k + 1) \tag{5}$$

where

$$I_{k,i}^{\mathsf{EDF\text{-}CF}}(l) = \left\lfloor \frac{l}{T_i} \right\rfloor (C_i - \phi_i) + \min\left(C_i - \phi_i, l - \left\lfloor \frac{l}{T_i} \right\rfloor T_i\right). \tag{6}$$

Proof: The theorem holds from Theorem 2 and Lemma 6. ■

It is easy to see that Theorem 3 dominates Lemma 6. Although we have chosen EDF-CF to demonstrate the interference reduction technique, it can be easily applied to any interference-based schedulability analysis that calculates interference similar to Eq. (3), such as the ones for any work-conserving algorithm [16], fixed-priority algorithms [16], EDZL [15] and LLF [17].

V. Improving Schedulability by Deadline Reduction

Real-time scheduling aims at meeting the resource requirements of each job. That is, C_i executions should be performed within its deadline D_i. While increasing the deadline of a job violates this requirement, reducing it does not; a shorter deadline only forces the job to finish earlier than required. So far however, there are no studies that consider deadline reduction for better schedulability, mainly because deadline reduction provides no benefit to existing scheduling algorithms or schedulability tests. On the contrary, a shorter deadline gives less time for jobs to complete their executions.

The CF policy however, can benefit from deadline reduction. Whenever the deadline of a job is reduced, the number of slots for the job to be available in an interval is also reduced, and then the number of contention-free slots in the interval may increase. This can result in better schedulability. Considering both the negative effect of reduced time to deadline and the positive effect of increased contention-free slots, in this section we present deadline reduction techniques for better schedulability of the EDF-CF test in Theorem 3. Deadline reduction for improved schedulability is a novel concept, whose benefits are specific to the CF policy proposed in this paper.

A. The effect of deadline reduction on the schedulability of EDF-CF

We may change the relative deadline D_k of a task to D'_k where $D'_k = C_k, C_k + 1, ..., D_k$. This means we have $D_k - C_k + 1$ options for the deadline of a task, and the number of all possible combinations of deadlines for a task set is $\prod_{\tau_k \in \mathcal{T}}(D_k - C_k + 1)$. Examining all the combinations is beneficial for schedulability in that a task set is deemed schedulable if we find just one combination that satisfies Theorem 3. However, such an exhaustive search requires exponential time-complexity. Therefore, to develop a tractable deadline reduction technique, we must investigate only some combinations. This entails the analysis of identifying which deadline combinations are most likely to be schedulable. For this purpose, we first analyze how the deadline reduction of a single task affects the minimum number of contention-free slots for each task.

Deadline reduction of a task τ_j has two different impacts: (a) possible increase in $\{\phi_h\}_{h \neq j, \tau_h \in \mathcal{T}}$, and (b) possible decrease in ϕ_j. Once D_j is reduced, the number of maximum slots where τ_j is available during an interval of length D_h is also reduced ($\zeta_j(D_h)$ in Eq. (1)), and this leads to impact (a). On the other hand, if we look at $\phi_j = \phi(D_j)$, the interval length $l = D_j$ itself in Eq. (2) decreases, and this results in impact (b).

The next step is to analyze how the deadline reduction of a single task affects the schedulability of EDF-CF in Theorem 3. We re-arrange the inequalities in Eq. (5) so that the RHS does not vary with tasks as follows:

$$\frac{m \cdot (C_k - 1)}{D_k} + \frac{\sum_{i \neq k, \tau_i \in \mathcal{T}} \min(I_{k,i}^{\mathsf{EDF\text{-}CF}}(D_k), D_k - C_k + 1)}{D_k} < m \quad \text{, for all } \tau_k \in \mathcal{T}. \tag{7}$$

