# Global EDF-based Scheduling with Efficient Priority Promotion *

Shinpei Kato and Nobuyuki Yamasaki
Department of Information and Computer Science
Keio University, Yokohama, Japan
{shinpei,yamasaki}@ny.ics.keio.ac.jp

## Abstract

*This paper presents an algorithm, called Earliest Deadline Critical Laxity (EDCL), for the efficient scheduling of sporadic real-time tasks on multiprocessors systems. EDCL is a derivative of the Earliest Deadline Zero Laxity (EDZL) algorithm in that the priority of a job reaching certain laxity is imperiously promoted to the top, but it differs in that the occurrence of priority promotion is confined to at the release time or the completion time of a job. This modification enables EDCL to bound the number of scheduler invocations and to relax the implementation complexity of scheduler, while the schedulability is still competitive with EDZL. The schedulability test of EDCL is designed through theoretical analysis. In addition, an error in the traditional schedulability test of EDZL is corrected. Simulation studies demonstrate the effectiveness of EDCL in terms of guaranteed schedulability and exhaustive schedulability by comparing with traditional efficient scheduling algorithms.*

## 1 Introduction

The scheduling of recurrent real-time tasks on multiprocessors has been one of the primary subjects in the real-time computing community, ever since Dhall and Liu demonstrated that the Earliest Deadline First (EDF) scheduling [13] is no longer optimal on multiprocessors [10]. Because the trend of embedded real-time systems has been moving to the multicore platforms in recent years, research on this subject is now more significant.

It is widely known that a set of independent periodic tasks, in which the deadline of each task is equal to its period, is always successfully scheduled by EDF on a single processor if the total utilization of the tasks does not exceed 1. Unfortunately, this optimality of EDF breaks down on multiprocessor systems. A set of periodic tasks cannot be guaranteed to be schedulable on $m$ identical processors if the total utilization exceeds $m(1 - u_{max}) + u_{max}$, where $u_{max}$ is the maximum utilization of every individual task [11]. In the worst case, letting $u_{max} = 1$, EDF may cause a deadline to be missed if the total utilization is slightly greater than 1, even though there are $m$ processors. Such performance deterioration is often called *Dhall's effect*.

There are mainly two concepts in the scheduling of real-time tasks on multiprocessors: optimality and simplicity. The Pfair family [4, 5, 1] and LLREF [7] are optimal algorithms that achieve the full use of processor time with guaranteeing all tasks to meet deadlines. In other words, a set of periodic tasks is always successfully scheduled by the optimal algorithms on $m$ processors if the total utilization does not exceed $m$. On the other hand, the timing constraints of tasks are generally guaranteed by schedulability test in other efficient algorithms, such as EDF-US[$x$] [15, 2] and EDZL [8, 9]. The primary objective of those algorithms is an efficient scheduling to offer good performance rather than an optimal scheduling with complex computation. Since they are designed with simplified theories, the scheduling overhead and the implementation complexity are much smaller than the optimal algorithms, though the guaranteed total utilization is usually less than $m$. Nonetheless, they perform far better than EDF. Thus, this paper settles on the subject matter of such an efficient scheduling without complexity from the viewpoint of practical use.

EDF-US[$x$] overcomes the weakness of EDF in an elementary way. In EDF-US[$x$] scheduling, the tasks with individual utilizations greater than $x$ are statically assigned the highest priority. Baker showed that $x = 1/2$ is an optimal configuration for this algorithm, which leads to that a set of tasks is successfully scheduled on $m$ processors if the total utilization is no greater than $(m + 1)/2$ [2]. Unfortunately, EDF-US[$x$] can perform worse than EDF depending on the characteristics of given tasks, namely a set of tasks that can be successfully scheduled by EDF may not be successfully scheduled by EDF-US[$x$].

EDZL is another alternative of EDF, which is known to be at least as effective as EDF. That is, a set of tasks that can be successfully scheduled by EDF can be also successfully scheduled by EDZL. In EDZL scheduling, jobs are prioritized by the EDF policy as long as the laxity of every job is positive. If any job reaches zero laxity, its priority is imperiously promoted to the top to meet its deadline. Accordingly, the number of scheduler invocations is not far beyond EDF, while it offers excellent performance far beyond EDF. Piao *et al.* proved that a set of periodic tasks can be successfully

*This work is supported by the fund of Research Fellowships of the Japan Society for the Promotion of Science for Young Scientists. This work is also supported in part by the fund of Core Research for Evolutional Science and Technology, Japan Science and Technology Agency.

scheduled by EDZL on $m$ processors if the total utilization does not exceed $(m+1)/2$ [14]. Further schedulability analysis of EDZL was presented by Cirinei and Baker [9]. They considered more precise schedulability test of EDZL by applying the technique of the schedulability test of EDF devised by Bertogna *et al.* [6].

In EDZL scheduling, a scheduler must be invoked every time any job reaches zero laxity. Since the zero laxity can occur at any time, the number of scheduler invocations is not bounded. In addition, fine-grained timers are required to make the scheduling points for the priority promotions at zero laxity. Meanwhile the scheduling points of EDF and EDF-US[$x$] are made only at job releases and completions. In other words, a scheduler needs to be invoked only when jobs are released and complete. Thus, the number of scheduler invocations is bounded. Since the use of fine-grained timers takes additional implementation costs and the increase of scheduler invocations boosts run-time overhead, we consider a derivative of EDZL in this paper to bound the number of scheduler invocations and to relax the implementation complexity of scheduler.

This paper presents an EDF-based algorithm, called Earliest Deadline Critical Laxity (EDCL), for the efficient scheduling of sporadic real-time tasks on multiprocessor systems. The basic concept of EDCL is that the priority of a job does not necessarily have to be promoted at zero laxity to meet its deadline but can be promoted at the release time or the completion time of a job. This modification of the priority promotion concept leads to that a scheduler needs to be invoked only at job releases and completions. Thus, the number of scheduler invocations is bounded and the implementation complexity of scheduler is relax, while the schedulability is still competitive with EDZL.

The remainder of this paper is organized as follows. The system model and the terminology are defined in the next section. Section 3 presents the algorithm design. Section 4 then gives the schedulability analysis. The effectiveness of the presented algorithm is demonstrated though simulation studies in Section 5. We conclude this work in Section 6.

## 2 System Model

The system is modeled with $m$ identical processors and a set of $n$ sporadic tasks, denoted by $\tau = \{\tau_1, \tau_2, ..., \tau_n\}$. Each task $\tau_i$ is characterized by tuple $(c_i, d_i, p_i)$, where $c_i$ is a worst-case computation time, $d_i$ is a relative deadline, and $p_i$ is a minimum inter-release time (period). Note that $c_i \leq \min\{d_i, p_i\}$ must hold, otherwise $\tau_i$ would never meet a deadline. A task generates an infinite sequence of jobs. The execution time of every job of $\tau_i$ is assumed to be $c_i$. The inter-release intervals of any successive jobs of $\tau_i$ are separated by at least $p_i$. When a job of $\tau_i$ is released at time $t$, the job has a deadline at time $t + d_i$. The remaining execution time of a job of $\tau_i$ executing at time $t$ is denoted by $e_i(t)$, and its laxity is denoted by $x_i(t)$. Note that $t + e_i(t) + x_i(t) = d_i$ holds for any $t$. For the sake of generalization, the job window $\Delta_i$ is defined to be $\Delta_i = \min\{d_i, p_i\}$, and the utilization (density) $\lambda_i$ is defined to be $\lambda_i = c_i/\Delta_i$. Finally, the total utilization $U(\tau)$ is defined to be $U(\tau) = \sum_{\tau_i \in \tau} \lambda_i$.

