applied sciences

MDPI

Article

A Comparative Study on the Schedulability of the EDZL Scheduling Algorithm on Multiprocessors

Sangchul Han, Woojin Paik, Myeong-Cheol Ko and Minkyu Park *

Department of Computer Engineering, Konkuk University, Chungju 27478, Republic of Korea; schan@kku.ac.kr (S.H.); wjpaik@kku.ac.kr (W.P.); cheol@kku.ac.kr (M.-C.K.)
* Correspondence: minkyup@kku.ac.kr

Abstract: As multiprocessor (or multicore) real-time systems become popular, there has been much research on multiprocessor real-time scheduling algorithms. This work evaluates EDZL (Earliest Deadline until Zero Laxity), a scheduling algorithm for real-time multiprocessor systems. First, we compare the performance of EDZL schedulability tests. We measure and compare the ratio of task sets admitted by each test. We also investigate the dominance between EDZL schedulability tests and discover that the union of the demand-based test and the utilization-based test is an effective combination. Second, we compare the schedulability of EDZL and EDF$^{(k)}$. We prove that the union of the EDZL schedulability tests dominates the EDF$^{(k)}$ schedulability test, i.e., the union of the EDZL schedulability tests can admit all task sets admitted by the EDF$^{(k)}$ schedulability test. We also compare the schedulability of EDZL and EDF$^{(k)}$ through scheduling simulation by measuring the ratio of successfully scheduled task sets. EDZL can successfully schedule 7.0% more task sets than EDF$^{(k)}$.

Keywords: multiprocessor; real-time; schedulability; EDZL; EDF$^{(k)}$; dominance

Citation: Han, S.; Paik, W.; Ko, M.-C.; Park, M. A Comparative Study on the Schedulability of the EDZL Scheduling Algorithm on Multiprocessors. *Appl. Sci.* **2023**, *13*, 10131. https://doi.org/10.3390/app131810131

Academic Editors: Hongming Shan and Ruibin Feng

Received: 10 July 2023
Revised: 25 August 2023
Accepted: 5 September 2023
Published: 8 September 2023

1. Introduction

As multiprocessor systems running real-time applications become popular, there has been much research on algorithms that can correctly schedule real-time tasks with timing constraints as well as significantly utilize multiple processors (or multicores) [1–6]. Some of them are optimal, i.e., they can fully utilize multiple processors, but they suffer from scheduling overhead such as task splitting and frequent preemption. EDZL (Earliest Deadline until Zero Laxity) [7,8] is a practical real-time scheduling algorithm for multiprocessor systems. It is not optimal but it can achieve a high processor utilization with restricted scheduling overhead. Many researchers have extended or applied the EDZL algorithm to various applications. Alhussian and Zakaria [9] proposed a variant of EDZL, the LEZL (Largest Execution until Zero Laxity) scheduling algorithm. Choi et al. [10] proposed the EVDZL (Earliest Virtual Deadline Zero Laxity) algorithm that applies EDZL to mobile GPU scheduling. There are other research works that apply EDZL scheduling algorithm to virtual cloud environments, mixed-criticality systems or large-scale real-time cloud systems [11–14]. In reference [15], the EDZL algorithm was integrated with a task-level re-execution-based fault tolerance technique.

To employ a real-time scheduling algorithm in real systems, a schedulability test for the algorithm is undoubtedly required. A schedulability test is a method that examines in advance whether a task set will be correctly scheduled by the scheduling algorithm or not. We can guarantee the satisfaction of the timing constraints (deadlines) of real-time tasks to implement a safe real-time system only if a given task set is admitted by one of the schedulability tests for the scheduling algorithm. If a task set that does not pass any schedulability test was executed, it might violate its timing constraints, i.e., miss a deadline. A deadline miss in hard real-time systems may cause a system failure or catastrophe. We do not perform a task set on a hard real-time system if the task set does not pass any

schedulability test. Note that there can be more than one schedulability test for a scheduling algorithm. Many researchers try to develop a tighter schedulability test because a tighter schedulability test can accommodate more tasks and admit more task sets.