We look at how the deadline reduction of a task affects the first and second term of the LHS of inequalities in Eq. (7). Suppose we reduce D_j and keep the relative deadlines of other tasks as is. Then, the first term for task τ_j gets larger due to the decreased denominator ($k = j$ in Eq. (7)), while the first term for $\{\tau_h\}_{\tau_h \in \mathcal{T}, h \neq j}$ does not change. The change in the second term is more complicated and unpredictable due to the changes in $\{\phi_h\}_{\tau_h \in \mathcal{T}}$. In the second term for task τ_j, while the denominator decreases, $I_{k,i}^{\mathsf{EDF\text{-}CF}}(D_k)$ may also decrease due to reduction in the interval length (D_j) and increases in $\{\phi_i\}_{i \neq j}$. In the second term for $\{\tau_h\}_{\tau_h \in \mathcal{T}, h \neq j}$, the interval length of $I_{k,i}^{\mathsf{EDF\text{-}CF}}(D_k)$ and the denominator of the term do not change. However, $I_{k,i}^{\mathsf{EDF\text{-}CF}}(D_k)$ for $\{\tau_i\}_{\tau_i \in \mathcal{T}, i \neq j}$ may decrease due to the possible increase in $\{\phi_i\}_{i \in \mathcal{T}, i \neq j}$, while that for τ_j may increase due to the possible decrease in ϕ_j.

To summarize, the deadline reduction of a single task affects the schedulability of each task in a negative and positive way at the same time, so that there is a possibility of improving schedulability through deadline reduction. Note that the positive effect of deadline reduction on schedulability is derived solely from the CF policy. In other words, schedulability tests of algorithms that do not employ the CF policy can hardly be improved through this technique. For example, deadline reduction has almost no impact on the LHS of inequalities in Eq. (7) (test for EDF). This is because there is no effect of $\{\phi_i\}_{\tau_i \in \mathcal{T}}$ on $I_{k,i}^{\mathsf{EDF}}(l)$, and as a result $I_{k,i}^{\mathsf{EDF}(l)}/l$ does not generally get smaller when l is reduced. In the next subsection, we introduce heuristics for deadline reduction based on the above analysis.

B. Heuristics for deadline reduction

For tractability, we need to investigate a polynomial number of combinations of deadlines. Hence we consider the following procedure with a step size of α: the initial deadline of each task is the same as the original deadline; we reduce the deadline of a task by α, where the task is chosen based on a heuristic (described later); and such a reduction is repeated until the current setting of deadlines is deemed schedulable by Theorem 3 or the deadlines of at least $m+1$ tasks are equal to their execution times. It is trivially true that $m+1$ tasks satisfying $D_k = C_k$ are not schedulable on a platform comprised of m processors, and therefore we stop the process once we satisfy this condition. Figure 6 gives a formal description of this deadline reduction procedure.

1) For each $\tau_i \in \mathcal{T}$,
   a) Calculate the LHS of inequalities in Eq. (7). If the value is not smaller than m, go to 3)
2) If the LHS of inequalities in Eq. (7) are all less than m, the task set is deemed schedulable.
3) Check whether there are at least $m + 1$ tasks satisfying $D_k = C_k$. If so, the task set is deemed unschedulable.
4) Choose τ_j among tasks satisfying $D_j > C_j$ using the given heuristic. Set $D_j \leftarrow \max(C_j, D_j - \alpha)$

Figure 6. Description of the deadline reduction procedure

Here, the number of iterations is $O(\sum_{k \in \mathcal{T}} \lceil (D_k - C_k)/\alpha \rceil)$, which is at most $n \cdot \max_{\tau_k \in \mathcal{T}}(D_k - C_k)$ even when α is 1. The next step is to determine a suitable value for step-size α and heuristics for task selection.

In this paper we choose α as $\max_{\tau_k \in \mathcal{T}}(D_k - C_k)$ so that the number of iterations of Steps 1) - 4) in Figure 6 is $O(m)$, resulting in a total time complexity of $O(m \cdot n^2)$. While it is trivial that a larger α leads to a low time-complexity procedure, we observed that the resulting improvement in schedulability is also comparable to those obtained with smaller values of α. From this observation, we can infer that one of the effective strategies to make task systems schedulable is to keep original deadlines of some tasks and reduce deadlines of other tasks as much as possible. Reduced deadline tasks generate additional contention-free slots, which can be utilized to schedule those tasks whose deadlines have not been reduced. This further decreases contention for the reduced deadline tasks, which enables the scheduler to successfully schedule them within those highly constrained deadlines.