A set of ready tasks at any time is denoted by $\gamma$. All tasks are independent and preemptive. More than one task cannot be executed simultaneously on a processor. A task cannot be processed in parallel. Jobs of the same task must be executed sequentially, which means that every job of $\tau_i$ cannot start before the preceding job of $\tau_i$ completes.

## 3 Algorithm Description

This section gives a theoretical description of EDCL. EDCL is a derivative of EDZL in that the priority of a job reaching certain laxity is imperiously promoted to the top, but it differs in that the occurrence of priority promotion is confined to at the release time or the completion time of a job. This modification to EDZL simplifies the design of algorithm, since a scheduler needs to be invoked only at job releases and completions. Hence, there is no need to use fine-grained timers to make scheduling points.

We first of all explore the scheduling point at which a job *would* miss its deadline unless it is scheduled immediately. Then, the algorithm is designed so that the priority of a job which reaches such a point in EDF scheduling is forcefully promoted to the top to meet its deadline. For simplicity of description, let $t_s$ denote any time at which any job is released or completes henceforth.

**Definition 1.** *A job of task $\tau_i$ is said to be **critical**, if its laxity holds the following condition at time $t_s$, where $e_{min}$ denotes the minimum remaining execution time of the m jobs that have the earliest deadlines.*

$$x_i(t_s) < e_{min} \tag{1}$$

*Then, the laxity $x_i(t_s)$ is defined **critical laxity**.*

It is clear that a critical job will miss its deadline in EDF scheduling unless it is dispatched for execution, since the scheduler will never dispatch the critical job before at least one of the $m$ jobs with the earliest deadlines completes. An example of such a deadline miss with three processors is depicted in Figure 1. A gray-color job with deadline $t_d$ becomes critical at time $t_s$ due to its laxity less than $e_{min}$. This job will never meet its deadline, because it can never resume execution before time $t_s + e_{min}$ in EDF scheduling. Note that jobs with laxity equal to $e_{min}$ are not included in critical jobs. Those tasks can possibly complete at their deadlines, only if they can resume execution right after the job with remaining execution time $e_{min}$ completes.

In order to avoid such a deadline miss without incurring additional scheduling points, EDCL gives the highest priority to jobs which reach critical laxity at any job releases and completions. As long as no jobs are critical, EDCL completely performs as EDF. Even when it performs differently, the scheduling points are made only at job releases and completions. In consequence, EDCL is able to bound the number of scheduler invocations and to relax the implementation complexity of scheduler, compared with EDZL.

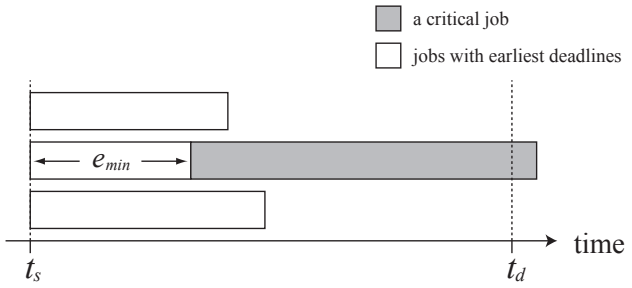


**Figure 1. A deadline miss in EDF scheduling**

**Theorem 1.** *EDCL is at least as effective as EDF.*

*Proof.* EDCL performs as EDF until any job becomes critical but is not scheduled. EDF will surely cause such a critical job to miss its deadline, while EDCL may be able to save it by the priority promotion. Hence, the theorem is true. □

**Theorem 2.** *The number of scheduler invocations per job release in any interval for EDCL is at most* 2.

*Proof.* Let $J(I)$ be the number of jobs which are released in any interval $I$. Since a scheduler is invoked only when jobs are released or complete, it is clear that the number of scheduler invocations in interval $I$ is bounded to $2J(I)$. Hence, the theorem is true. □

Figure 2 illustrates the procedure of the EDCL scheduler. The scheduler is invoked only when **(i)** jobs with earlier deadlines than the current jobs are released, or **(ii)** any jobs complete. If there are less than $m$ ready tasks, the scheduler executes all the ready tasks. Otherwise, the scheduler dispatches $m$ tasks with the earliest deadlines in a set of ready tasks. Then, it computes the minimum remaining execution time of the $m$ tasks, which is denoted by $e_{min}$. If there are ready jobs with laxity less than $e_{min}$, the priorities of those jobs are promoted to the top to avoid missing their deadlines. Notice that the scheduler needs to break ties among critical jobs if the number of such critical jobs is greater than $m$, since no more than $m$ jobs can be executed simultaneously on $m$ processors. In this paper, we take several policies for tie breaking: ties are broken **(i)** arbitrarily, **(ii)** in favor of shorter remaining execution time, **(iii)** in favor of less laxity, and **(iv)** in favor of earlier deadline.

The first policy makes a benefit that all the scheduler has to do is to dispatch the first $m$ critical jobs examined by linear search, meaning that there is no need to scan all the critical jobs. The other policies, on the other hand, take more complexity but are expected to perform better than the first policy. Consider that there are more than $m$ critical jobs. If the minimum of the remaining execution times of the dispatched $m$ critical jobs is greater than $e_{min}$, the rest of critical jobs will inevitably miss their deadlines. Hence, the second policy breaks ties in favor of shorter remaining execution time so that the minimum remaining execution time is likely to be less than $e_{min}$ and the rest of critical jobs may still have chance to meet their deadlines. The third policy is

```
1.  when any tasks are released or complete do
2.    if |γ| < m then
3.      execute all tasks in γ
4.    else
5.      e_min := min{e_i(t_s) | τ_i ∈ γ}
6.      δ := {τ_i | x_i(t_s) < e_min}
7.      if |δ| ≥ m then
8.        execute m tasks in δ
9.      else
10.       execute all tasks in δ
11.       execute m − |δ| tasks with the earliest
            deadlines in γ
12.     end if
13.   end if
14. end when
```

**Figure 2. EDCL scheduling algorithm**

encouraged by a similar idea. If ties are broken in favor of less laxity, the rest of critical jobs have more laxity. That is, there is more chance for them to meet their deadlines when the minimum remaining execution time of the selected $m$ critical jobs is less than $e_{min}$, since they have more laxity. The last policy is motivated by the known effectiveness of the EDF prioritization. The performance difference among the above tie breaking policies is evaluated through simulation studies presented in Section 5.