There have been several studies on the schedulability test for the EDZL algorithm. Piao et al. [16] proposed the first EDZL schedulability test, which is based on a schedulable utilization bound. Piao's test is very simple and too pessimistic. It cannot guarantee the successful execution of task sets whose total utilization is higher than $(m+1)/2$ where m is the number of processors. Wei et al. [17] presented a tighter test that can be applied to two processor systems. Baker et al. [18] proposed a more elaborate, slack-based test. By computing the slack of tasks iteratively, it provides a much tighter condition than Piao's test. Baker's slack-based test outperforms Piao's test in terms of the number of admitted task sets. However, sometimes the slack-based test does not admit task sets with low total utilization that are admitted even by Piao's test. Lee and Shin [19] proposed a simple, but effective, utilization-based test. The utilization test has some advantages over the slack-based test. First, its implementation is simple and the computation complexity is lower than the slack-based test. Second, it outperforms the slack-based test in terms of the number of admitted task sets. Third, it dominates Piao's test, as we will prove in Section 4.1. Additionally, the authors showed that the utilization-based test and the slack-based test do not dominate each other. Later, Lee and Shin also proposed a demand-based test [20]. They developed a demand bound function for job-level dynamic priority scheduling and proposed a complicated but very tight EDZL schedulability test. The demand-based test outperforms other EDZL schedulability tests in terms of the number of admitted task sets. They also showed that the demand-based test is not dominated by other tests.

The above research evaluated the EDZL schedulability tests from a quantitative perspective. The EDZL schedulability tests are compared in terms of the number of admitted task sets. But, these tests are not fully evaluated from a qualitative perspective. Whether the demand-based test dominates other tests is not revealed. The dominance between Piao's test and other tests is not disclosed, either. On the other hand, the EDZL schedulability tests have not been compared with other scheduling algorithms' schedulability tests, which are also practical and have comparable performance.

This paper evaluates the schedulability of the EDZL algorithm. First, we compare the performance of the EDZL schedulability tests from a quantitative and qualitative perspective. We generate all possible task sets in a restricted range and analyze them with the above-mentioned schedulability tests. We measure and compare the ratio of task sets admitted by each test. As a result, the demand-based schedulability test outperforms other tests, similarly to the experimental results in reference [20]. We also investigate the dominance between the EDZL schedulability tests. We prove that the utilization-based test dominates Piao's test. And we discover that the slack-based test, the utilization-based test and the demand-based test do not dominate each other. The experimental results implies that the union of the demand-based test and the utilization-based test is an effective combination. Second, we compare the schedulability of EDZL with EDF$^{(k)}$ [21], which is another practical multiprocessor real-time scheduling algorithm. We find out that the utilization-based EDZL schedulability test is equivalent to the EDF$^{(k)}$ schedulability test. This implies that the union of the EDZL schedulability tests dominates the EDF$^{(k)}$ schedulability test, i.e., the EDZL schedulability test can admit all task sets admitted by the EDF$^{(k)}$ schedulability test. We also compare the schedulability of EDZL and EDF$^{(k)}$ through scheduling simulation. We measure and compare the ratio of successfully scheduled task sets. EDZL can successfully schedule 7.0% more instances than EDF$^{(k)}$.

The contributions of this work are as follows.

- The dominance between the EDZL schedulability tests is unveiled. The utilization-based test dominates Piao's test. The slack-based, utilization-based, and demand-based tests do not dominate each other.
- This work proves that the utilization-based EDZL schedulability test is equivalent to the EDF$^{(k)}$ schedulability test. The two tests dominate each other.

- A EDZL schedulability test that dominates the EDF$^{(k)}$ schedulability test can be easily constructed by unioning the EDZL schedulability tests. The union of the EDZL schedulability tests admits more task sets than the EDF$^{(k)}$ schedulability test.
- The performance of EDZL and EDF$^{(k)}$ scheduling was compared in terms of the number of successfully scheduled task sets. We found out that EDZL can accommodate more task sets than EDF$^{(k)}$.