We consider three heuristics for choosing a task whose deadline is reduced in some iteration: (i) the largest density (C_i/D_i) first, (ii) the smallest laxity ($D_i - C_i$) first, and (iii) the largest LHS of inequalities in Eq. (7) first. We chose (i) and (ii) because they identify tasks that are the most difficult to schedule. Moreover, the negative impact of deadline reduction on the first term of the LHS in Eq. (7) is the least for tasks selected by (i). That is, C_i/D_i increases by the smallest amount for the largest density task, when D_i is reduced to C_i. Likewise, the negative impact on the denominator of the LHS is minimized for tasks selected by (ii), because the decrease in deadline is the smallest possible. Heuristic (iii) also tries to identify difficulty to schedule tasks, but instead of relying on task parameters, it uses Eq. (7). Theorem 3 can be re-phrased such that a task set is deemed schedulable if the maximum of the LHS of inequalities in Eq. (7) is less than m. In order to reduce the maximum, (iii) identifies the task that contributes this maximum value.

To evaluate the performance of the three heuristics, in

Section VII we consider seven heuristics: random choice as a reference point, (i), (ii) and (iii), and their corresponding counter approaches (e.g., the smallest density first is a counter approach to (i)).

VI. Discussion

In this section, we discuss two issues regarding applicability of the deadline reduction technique and further schedulability improvement of the CF policy.

A. Deadline reduction for implicit deadline task systems

For implicit deadline task systems where the deadline of each task is equal to its period, we cannot derive any contention-free slots because it is zero the minimum interval between the deadline of a job of a task and the release time of the next job of the same task. This means, the CF policy cannot improve the schedulability of base algorithms for implicit deadline task systems.

However, we still apply the CF policy for better schedulability using the deadline reduction technique in Figure 6. Once the deadlines of implicit deadline tasks are reduced, contention-free slots come into existence, and then the CF policy operates for better schedulability. We will demonstrate how much this CF policy improves the schedulability of implicit deadline task systems in Section VII.

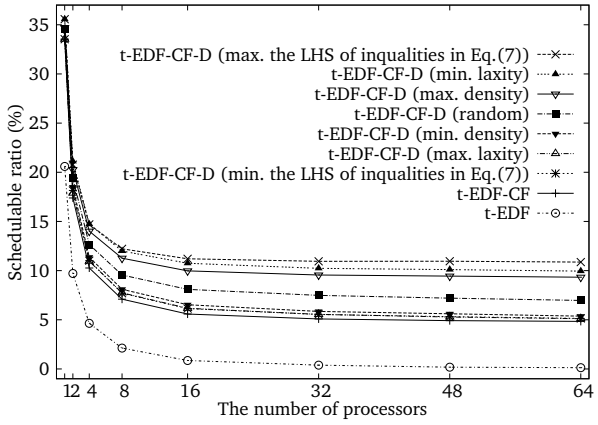
B. Further schedulability improvement of the CF policy for periodic tasks

In this subsection we address how to further improve the schedulability of the CF policy when task releases are strictly periodic [12]. The CF policy described in Figure 4 uses the minimum number of contention-free slots by Eq. (2) assuming the worst-case release patterns, and this makes it possible for the policy to operate without knowing which future time slots are contention-free. In case that we know all the future releases (e.g., a periodic task model), we can derive a larger lower-bound of the number of contention-free slots by considering the actual release pattern of jobs. For example, in Figure 3, while the minimum number of contention-free slots is 3, the exact number is 9. These increased contention-free slots improve schedulability by moving more executions in contending slots to contention-free ones.

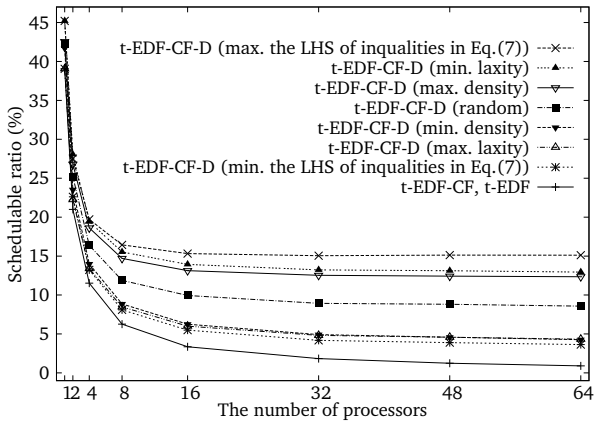
In Section VII, we evaluate the EDF-CF* algorithm that employs the larger lower-bound of contention-free slots under a periodic task model. Note that the schedulability test for EDF-CF in Theorem 3 also holds for EDF-CF*.