The procedure of EDZL is more expensive. The EDZL scheduler examines whether any job reaches zero laxity before the next job release or completion (this can be done by comparing the laxity of each task with $e_{min}$ like EDCL), since the scheduler must know the occurrence of the zero laxity to make a preemption. If the zero laxity occurs, the scheduler next computes the earliest time of its occurrence and sets a timer to preempt the current job with the zero laxity job. Meanwhile, EDCL does not require such a procedure, since the critical laxity must occur only at jobs releases and completions.

Figure 3, Figure 4, and Figure 5 show scheduling examples of EDF, EDF-US[1/2], and EDCL respectively when five periodic tasks, $\tau_1 = \tau_2 = \tau_3 = \tau_4 = (3, 10, 10)$ and $\tau_5 = (10, 15, 15)$, are submitted on two processors. As shown in the figures, $\tau_5$ misses a deadline at time 15 in EDF, and $\tau_4$ misses a deadline at time 10 in EDF-US[1/2]. On the other hand, all the five tasks are successfully scheduled by EDCL, since the priority of $\tau_5$ which would miss a deadline with the original priority is promoted to the top at time 3 due to critical laxity. Notice that the five tasks are also successfully scheduled by EDZL, but more scheduler invocations occur, compared with EDCL.

## 4 Schedulability Analysis

This section analyzes the schedulability of EDCL, using a similar approach presented in [9]. We focus on a job

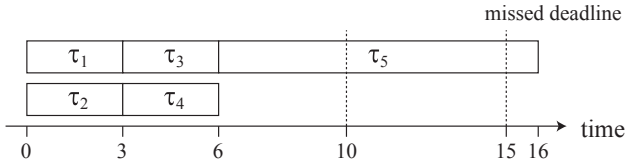


**Figure 3. EDF scheduling example**

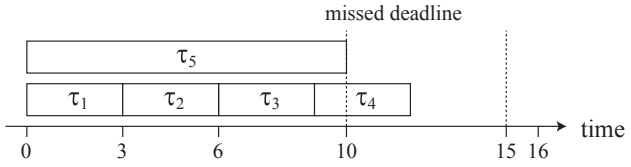


**Figure 4. EDF-US[1/2] scheduling example**

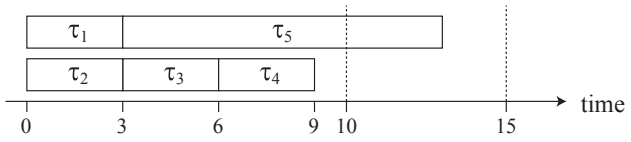


**Figure 5. EDCL scheduling example**

which first misses a deadline, and consider the conditions that are necessary for the job to miss its deadline. This job is called *problem job* henceforth. The time interval between the release time of the problem job and its missed deadline is called *problem window*. For simplicity of description, let $\tau_k$ always denote a task which contains the problem job and $t_d$ denote the deadline of the problem job.

It is clear that EDCL is a work-conserving scheduling algorithm. Hence, a job can miss its deadline only when competing jobs of other tasks given higher priorities than the job block the execution of the job for a sufficient amount of time. Using this property, the analysis **(i)** explores the lower bound on the total amount of time that must be consumed by competing jobs within the problem window to cause the problem job to miss its deadline, **(ii)** determines the upper bound on the maximum amount of time that can be contributed by each individual task, and **(iii)** examines if the sum of per-task upper bounds exceed the lower bound.

Notice that the following description is partially redundant to the schedulability analysis of EDZL presented in [9], however we repeat the description to make the analysis more comprehensible.

### 4.1 Lower Bound

We first consider the necessary conditions for the problem job to miss its deadline. EDCL gives the top priority to a critical job to meet its deadline. Thus, a critical job can be only blocked by critical jobs, so a critical job never misses its deadline unless other critical jobs occupy all the $m$ processors. In order for the problem job to miss its deadline, at least $m$ critical jobs must be scheduled at the same time.

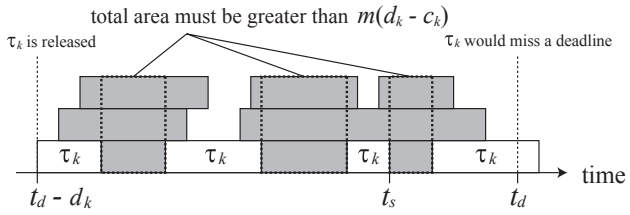


**Figure 6. Critical condition for $\tau_k$**

When $m$ critical jobs are scheduled at the same time, a necessary condition for the problem job to miss its deadline is that its laxity is less than the minimum remaining execution time of the $m$ critical jobs. Note that this condition may hold even though the problem job does not reach critical laxity, thereby we give another definition.

**Definition 2.** *A job of task $\tau_k$ is said to be **strictly-critical**, if its laxity holds the following condition at time $t_s$, where $e'_{min}$ denotes the minimum remaining execution time of the jobs with priorities higher than or equal to the job.*

$$x_k(t_s) < e'_{min} \tag{2}$$

*Then, the laxity $x_k(t_s)$ is defined **strictly-critical laxity**.*

In consequence, the necessary condition for EDCL to cause a deadline to be missed is: *m jobs become critical and another job becomes strictly-critical at the same time*. Hereinafter, all such $m+1$ jobs are deemed as problem jobs.

Now, we need to explore the condition to drive a job to be critical or strictly-critical, which is called *critical condition* in this paper. As we already described in Section 3, a job can become critical only when it *would* miss its deadline unless it is scheduled immediately at a scheduling point. A job can also become strictly-critical only in the same situation, though blocking jobs must be all critical. Figure 6 shows an example of the critical condition for a job of task $\tau_k$. Let $t_d$ denote the deadline of the job and $t_s$ denote the scheduling point at which the job becomes critical or strictly-critical. Since EDCL is work-conserving, a job is only blocked by other jobs with higher or equal priorities, except for its preceding job. In the figure, the area colored by gray indicates the executions of such jobs and the area enclosed by dots indicates their executions blocking $\tau_k$. Remember that $\tau_k$ does not actually miss a deadline if the jobs blocking after or at scheduling point $t_s$ are not all critical.

**Definition 3.** *The total amount of time that can be contributed by task $\tau_i$ in certain time interval $[a, b)$ is defined **competing work** $W_i(a, b)$.*

If $d_k \leq p_k$, it is clear from Figure 6 that the job can become critical or strictly-critical only when it would be blocked for longer than $d_k - c_k$ in the problem window if it is not scheduled immediately at scheduling point $t_s$. That is, the following condition must hold for the job of $\tau_k$ to become critical or strictly-critical.

$$\sum_{i \neq k} W_i(t_d - d_k, t_d) > m(d_k - c_k)$$

If $d_k > p_k$, the job will become critical or strictly-critical when it would be blocked for longer than $p_k - c_k$ in time interval $[t_d - p_k, t_d)$ if it is not scheduled immediately, with taking into account the constraint of its preceding job, though this expectation is pessimistic as mentioned in [9]. Hence, the job of $\tau_k$ will become critical or strictly-critical in time interval $[t_d - p_k, t_k)$, if the following condition holds.

$$\sum_{i \neq k} W_i(t_d - p_k, t_d) > m(p_k - c_k)$$

Using the same terminology in [9], time interval $[t_d - \Delta_k, t_d)$ is called *overload window* from now on, where $\Delta_k = \min\{d_k i, p_k\}$. Then, we can finalize the critical condition for $\tau_k$ in time interval $[t_d - \Delta_k, t_d)$ by Equation (3).

$$\sum_{i \neq k} W_i(t_d - \Delta_k, t_d) > m(\Delta_k - c_k) \quad (3)$$

Note that if Equation (3) holds '=', then the job never becomes critical or strictly-critical. This is a different point from the EDZL analysis in [9]. Moreover, note again that the conditions for being critical and strictly-critical are both unified by Equation (3), though the upper bounds for the left-hand side of the expression differ depending on the number of competing critical jobs. We discuss the upper bounds more specifically in the next section. Finally, we can derive the following lemma with respect to the necessary condition for EDCL scheduling to cause a deadline miss.