This paper is organized as follows. Section 2 provides the background knowledge about the EDZL and EDF$^{(k)}$ schedulability tests. Section 3 compares the performance of the EDZL schedulability tests. Section 4 compares the performance of EDZL and EDF$^{(k)}$ scheduling. Finally, Section 5 concludes our work.

2. Background

2.1. System Model

We consider the preemptive scheduling of periodic task sets $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ with unrestricted task migration on m identical processors (or cores). Task τ_i is a periodic task denoted by $\tau_i = (c_i, p_i)$, where c_i is an execution time and p_i is a period. Task τ_i periodically releases a job $\tau_{i,j}$ at time $(j-1)p_i$ and the job requires the execution of, at most, c_i by its deadline $d_{i,j}(= jp_i)$ for $j = 1, 2, \ldots$ The utilization of τ_i is defined as $u_i = c_i/p_i$. The total utilization of τ is $U(\tau) = \sum u_i$. The tasks are assumed to be indexed in the order of non-increasing utilization, i.e., $u_i \geq u_{i+1}$. A task set $\tau^{(i)}$ is defined as a subset of τ, consisting of $(n - i + 1)$ tasks of lower utilization, that is, $\tau^{(i)} = \{\tau_i, \tau_{i+1}, \ldots, \tau_n\}$. The amount of remaining execution of job $\tau_{i,j}$ at time t is $r_{i,j}(t)$. Laxity is the amount of time for which a job can idle safely without missing its deadline. We denote the laxity of $\tau_{i,j}$ at time t by $lx_{i,j}(t) = d_{i,j} - t - r_{i,j}(t)$. The notations are summarized in Table 1.

Table 1. Notations.

Notation	Description
m	the number of processors
n	the number of tasks
τ	periodic task set
τ_i	the i^{th} task
$\tau_{i,j}$	the j^{th} job of τ_i
c_i	the execution time of τ_i
p_i	the period of τ_i
$d_{i,j}$	the deadline of $\tau_{i,j}$
u_i	the utilization of τ_i
$U(\tau)$	the total utilization of task set τ
$\tau^{(i)}$	$\tau^{(i)} = \{\tau_i, \tau_{i+1}, \ldots, \tau_n\}$
$r_{i,j}(t)$	the remaining execution time of $\tau_{i,j}$ at time t
$lx_{i,j}(t)$	the laxity of $\tau_{i,j}$ at time t

We assume preemptive scheduling, i.e., any job can be preempted by a higher priority job at any time. We also assume that task migration is allowed, i.e., a preempted job can be resumed later on any available processor and the cost is negligible. A job cannot be executed on more than one processor at the same time and each processor cannot execute more than one job at the same time.

2.2. Real-Time Scheduling on Multiprocessors

Real-time scheduling approaches on multiprocessors can be categorized, in terms of task migration, as partitioned and global. In the partitioned approach, tasks are divided into partitions and each partition is associated with a processor. Tasks cannot migrate between partitions (processors), i.e., a task is assigned to a processor statically and all jobs released by the task should execute on the processor. Each processor schedules a partition with one of uniprocessor scheduling algorithms such as RM (Rate Monotonic) [22], EDF

(Earliest Deadline First) [22], and DM (Deadline Monotonic) [23]. The merit of partitioned scheduling is that well-studied uniprocessor scheduling analysis can be applied for each processor. However, it is difficult to partition tasks optimally and achieve a high processor utilization [24]. In the global approach, tasks are fully migratable. All jobs can start or resume execution on any available processor. The merit of the global approach is that tasks are automatically load-balanced and no task partitioning is required. Uniprocessor scheduling algorithms can be adopted in the global scheduling approach [21,25,26].

EDF has strong points that it can be implemented efficiently [27] and the numbers of preemption and migration can be bounded [28]. Most of all, EDF is optimal on uniprocessor systems. However, EDF is not optimal on multiprocessors. Simply employing an optimal uniprocessor scheduling algorithm may not achieve high processor utilization in multiprocessor systems. EDF may fail to successfully schedule task sets with low total utilization on multiprocessors, suffering from Dhall's effect [29]. A high-utilization task with a long deadline (its laxity must be small) is apt to be interrupted by short deadline tasks and miss its deadline (due to its small laxity).