VII. Performance Evaluation

This section presents simulation results to evaluate the performance of the proposed CF policy and its schedulability test.



(a) Constrained deadline task systems



(b) Implicit deadline task systems

Figure 7. The schedulability of the schedulability tests t-EDF, t-EDF-CF, and t-EDF-CF-D with various heuristics over varying number of processors

A. Evaluation Settings

We generate task sets based on a technique proposed earlier [20], which has also been used in many previous studies (e.g., see [16], [21]). We have three input parameters: (a) the number of processors m ($1, 2, 4, 8, 16, 32, 48$ or 64), (b) the task system (constrained or implicit deadline), and (c) individual task utilization (C_i/T_i) distribution (bimodal with parameter[3]: 0.1, 0.3, 0.5, 0.7, or 0.9, or exponential with parameter[4]: 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 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

[3]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 $1 - p$.

[4]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 $\lambda \cdot \exp(-\lambda \cdot x)$.

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 [19], [22].
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 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 system, 10,000 task sets are created for each task utilization model, thus resulting in 100,000 task sets in total. We evaluate the performance of three schedulability tests: 1) the EDF test in Lemma 6, 2) our proposed EDF-CF test in Theorem 3, and 3) our proposed EDF-CF test with deadline reduction in Figure 6. These tests are respectively annotated as 't-EDF', 't-EDF-CF', and 't-EDF-CF-D'. Note that the tests are independent of whether a task set is comprised of periodic or sporadic tasks; they only depend on the task parameters T_i, C_i and D_i.

We also perform actual simulation of the EDF and EDF-CF scheduling algorithms over the generated task sets during the first 100,000 time units for tractability. Results from these simulations show the effectiveness of the CF policy in improving schedulability. Whether a task set consists of periodic or sporadic tasks has a direct impact on schedulability of the scheduling algorithms, and hence we simulated both the cases. We observed that the overall results under the sporadic release are the same as that under the periodic release, except that the sporadic release slightly increases schedulability. Therefore, due to lack of space, we only present results from the periodic release in this paper. We annotate EDF and EDF-CF as 'EDF' and 'EDF-CF'. We also present simulation results of another EDF-CF algorithm, which utilizes the knowledge of future releases as described in Section VI-B. We annotate this algorithm as 'EDF-CF*'.

B. Comparison of the heuristics for deadline reduction

Figure 7 evaluates seven heuristics for the deadline reduction procedure presented in Figure 6: the proposed three heuristics in Section V-B including (i) the largest density (C_i/D_i) first, (ii) the smallest laxity ($D_i - C_i$) first and (iii) the largest LHS of inequalities in Eq. (7) first, and their corresponding counter approaches including (iv) the smallest density first, (v) the largest laxity first and (vi) the smallest LHS of inequalities in Eq. (7) first, and lastly a non-strategic heuristic - (vii) random choice. For both constrained and implicit task sets in Figure 7(a) and (b), respectively, the schedulability of (iii) is better than that of (i) and (ii). However, (i) and (ii) are also effective heuristics in that they definitely outperform the reference point (vii). We also observe that the schedulability of the counter approaches (iv), (v) and (vi) are worse than that of (vii). This observation

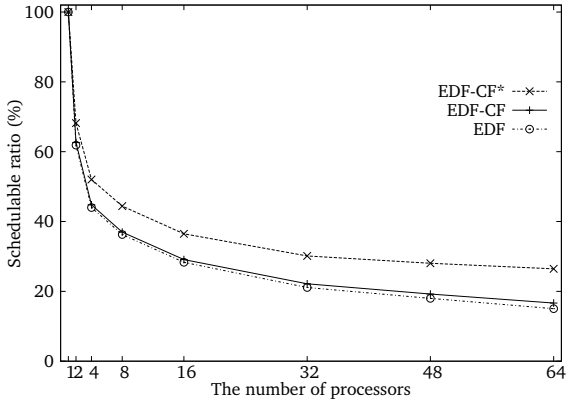


Figure 8. The schedulability of constrained deadline task systems over varying number of processors identified using scheduling algorithms EDF, EDF-CF and EDF-CF*

lends further support to the view that heuristics (i), (ii) and (iii) are generally effective in improving schedulability. We employ the best heuristic (iii) for t-EDF-CF-D, and present results in the following subsection.