**Lemma 1.** *If a set of sporadic tasks is scheduled by EDCL on m processors, at least m + 1 tasks must satisfy Equation* (3) *in order to cause a deadline miss.*

*Proof.* It is trivial from the above discussion. □

## 4.2 Upper Bound

In this section, we obtain the upper bound for competing work $W_i(t_d - \Delta_k, t_d)$ of each task $\tau_i$, which contributes to driving problem task $\tau_k$ to be critical or strictly-critical in overload window $[t_d - \Delta_k, t_d)$. If the necessary condition derived in Lemma 1 does not hold even with those upper bounds, a given task set can never cause a deadline miss.

Though we assume sporadic tasks in this paper, the execution time of each task is assumed to be fixed. Thus, it is obvious that the competing work cannot be increased by outspreading the inter-release interval. As a result, the competing work of each task in any overload window is never larger than the case in which each task is periodically released at the minimum interval. As for jobs which contribute to the competing work, we only need to concern jobs with deadlines after $t_d - \Delta_k$, since we assume that the first deadline miss is caused by one of the problem jobs and no deadlines are missed before the overload window. On the other hand, jobs with deadlines in the overload window or at time $t_d$ are clearly contributors to the competing work. However, it depends on laxity whether ones with deadlines after the overload window are contributors or not. If the jobs are not critical, then they cannot be contributors. On the other hand, if they are critical, then they can be contributors. Therefore, we segregate those two cases.

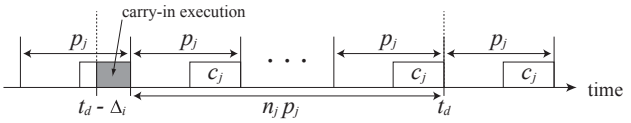


**Figure 7. Upper bound of Case 1**

**Case1**

We first consider the tasks which contain no critical jobs when the problem job becomes critical. In such a case, jobs with deadlines after the overload window can never interfere the problem job. Thus, we only need to consider the jobs with deadlines in the overload window. Note that this case exactly follows the analysis in [9]. Hence, we can refer to [9] that the maximum competing work of each task $\tau_i$ can be never greater than the case in which all the competing jobs of $\tau_i$ are executed as late as possible and a deadline of $\tau_i$ is aligned with the deadline of the problem job.

Figure 7 shows the worst-case phase of $\tau_i$, where $n_i$ is the number of jobs that have both release times and deadlines in the overload window, which can be computed as

$$n_i = \left\lfloor \frac{\Delta_k}{p_i} \right\rfloor.$$

According to [9], the length of interval during which the carried-in job can be executed is at most $\max\{0, \Delta_k - n_i p_i\}$. Therefore, the length of the carried-in execution can be never greater than

$$\min\{c_i, \max\{0, \Delta_k - n_i p_i\}\}.$$

**Lemma 2.** *If a set of sporadic tasks is scheduled by EDCL on m processors, the upper bound for the competing work of task $\tau_i$ in any overload window of the problem job of $\tau_k$, denoted by $\bar{W}_i^{\alpha}(\tau_k)$, is expressed by Equation* (4)*, if a job of $\tau_i$* ***cannot be critical*** *when the problem job becomes critical.*

$$\bar{W}_i^{\alpha}(\tau_k) = n_i c_i + \min\{c_i, \max\{\Delta_k - n_i p_i\}\} \quad (4)$$

*Proof.* It is trivial from the preceding discussion. □

**Case2**

We next consider the tasks which contain critical jobs when the problem job becomes critical. In such a case, jobs with deadlines after the overload window can affect the problem job, since the jobs are given the top priorities due to critical laxity. Thus, a deadline of $\tau_i$ is not necessarily aligned with the deadline of the problem job, and the worst-case phase of $\tau_i$ can differ from the previous case.

If a task set is scheduled by EDZL, the worst-case phase of task $\tau_i$ can be also described by Figure 7, because jobs with deadlines after the overload window can interfere the

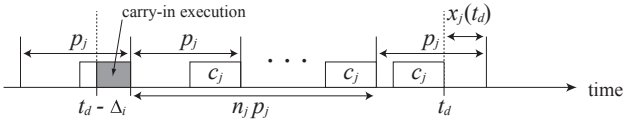


**Figure 8. Upper bound of Case 2**

problem job only when it has zero laxity and such jobs must complete at their deadlines due to zero laxity, so after all it is equivalent to considering the jobs to be executed as late as possible. However, if a task set is scheduled by EDCL, critical jobs with deadlines after the overload window can finish before their deadlines, thereby we cannot apply the same worst-case phase to EDCL scheduling. This is a main difference from the EDZL analysis.

In order to maximize the competing work, the release times of $\tau_i$ should be still periodic, then we only need to consider moving the phase of the release times in Figure 7. We argue that the competing work of $\tau_i$ is maximized when the finish time of its final job released before or in the overload window is aligned with the deadline of the problem job, as shown in Figure 8. Note that the start time of the final job must be $t_d - c_i$, since it cannot be scheduled unless it is critical due to its deadline later than $t_d$. The following discussion lead the argument to be correct.

- If we shift forward, i.e. shift later in time, the phase of $\tau_i$ by amount $a \leq p_i - x_i(t_d)$ in Figure 8, then the competing work of $\tau_i$ at the end of the overload window is decreased by $\min\{a, c_i\}$. The shift can increase the competing work at the start of the overload window, but the amount is at most $\min\{a, c_i\}$. Thus, the forward movement cannot increase the competing work of $\tau_i$.
- If we shift backward the phase of $\tau_i$ by amount $a \leq x_i(t_s)$ in Figure 8, the competing work of $\tau_i$ at the start of the overload window is decreased, while no contribution is given at the end of the overload window. Thus, the backward movement cannot also increase the competing work of $\tau_i$.