One of the ways to overcome Dhall's effect is assigning high priority to high-utilization tasks to give them more chance to execute. EDF-variant algorithms such as EDF-US$[m/(2m-1)]$, fpEDF, and EDF$^{(k)}$ were developed to adopt this policy. EDF-US$[m/(2m-1)]$ [30] assigns the highest priority to tasks with utilization greater than $m/(2m-1)$ and assigns a normal priority to the remaining tasks based on EDF. EDF-US$[m/(2m-1)]$ can successfully schedule any task set whose total utilization is less than or equal to $m^2/(2m-1)$. fpEDF [31] gives the highest priority to, at most, $(m-1)$ largest-utilization tasks with a utilization greater than $1/2$. The remaining tasks are also given a normal priority based on EDF. fpEDF can successfully schedule any task set whose total utilization is less than or equal to $(m+1)/2$. EDF$^{(k)}$ gives the highest priority to $(k-1)$ largest-utilization tasks and the remaining tasks are given a normal priority based on EDF. The schedulability of EDF$^{(k)}$ is discussed in Section 2.4. Since EDF$^{(k)}$ dominates EDF-US$[m/(2m-1)]$ and fpEDF, as we will show in Section 2.4, it is sufficient to consider the schedulability of EDF$^{(k)}$.

Another approach against Dhall's effect is raising the priority of urgent jobs dynamically. Laxity can be a metric for urgency. Small laxity indicates that there is little time to idle or wait for execution. If the laxity of a job is zero, the job should be executed immediately. Otherwise, it would miss its deadline. A ZL (Zero Laxity) policy raises the priority of a job if its laxity becomes zero, so that it can be executed immediately. ZL policy can improve the performance of fixed priority scheduling algorithms dramatically. ZL policy has been analyzed and adapted to many real-time scheduling schemes [12,32–36]. EDZL [7] is a global scheduling algorithm for multiprocessor real-time systems that firstly employs ZL policy. In essence, EDZL gives priority to a job based on EDF. If a job's laxity becomes zero, ZL policy changes the priority of the job to the highest one. Note that the laxity is a non-increasing function. Once a job is given the highest priority, it retains the priority until it completes execution. The schedulability of EDZL is discussed in detail in Section 2.3.

The first optimal scheduling algorithm in the global approach is Pfair [1]. Pfair splits every job into sub-jobs and assigns a pseudo release time and deadline for the sub-jobs such that every job can progress at the same proportion. Pfair can successfully schedule any task set whose total utilization is less than or equal to m. The drawbacks of Pfair is the very high scheduling overhead. The number of context switches (or preemptions) and migrations may be very high. To mitigate these drawbacks, other optimal algorithms have been developed, like ERfair [37], EKG [2], LLREF [3], DP-Wrap [38], U-EDF [39], and RUN [40]. Some of them employ a semi-partitioned approach where some tasks are allowed to migrate but some are not. However, these algorithms require complex computation for scheduling decisions and still incur more preemptions and migrations than EDZL and EDF$^{(k)}$.

In this work, we compare the schedulability of EDZL and EDF$^{(k)}$. EDZL and EDF$^{(k)}$ are not optimal, but we choose these algorithms because they are practical in the sense

that the scheduler implementation cost is low, the scheduling overhead is bounded, and they can achieve high processor utilization. They do not require task partitioning, job splitting, frequent priority change or complex scheduling decisions. The implementation of the schedulers is relatively simple. In addition, since the priority is given at job release or the zero-laxity point (in case of EDZL), the number of preemptions in a time interval is bounded by the number of jobs or two times the number of jobs (in case of EDZL) released in the interval. The number of migrations is bounded by the number of preemptions.