C. The schedulability improvement of the CF policy

Figure 7(a) plots the percentage of constrained deadline task sets deemed schedulable by schedulability tests t-EDF, t-EDF-CF, and t-EDF-CF-D for a variety of the number of processors (m). While t-EDF only deems few task sets schedulable as m increases, t-EDF-CF deems about 5% of the task sets schedulable even when $m = 64$, and this improvement comes from the interference reduction associated with the CF policy. Moreover, if we apply the deadline reduction technique in Figure 6, the number of task sets deemed schedulable by t-EDF-CF-D is more than doubled when compared to t-EDF-CF.

Now we look at the percentage of constrained deadline task sets deemed schedulable by the scheduling algorithms EDF, EDF-CF, and EDF-CF* in Figure 8. We first confirm that the performance of EDF-CF dominates that of its base algorithm EDF. Second, we observe that the schedulability of EDF-CF* is remarkably improved when compared to that of EDF, and the gap between EDF and EDF-CF* gets amplified as m increases. For example, the schedulability of EDF-CF* when $m = 64$ is 1.77 times larger than that of EDF (EDF-CF*: 26.7%, EDF: 15.1%). This means the CF policy significantly enhances the schedulability of EDF if we know future release patterns, and this enhancement is also scalable with the number of processors.

The next results are aimed at implicit deadline task sets. Figure 7(b), which corresponds to Figure 7(a), plots the percentage of implicit deadline task sets deemed schedulable by the schedulability tests t-EDF, t-EDF-CF, and t-EDF-CF-D. Here the CF policy does not have an impact on schedulability, unless deadline reduction is used. This means that the schedulability ratio of t-EDF-CF is the same as that of

t-EDF (in Figure 7(b)). However, as seen in the same figure, the CF policy improves schedulability using the deadline reduction technique of Figure 6. While t-EDF (as well as t-EDF-CF) rarely finds schedulable task sets as m increases, t-EDF-CF-D deems about 15% task sets schedulable when $m = 64$.

In summary, the CF policy significantly improves the schedulability of EDF only when it has knowledge of future (as in the case of periodic task systems), but our proposed schedulability test associated with the CF policy remarkably improves the schedulability of EDF in all cases.

VIII. Conclusions

In this paper we have presented the CF policy that takes advantage of contention-free slots. We have shown that the CF policy, which can be incorporated into any work-conserving, preemptive algorithm, improves not only existing algorithms themselves but also the schedulability tests of those algorithms. Further, we have derived a counter-intuitive claim that reducing task deadlines can help schedulability when the CF policy is employed.

Note that the proposed CF policy can be employed independently of other policies like zero-laxity (ZL), to achieve a cummulative improvement in schedulability. For example, the EDZL algorithm [15] can be further improved by expanding it with the CF policy. It is also worth noting that the schedulability improvement arising from deadline reduction is inherently tied to the CF policy, and therefore cannot be achieved under other policies, including ZL.

One disadvantage of the CF policy is it may incur additional preemptions. We plan to understand how the policy imposes such extra preemptions and derive some bound on the number of additional preemptions. Another direction for future work is to apply the CF policy to other scheduling algorithms.

Acknowledgements[5]

This work was supported in part by the IT R&D Program of MKE/KEIT [2011-KI002090, Development of Technology Base for Trustworthy Computing], Basic Research Laboratory (BRL) Program (2009-0086964), Basic Science Research Program (2010-0006650), and the Personal Plug&Play DigiCar Research Center (NCRC, 2010-0028680) through the National Research Foundation of Korea (NRF) funded by the Korea Government (MEST), KAIST-Microsoft Research Collaboration Center, KAIST ICC, and KI-DCS grants.