Now, we need to calculate the competing work of $\tau_i$. Since the laxity is never decreased, the final job of $\tau_i$ released before or in the overload window becomes critical when its laxity is $x_i(t_d)$. It is clear that the competing work of $\tau_i$ is increased as $x_i(t_d)$ is increased in Figure 8. We can easily obtain that the maximum value of $x_i(t_d)$ is at most

$$x_i(t_d) \leq d_i - c_i.$$

We also need to remember that the problem job is assumed not to be critical yet. Hence, the laxity of the problem job must be greater than or at least equal to that of the final job of $\tau_i$, otherwise the problem job must be also critical and it is contradict to the assumption. That is, $x_i(t_d)$ also holds

$$x_i(t_d) \leq d_k - c_k.$$

In addition, the condition for a job to be critical is that its laxity is less than the minimum remaining execution time of the jobs with higher or equal priorities. So, $x_i(t_d)$ must be within the range of

$$x_i(t_d) \leq \max\{c_j \mid \tau_j \in \tau, j \neq i\}.$$

Finally, the upper bound of $x_i(t_d)$ for any $t_d$ of $\tau_k$, denoted by $\bar{x}_i(\tau_k)$, can be expressed by Equation (5).

$$\bar{x}_i(\tau_k) = \min\{d_i - c_i, d_k - c_k, \min\{c_j \mid \tau_j \in \tau, j \neq i\}\} \quad (5)$$

Since the phase of $\tau_i$ is changed, we need to recalculate the value of $n_i$ and the length of the carry-in execution. Using $\bar{x}_i(\tau_k)$ obtained above, the maximum number of jobs of $\tau_i$ which have both release times and deadlines in the overload window can be computed as

$$n_i = \left\lfloor \frac{\Delta_k - (p_i - \bar{x}_i(\tau_k))}{p_i} \right\rfloor.$$

Then, the length of the carry-in execution is bounded to

$$\min\{c_i, \max\{\Delta_k - n_i p_i - (p_i - \bar{x}_i(\tau_k))\}\}.$$

**Lemma 3.** *If a set of sporadic tasks is scheduled by EDCL on m processors, the upper bound for the competing work of task $\tau_i$ in any overload window of the problem job of $\tau_k$, denoted by $\bar{W}_i^{\beta}(\tau_k)$, is expressed by Equation* (6)*, if a job of $\tau_i$* ***can be critical*** *when the problem job becomes critical, where $n_i' = n_i + 1$ due to limitation of space.*

$$\bar{W}_i^{\beta}(\tau_k) = n_i' c_i + \min\{c_i, \max\{0, \Delta_k - n_i' p_i + \bar{x}_i(\tau_k)\}\} \quad (6)$$

*Proof.* It is trivial from the preceding discussion. □

### 4.3 Schedulability Test

Although we obtained the upper bound for the contribution of each task $\tau_i$ to the competing work in any overload window of the problem job, for more precise analysis we need to take into account the concept of *interference* introduced by Bertogna *et al.* [6], which was also considered in [9]. The interference of $\tau_i$ on the problem job is the total length in which the problem job is blocked by jobs with higher or equal priorities and a job of $\tau_i$ is one of the $m$ jobs blocking the problem job.

According to [6], a sufficient condition to cause a deadline of $\tau_k$ to be missed is that the interference of each $\tau_i$ on the problem job of $\tau_k$ in interval $[t_d - d_k, t_d)$, which is the problem window in this paper, is at least greater than $d_k - c_k$. This upper bound is very intuitive from Figure 6. Assume that $\tau_k$ cannot be scheduled immediately at time $t_s$. Then, $\tau_k$ will miss a deadline if the plotted area is greater than $m(d_k - c_k)$. Since we assume that a job is not permitted to execute in parallel, it is sufficient for $\tau_i$ to consider only the portion of $d_k - c_k$, even though it can consume more processor time. Note that we should replace $d_k$ in [6] with $\Delta_k$ in this paper, because we permit $p_k < d_k$.

Let $\bar{W}_i(\tau_k)$ unify $\bar{W}_i^{\alpha}(\tau_k)$ and $\bar{W}_i^{\beta}(\tau_k)$ regardless of whether $\tau_i$ contain a critical job when $\tau_k$ becomes critical or not. Then, we can derive the following theorem with respect to the schedulability of EDCL.

**Theorem 3.** *A set of sporadic tasks can be successfully scheduled by EDCL on m processors, unless one of the following conditions holds for at least m + 1 tasks.*

- $\sum_{i \neq k} \min\{\bar{W}_i(\tau_k), \Delta_k - c_k\} > m(\Delta_k - c_k)$
- $\sum_{i \neq k} \min\{\bar{W}_i(\tau_k), \Delta_k - c_k\} = m(\Delta_k - c_k)$ *and* $\forall i \neq k : \Delta_k - c_k < \bar{W}_i(\tau_k)$

*Proof.* It is obvious from Equation (3) that $\tau_k$ can contain critical or strictly-critical jobs if the first condition in the theorem holds, since Equation (3) is the necessary condition for $\tau_k$ to become critical or strictly-critical.

The validity of the second condition in the theorem is then considered. If the total interference is not greater than $m(\Delta_k - c_k)$, $\tau_k$ cannot miss a deadline. However, if all the tasks have competing work greater than $\Delta_k - c_k$, then the actual total interference must be greater than $m(\Delta_k - c_k)$, even though $\sum_{i \neq k} \min\{\bar{W}_i(\tau_k), \Delta_k - c_k\} = m(\Delta_k - c_k)$ holds. In contrast, if at least one task has competing work less than or equal to $\Delta_k - c_k$, then the actual total interference is still equivalent to $m(\Delta_k - c_k)$, and $\tau_k$ cannot become critical or strictly-critical.

Consequently, $\tau_k$ can become critical or strictly-critical only when either of the above conditions holds. According to Lemma 1, there must be at least $m + 1$ tasks which can contain critical or strictly-critical jobs, if a deadline is missed. Hence, the theorem is true. □

The remaining concern is: how we determine whether each task can contain critical or strictly-critical jobs. Since the number of the tasks which can contain critical or strictly-critical jobs is needed to verify if a task can contain critical or strictly-critical jobs, a set of those tasks cannot be obtained easily. We first introduce a pessimistic idea that all tasks are assumed to contain critical jobs. From the discussion in Section 4.2, it is clear that the competing work of $\tau_i$ for the case in which the final job of $\tau_i$ released before or in the overload window of the problem job is critical is greater than that for the other case. Thus, the total competing work cannot be greater than the case in which we assume that all tasks contain critical jobs.