2.3. EDZL Schedulability Test

EDZL employs a ZL policy. EDZL gives priority to a job based on EDF. If a job's laxity becomes zero, ZL policy changes the priority of the job to the highest one. There are several studies on the EDZL schedulability test. In reference [8], the authors showed that EDZL dominates EDF. This implies that any sufficient schedulability test for EDF scheduling can be used for EDZL scheduling. Based on this, Piao et al. [16] proposed a schedulable utilization bound-based schedulability test. They proved that any task set satisfying $U(\tau) \leq (m+1)/2$ is schedulable by EDZL. This is the first EDZL schedulability test. Later, Wei et al. [17] presented a tighter test for $m = 2$.

Baker et al. [18] proposed a slack-based test. By computing the slack of tasks iteratively, it checks whether $(n - m)$ tasks have a positive slack lower bound, i.e., whether no more than m tasks may have zero laxity at the same time. Algorithm 1 is the pseudo-code for the iterative slack-based EDZL schedulability test, where

$$n_i^*(s_i^{lb}) = \left\lfloor \frac{max(0, p_k - s_i^{lb})}{p_i} \right\rfloor \tag{1}$$

and

$$\epsilon_i(s_i^{lb}) = min(e_i, max(0, p_k - s_i^{lb}) - n_i^*(s_i^{lb})p_i). \tag{2}$$

Baker's slack-based test outperforms Piao's test substantially in terms of the number of admitted task sets. But they do not dominate each other. That is, there exists a task set that can be admitted by Piao's test but not by the slack-based test, and vice versa. In addition, the slack-based test shows a higher time complexity than Piao's test.

Algorithm 1 Iterative slack-based EDZL schedulability test [18]

$s^{lb} \leftarrow (0, \ldots, 0)$
repeat
  $converged \leftarrow True$
  $infeasible \leftarrow 0$
  for $k \in \{1, \ldots, n\}$ **do**
    $newSlack \leftarrow p_k - e_k - \frac{1}{m} \sum_{i \neq k} min(n_i^*(s_i^{lb}) \cdot e_i + \epsilon_i(s_i^{lb}), p_k - e_k)$
    if $s_k^{lb} < newSlack$ **then**
      $s_k^{lb} \leftarrow newSlack$
      $converged \leftarrow False$
    endif
    if $s_k^{lb} \leq 0$ **then**
      $infeasible \leftarrow infeasible + 1$
    endif
  done
until $converged$ **or** $infeasible \leq m$
return $infeasible \leq m$

Lee and Shin [19] proposed a utilization-based test (Theorem 1). It leverages the EDF schedulability test proposed in reference [21] and the fact that EDZL dominates EDF [8].

The merit of this test is simplicity. Since Equation (3) includes only task utilization values, the test performs in a lower time complexity than the slack-based test and the demand-based test. Moreover, the authors showed that the utilization-based test can admit much more task sets than Baker's slack-based test. It is known that the utilization-based test and the slack-based test do not dominate each other [19]. However, the dominance between the utilization-based test and Piao's test has not been disclosed.

Theorem 1 (Theorem 2 in reference [19]). *On m identical processors, task set τ will be correctly scheduled by EDZL if there exists $m' = 1, 2, \ldots, m$ such that*

$$\sum_{\tau_i \in T_1} u_i \leq m' - (m' - 1) \cdot \max_{\tau_j \in T_1} u_j \tag{3}$$

where

$$T_1 = \{\tau_i \in T \mid \tau_i \notin m - m' \text{ tasks with the largest } u_i\}.$$

Later, Lee and Shin derived a demand-bound function for EDZL and developed a demand-based test [20]. They also presented the demand-based test for LLF (Least Laxity First) scheduling [41]. These are the first demand-based tests for job-level dynamic priority scheduling. They demonstrated that the demand-based test for EDZL outperforms the slack-based test and the utilization-based test through simulation. However, the dominance between them was not fully discovered. The authors showed that the demand-based test is not dominated by other tests. However, whether the demand-based test dominates other tests is not disclosed. Theorem 2 shows the test, where the demand bound function of τ_i in an interval of length l is defined as follows.