This work was also partially funded by the Portuguese Science and Technology Foundation (FCT), the European Commission (ARTISTDesign), the ARTEMIS-JU (RECOMP), and the Luso-American Development Foundation (FLAD).

[5]The information in this document is provided "as is", and no guarantee or warranty is given that the information is fit for any particular purpose. The user uses the information at its sole risk and liability.

References

[1] S. Cho, S.-K. Lee, S. Ahn, and K.-J. Lin, "Efficient real-time scheduling algorithms for multiprocessor systems," *IEICE Trans. on Communications*, vol. E85–B, no. 12, pp. 2859–2867, 2002.

[2] A. Srinivasan and S. Baruah, "Deadline-based scheduling of periodic task systems on multiprocessors," *Information Processing Letters*, vol. 84, no. 2, pp. 93–98, 2002.

[3] B. Andersson, S. Baruah, and J. Jonsson, "Static-priority scheduling on multiprocessors," in *RTSS*, 2001, pp. 193–202.

[4] S. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, "Proportionate progress: a notion of fairness in resource allocation," *Algorithmica*, vol. 15, no. 6, pp. 600–625, 1996.

[5] H. Cho, B. Ravindran, and E. D. Jensen, "An optimal real-time scheduling algorithm for multiprocessors," in *RTSS*, 2006, pp. 101–110.

[6] J. H. Anderson and A. Srinivasan, "Early-release fair scheduling," in *ECRTS*, 2000, pp. 35–43.

[7] B. Andersson and E. Tovar, "Multiprocessor scheduling with few preemptions," in *RTCSA*, 2006, pp. 322–334.

[8] K. Funaoka, S. Kato, and N. Yamasaki, "Work-conserving optimal real-time scheduling on multiprocessors," in *ECRTS*, 2008, pp. 13–22.

[9] B. Andersson and K. Bletsas, "Sporadic multiprocessor scheduling with few preemptions," in *ECRTS*, 2008, pp. 243–252.

[10] A. Easwaran, I. Shin, and I. Lee, "Optimal virtual cluster-based multiprocessor scheduling," *Real-Time Systems*, vol. 43, no. 1, pp. 25–59, 2009.

[11] J. Lee, A. Easwaran, I. Shin, and I. Lee, "On optimal multiprocessor scheduling considering concurrency and urgency," in *RTSS Work-in-Progress Session*, 2010, pp. 21–24.

[12] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[13] N. Fisher, J. Goossens, and S. Baruah, "Optimal online multiprocessor scheduling of sporadic real-time tasks is impossible," *Real-Time Systems*, vol. 45, pp. 26–71, 2010.

[14] M. Bertogna, M. Cirinei, and G. Lipari, "Improved schedulability analysis of EDF on multiprocessor platforms," in *ECRTS*, 2005, pp. 209–218.

[15] T. P. Baker, M. Cirinei, and M. Bertogna, "EDZL scheduling analysis," *Real-Time Systems*, vol. 40, pp. 264–289, 2008.

[16] M. Bertogna, M. Cirinei, and G. Lipari, "Schedulability analysis of global scheduling algorithms on multiprocessor platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, pp. 553–566, 2009.

[17] J. Lee, A. Easwaran, and I. Shin, "LLF schedulability analysis on multiprocessor platforms," in *RTSS*, 2010, pp. 25–36.

[18] A. Mok, "Fundamental design problems of distributed systems for the hard-real-time environment," Ph.D. dissertation, Massachusetts Institute of Technology, 1983.

[19] T. P. Baker and M. Cirinei, "A necessary and sometimes sufficient condition for the feasibility of sets of sporadic hard-deadline tasks," in *RTSS*, 2006, pp. 178–190.

[20] T. P. Baker, "Comparison of empirical success rates of global vs. paritioned fixed-priority and edf scheduling for hand real time," Dept. of Computer Science, Florida State University, Tallahasee, Tech. Rep. TR-050601, 2005.

[21] B. Andersson, K. Bletsas, and S. Baruah, "Scheduling arbitrary-deadline sporadic task systems on multiprocessor," in *RTCSA*, 2008, pp. 197–206.

[22] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, and S. Stiller, "Implementation of a speedup-optimal global edf schedulability test," in *ECRTS*, 2009, pp. 259–268.