**Theorem 4. (Pessimistic schedulability test).** *A set of sporadic tasks can be successfully scheduled by EDCL on m processors, unless one of the following conditions holds for at least m + 1 tasks.*

- $\sum_{i \neq k} \min\{\bar{W}_i^{\beta}(\tau_k), \Delta_k - c_k\} > m(\Delta_k - c_k)$
- $\sum_{i \neq k} \min\{\bar{W}_i^{\beta}(\tau_k), \Delta_k - c_k\} = m(\Delta_k - c_k)$ *and* $\forall i \neq k : \Delta_k - c_k < \bar{W}_i^{\beta}(\tau_k)$

*Proof.* The proof obviously follows Theorem 3 and the preceding discussion. □

We next derive a tighter schedulability test, trading with more computation time. A basic idea is that we recursively search for the tasks which can contain critical or strictly-critical jobs. Hereinafter, let $\alpha$ be a set of the tasks which cannot contain critical or strictly-critical jobs and $\beta$ be a set of the tasks which can contain them. For simplicity of description, we define $\bar{W}^{\alpha}(\tau_k)$ and $\bar{W}^{\beta}(\tau_k)$ as follows.

$$\bar{W}^{\alpha}(\tau_k) = \sum_{\tau_i \in \alpha, i \neq k} \min\{\bar{W}_i^{\alpha}(\tau_k), \Delta_k - c_k\}$$

$$\bar{W}^{\beta}(\tau_k) = \sum_{\tau_i \in \beta, i \neq k} \min\{\bar{W}_i^{\beta}(\tau_k), \Delta_k - c_k\}$$

$\bar{W}^{\alpha}(\tau_k)$ is the total competing work of the tasks which contain no critical jobs when $\tau_k$ becomes critical in any overload window, and $\bar{W}^{\beta}(\tau_k)$ is that of the tasks which contain critical jobs. Using those notations, a tight schedulability test is described in Figure 9.

The tight schedulability test first assumes that no tasks become critical or strictly-critical. Under this assumption, if no tasks hold the critical condition, then no tasks will surely become critical, which means that the task set can be successfully scheduled by EDF. However, if any tasks $\{\tau_k\}$ hold the critical condition, then the test needs to verify again whether the tasks which did not hold the critical condition in the preceding test hold the critical condition. At this point, the test must assume that $\{\tau_k\}$ can become critical, since the preceding test did not assume that. Then, this procedure is repeated until one of the followings occurs.

- At least $m+1$ tasks which can be critical are found out. Then, the task set is rejected.
- New tasks are not found out to be critical, meaning that the further recursion will not find out any other tasks which can be critical. Then, the task set can be accepted.

Since the procedure is repeated at most $m+1$ times, the computation order of the schedulability test depicted in Figure 9 is $O(n^2 m)$, while that of the one in Theorem 4 is $O(n^2)$.

**Theorem 5. (Tight schedulability test).** *A set of sporadic tasks can be successfully scheduled by EDCL on m processors, if the schedulability test depicted in Figure 9 can accept the set.*

*Proof.* It is trivial from the preceding discussion. □

### Revision of EDZL Schedulability Test

Here, we report that the schedulability test of EDZL presented in [9] has a pitfall. Though this is a digression from the subject of this paper, it is very important since EDZL is now widely used and discussed.

Cirinei and Baker in [9] derived that a set of sporadic tasks can be successfully scheduled by EDZL on $m$ processors, unless the following condition holds for at least $m + 1$ tasks, and it holds strictly $>$ for at least one of them, where $\omega_i(\tau_k) = W_i^{\alpha}(\tau_k)/\Delta_k$ (the description is slightly modified).

$$\sum_{i \neq k} \min\{\omega_i(\tau_k), 1 - \lambda_k\} \geq m(1 - \lambda_k) \tag{7}$$

1. Let $\alpha = \tau$ and $\beta = \emptyset$.
2. For each $\tau_k \in \alpha$, calculate $\bar{W}^{\alpha}(\tau_k)$ and $\bar{W}^{\beta}(\tau_k)$.
3. For each $\tau_k \in \alpha$, if one of the following conditions holds, insert $\tau_k$ into $\beta$ and remove it from $\alpha$, i.e. $\beta = \beta \cup \{\tau_k\}$ and $\alpha = \alpha \setminus \{\tau_k\}$.
   - $\bar{W}^{\alpha}(\tau_k) + \bar{W}^{\beta}(\tau_k) > m(\Delta_k - c_k)$
   - $\bar{W}^{\alpha}(\tau_k) + \bar{W}^{\beta}(\tau_k) = m(\Delta_k - c_k)$ and $\forall i \neq k, \tau_i \in \alpha : \Delta_k - c_k < \bar{W}_i^{\alpha}(\tau_k)$ and $\forall i \neq k, \tau_i \in \beta : \Delta_k - c_k < \bar{W}_i^{\beta}(\tau_k)$
4. If no tasks held one of the above conditions, the test accepts the task set, and the procedure exits.
5. If the number of the tasks in $\beta$ is greater than $m$, the test rejects the task set, and the procedure exits.
6. Go back to Step 2.

**Figure 9. Tight schedulability test**

However, consider a set of $m + 1$ tasks. Then, the left-hand side of Equation (7) for each task $\tau_k$ is at most $m(1 - \lambda_k)$, since there are only $m$ tasks for the targets of the sigma function and they each can have the value at most $1 - \lambda_k$ due to the function $\min\{\omega_i(\tau_k), 1 - \lambda_k\}$. Thus, none of the $m + 1$ tasks cannot hold $>$, which means that a set of $m+1$ tasks is always accepted. However, it was reported in [16] that a set of $m + 1$ tasks can cause a deadline miss in EDZL scheduling. Hence, it is true that the above test has a pitfall. The pitfall is an ignorance of the case in which all the tasks take $1 - \lambda_k$ for $\min\{\omega_i(\tau_k), 1 - \lambda_k\}$ in Equation (7). Thus, we can revise the schedulability test of EDZL as follows.

**Theorem 6. (Revised EDZL Schedulability Test).** *A set of sporadic tasks can be successfully scheduled by EDZL on m processors, unless Equation* (7) *holds for at least* $m + 1$ *tasks, and one of the following conditions holds for at least one of them.*

- $\sum_{i \neq k} \min\{\omega_i(\tau_k), 1 - \lambda_k\} > m(1 - \lambda_k)$
- $\sum_{i \neq k} \min\{\omega_i(\tau_k), 1 - \lambda_k\} = m(1 - \lambda_k)$ *and* $\forall i \neq k : 1 - \lambda_k < \omega_i(\tau_k)$

*Proof.* The proof follows the above discussion. □

## 5 Simulation Study

In this section, simulation studies evaluate the EDCL algorithm in terms of schedulability, with comparing the traditional algorithms: EDF, EDF-US[1/2], and EDZL. The schedulability of each algorithm is estimated as follows. For certain system utilization $U_{sys}$ ($0 \leq U_{sys} \leq 1$), 100,000 task sets are randomly generated so that $U_{sys} = U(\tau)/m$ holds for all the task sets, and submitted to all the scheduling algorithms. Then, the success ratio of an algorithm is defined by the following expression.

$$\frac{\text{the number of successfully scheduled task sets}}{\text{the number of scheduled task sets (100,000)}}$$

The system utilization is varied within the range of [0.3, 1]. The schedulability of the algorithm is estimated to be high as it achieves high success ratio at high system utilization.

The successfully-scheduled task set is defined as follows. For evaluation of the guaranteed schedulability, a task set is said to be successfully scheduled, if the schedulability test accepts the task set. The schedulability test of each algorithm is implemented as follows:

- **EDF:** A given task set $\tau$ is accepted if $U(\tau) \leq m(1 - u_{max}) + u_{max}$ holds [11], where $u_{max}$ is the maximum utilization of every individual task. Even if the above condition does not hold, it is also accepted if one of the following conditions [6] holds.
  - $\sum_{i \neq k} \min\{\omega_i(\tau_k), 1 - \lambda_k\} < m(1 - \lambda_k)$
  - $\sum_{i \neq k} \min\{\omega_i(\tau_k), 1 - \lambda_k\} = m(1 - \lambda_k)$ and $\exists i \neq k : 0 < \omega_i(\tau_k) \leq 1 - \lambda_k$
- **EDF-US[1/2]:** Let $h$ be the smaller of $m - 1$ or the number of heavy tasks with utilization greater than $1/2$ in a given task set $\tau$. $\tau$ is accepted if $n - h$ light tasks are accepted by the above EDF test.
- **EDZL:** The schedulability test follows Theorem 6.
- **EDCL(P):** The schedulability test follows Theorem 4 ('P' stands for the pessimistic test).
- **EDCL(T):** The schedulability test follows Theorem 5 ('T' stands for the tight test).

For evaluation of the exhaustive schedulability, on the other hand, a task set is said to be successfully scheduled, if the task set can be actually scheduled by a scheduling algorithm without missing any deadline.

Each task set $\tau$ with $U(\tau) = U_{sys} \times m$ is generated as follows. The utilization (density) of a new task $\tau_i$ is determined based on a uniform distribution within the range of [0.1, 1.0] for fairness. Then, $\tau_i$ is appended to $\tau$ as long as $U(\tau) \leq U_{sys} \times m$. When $U(\tau)$ exceeds $U_{sys} \times m$, the utilization of $\tau_i$ is adjusted so that $U(\tau) = U_{sys} \times m$ holds. The period of $\tau_i$ is randomly determined within the range of [1000, 100000], assuming real-time applications with periods ranging from 1ms to 100ms. The deadline is equal to the period, and the execution time is computed as $c_i = \lambda_i p_i$. The length of each simulation is $\min\{\text{lcm}(\{p_i \mid \tau_i \in \tau\}), 2^{32}\}$.

Figure 10~12 show the results for the success ratio of each algorithm, which are verified by each schedulability test. EDCL outperforms EDF substantially, even though the test is pessimistic. While the success ratio of the EDF test drops below 100% when the system utilization exceeds 40%, that of the pessimistic EDCL test is retained 100% until the system utilization reaches 60 ~ 62%. However, the pessimistic EDCL test can be inferior to the EDF-US[1/2]

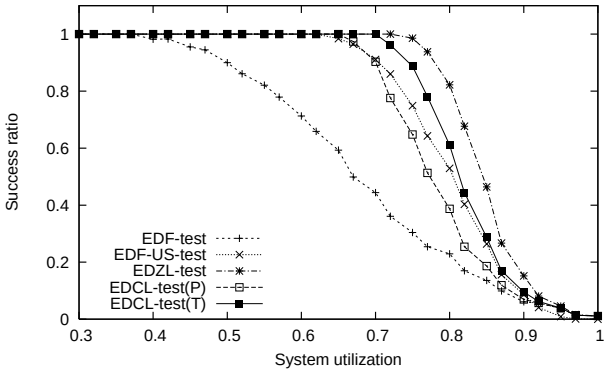


**Figure 10. Guaranteed success ratio ($m = 4$).**

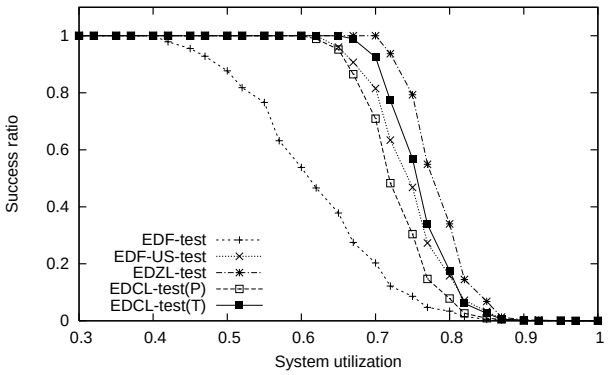


**Figure 11. Guaranteed success ratio ($m = 8$).**

test, especially after the success ratio drops below 100%. The tight EDCL test, meanwhile, always outperforms the EDF-US[1/2] test by $3 \sim 8\%$ for the system utilization in which the success ratio can be retained 100%. The EDZL test achieves the highest success ratio as expected, which is about 3% surperior to the tight EDCL test.

According to the simulations, the algorithms except for EDF are relatively competitive, with respect to the guaranteed schedulability. Though EDF-US[1/2] performs competitively with EDCL and EDZL, its exhaustive performance is clearly inferior to them, as shown later. EDZL is always better than EDCL. It is obvious that the performance difference between EDCL and EDZL are attributed by their different rules for priority promotions. The difference can be shrunk by using the tight schedulability test for EDCL, but EDCL can never outperform EDZL in term of schedulability. Instead, EDCL has a smaller bound on the number of scheduler invocations and can be more easily implemented. In that sense, EDCL and EDZL have relative merits to each other with competitive performance.

Figure 13~15 show the results for the success ratio of each algorithm, which are verified by each exhaustive scheduling (tasks are actually scheduled until a job misses its deadline or the simulation exits). The suffixes of '(R)', '(L)', and '(D)' for EDCL stand for the policies of tie breaking respectively: ties are broken in favor of shorter Remaining execution time, less Laxity, and earlier Deadline. Note that ties are broken arbitrarily for EDCL with no suffix.

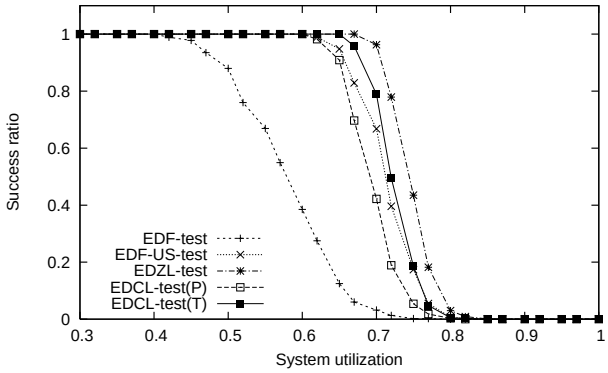


**Figure 12. Guaranteed success ratio ($m = 16$).**

According to the simulations, EDCL and EDZL perform far beyond EDF. As reported in [9], EDZL can successfully schedule almost all the task sets, even though the system utilization is 100%. EDCL can also perform competitively with EDZL, especially when ties are broken in favor of less laxity. Thus, we can observe the effectiveness of breaking ties. Though EDZL is slightly better than EDCL, the EDCL scheduler needs to be invoked only when jobs are released or complete. As a result, it can be said again that EDCL and EDZL are competitive even for the exhaustive schedulability as well as the guaranteed one. EDF-US[1/2] is a middle-grounder, however we need to remember that it can perform worse than EDF depending on given task sets.

## 6 Conclusion

This paper presented the EDCL algorithm, which is a derivative of EDZL, for the efficient scheduling of sporadic real-time tasks on multiprocessor systems. We proved that **(i)** EDCL is at least as effective as EDF, and **(ii)** the number of scheduler invocations per job release is at most 2. EDZL also has the former property but does not have the latter one. In addition, since the scheduling points are made only at job releases and completions, the implementation complexity of EDCL is more relaxed than EDZL. We then designed pessimistic and tight schedulability tests of EDCL. We also corrected an error in the traditional schedulability test of EDZL. The simulation studies showed that EDCL can be competitive with EDZL far beyond EDF, with respect to both guaranteed schedulability and exhaustive schedulability. In conclusion, EDCL is a novel real-time scheduling algorithm for multiprocessor systems.

In the end, the insights into the future work are described. Though the schedulability test of EDCL was derived in this paper, the utilization bound was not explored. In fact, there is a task set that cannot be successfully scheduled by EDCL if the total utilization exceeds $(m + 1)/2$. Since jobs can be preempted only at job releases and completions, a set of $m + 1$ tasks with all execution times $x + \epsilon$ and deadlines $2x$

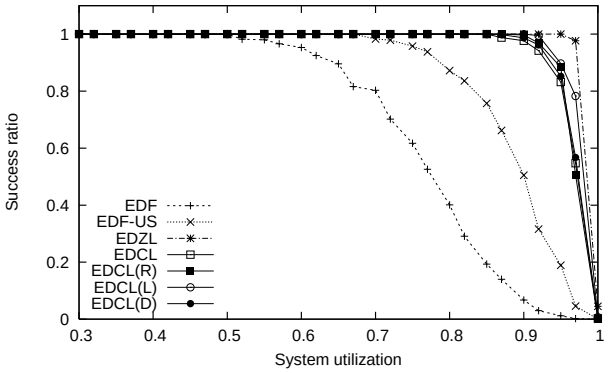


**Figure 13. Exhaustive success ratio ($m = 4$).**

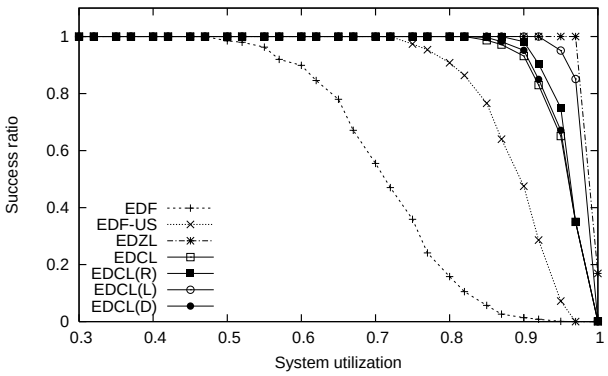


**Figure 14. Exhaustive success ratio ($m = 8$).**

are not schedulable. The argument is true if $\epsilon \to 0$. Thus, the tightness of this utilization bound will be analyzed.

The superiority of EDCL over EDF-US[$x$] will be also considered. In this paper, it was demonstrated that EDCL is at least as effective as EDF. We wonder if EDCL also strictly dominates EDF-US[$x$].

According to the simulations results, EDCL has a meaningful gap between guaranteed schedulability and exhaustive schedulability. Thus, the theoretical analysis of EDCL will be moreover conducted. We will also examine if EDCL is predictable [12] and sustainable [3].

## References


[1] J. Anderson and A. Srinivasan. Early-Release Fair Scheduling. In *Proc. of the Euromicro Conference on Real-Time Systems*, pages 35–43, 2000.

[2] T.P. Baker. An Analysis of EDF Schedulability on a Multiprocessor. *IEEE Trans. on Parallel and Distributed Systems*, 16:760–768, 2005.

[3] S. Baruah and A. Burns. Sustainable Scheduling Analysis. In *Proc. of the IEEE Real-Time Systems Symposium*, pages 159–168, 2006.

[4] S. Baruah, N. Cohen, C.G. Plaxton, and D. Varvel. Proportionate Progress: A Notion of Fairness in Resource Allocation. *Algorithmica*, 15:600–625, 1996.


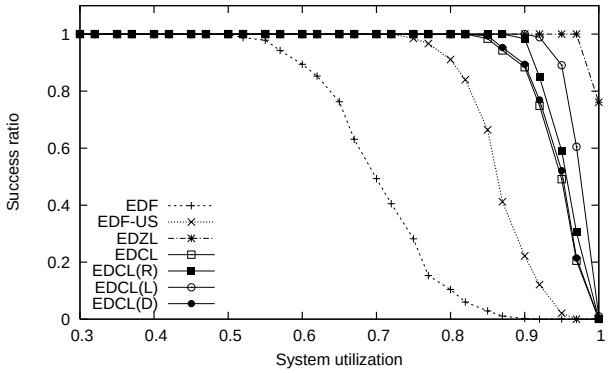


**Figure 15. Exhaustive success ratio ($m = 16$).**


[5] S. Baruah, J. Gehrke, and C.G. Plaxton. Fast Scheduling of Periodic Tasks on Multiple Resources. In *Proc. of the International Parallel Processing Symposium*, pages 280–288, 1995.

[6] M. Bertogna, M. Cirinei, and G. Lipari. Improved Schedulability Analysis of EDF on Multiprocessor Platforms. In *Proc. of the Euromicro Conference on Real-Time Systems*, pages 209–218, 2005.

[7] H. Cho, B. Ravindran, and E. Jensen. An Optimal Real-Time Scheduling Algorithm for Multiprocessors. In *Proc. of the IEEE Real-Time Systems Symposium*, pages 101–110, 2006.

[8] S. Cho, S.K. Lee, A. Han, and K.J. Lin. Efficient Real-Time Scheduling Algorithms for Multiprocessor Systems. *IEICE Trans. on Communications*, E85-B(12):2859–2867, 2002.

[9] M. Cirinei and T.P. Baker. EDZL Scheduling Analysis. In *Proc. of the Euromicro Conference on Real-Time Systems*, pages 9–18, 2007.

[10] S. K. Dhall and C. L. Liu. On a Real-Time Scheduling Problem. *Operations Research*, 26:127–140, 1978.

[11] J. Goossens, S. Funk, and S. Baruah. Priority-driven Scheduling of Periodic Task Systems on Multiprocessors. *Real-Time Systems*, 25:187–205, 2003.

[12] R. Ha and J. Liu. Validating Timing Constraints in Multiprocessor and Distributed Real-Time Systems. In *Proc. of the IEEE International Conference on Distributed Computing Systems*, pages 162–171, 1994.

[13] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *Journal of the ACM*, 20:46–61, 1973.

[14] X. Piao, S. Han, H. Kim, M. Park, Y. Cho, and S. Cho. Predictability of Earliest Deadline Zero Laxity Algorithm for Multiprocessor Real-Time Systems. In *Proc. of the IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing*, pages 359–364, 2006.

[15] A. Srinivasan and S.K. Baruah. Deadline-based Scheduling of Peroidic Task Systems on Multiprocessors. *Information Processing Letters*, 84(2):93–98, 2002.

[16] H.W. Wei, Y.H. Chao, S.S. Lin, K.J. Lin, and W.K. Shih. Current Results on EDZL Scheduling for Multiprocessor Real-Time Systems. In *Proc. of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 120–130, 2007.