$$DBF_{EDZL}(\tau_i, l) = \left(\left\lfloor \frac{l - p_i}{p_i} \right\rfloor + 1\right) \cdot e_i + \max\left(0, l - \left(\left\lfloor \frac{l - p_i}{p_i} \right\rfloor + 1\right) \cdot p_i - (p_i - e_i)\right) \tag{4}$$

and

$$DBF'_{EDZL}(\tau_i, l) = \left\lfloor \frac{1}{p_i} \right\rfloor \cdot e_i + \min(e_i,\ l \mod p_i). \tag{5}$$

Theorem 2 (Theorem 2 in reference [20]). *A task set τ is schedulable under EDZL on an m-core platform if at least $|\tau| - m$ tasks ($\tau_k \in \tau$) satisfy the following condition for all $l_k \geq 0$:*

$$\sum_{\tau_i \in \tau} \hat{I}_{EDZL}(\tau_i, l_k) + \sum_{m-1 \text{ largest } \tau_i \in \tau} \{\hat{I}'_{EDZL}(\tau_i, l_k) - \hat{I}_{EDZL}(\tau_i, l_k)\} < m \cdot (l_k + D_k - C_k) \tag{6}$$

where

$$\hat{I}_{EDZL}(\tau_i, l) = \begin{cases} \min(DBF_{EDZL}(\tau_i, l + p_k), l + p_k - e_k) & \text{if } \tau_i \neq \tau_k \\ \min(DBF_{EDZL}(\tau_i, l + p_k) - e_k, l) & \text{if } \tau_i = \tau_k \end{cases}$$

and

$$\hat{I}'_{EDZL}(\tau_i, l) = \begin{cases} \min(DBF'_{EDZL}(\tau_i, l + p_k), l + p_k - e_k) & \text{if } \tau_i \neq \tau_k \\ \min(DBF'_{EDZL}(\tau_i, l + p_k) - e_k, l) & \text{if } \tau_i = \tau_k \end{cases}$$

2.4. EDF$^{(k)}$ Schedulability Test

EDF$^{(k)}$ gives the highest priority to the jobs of $(k-1)$ largest-utilization tasks. And the jobs of the remaining tasks are given a priority according to EDF policy. This algorithm is simple but effective. We can show that EDF$^{(k)}$ dominates EDF-US$[m/(2m-1)]$ and fpEDF. EDF$^{(k)}$ can generates EDF-US$[m/(2m-1)]$ schedule or fpEDF schedule by controlling k. For a given task set, EDF$^{(k)}$ can generate the same schedule as EDF-US$[m/(2m-1)]$ by setting k such that $u_{k-1} > m/(2m-1)$ and $u_k \leq m/(2m-1)$. Similarly, EDF$^{(k)}$ can generate the same schedule as fpEDF by setting k such that $u_{k-1} > 1/2$ and $k \leq m-1$.

Therefore, any task set schedulable by EDF-US[$m/(2m-1)$] or fpEDF is also schedulable by EDF$^{(k)}$. Lemmas 1 and 2 follow.

Lemma 1. *EDF$^{(k)}$ dominates EDF-US[$m/(2m-1)$].*

Lemma 2. *EDF$^{(k)}$ dominates fpEDF.*

In reference [21], the authors proposed a scheme that calculates a minimum number of required processors to successfully schedule a given task set (Theorem 3). Based on this, we can give a schedulability test for EDF$^{(k)}$, as shown in Corollary 1.

Theorem 3 (Theorem 8 in [21])**.** *On m identical processors, task set τ will be correctly scheduled by EDF$^{(k)}$ where*

$$m = (k-1) + \left\lceil \frac{U(\tau^{(k+1)})}{1-u_k} \right\rceil \tag{7}$$

Corollary 1. *On m identical processors, task set τ will be correctly scheduled by EDF$^{(k)}$ if there exists $k = 1, 2, \ldots, m$ such that*

$$m \geq (k-1) + \left\lceil \frac{U(\tau^{(k+1)})}{1-u_k} \right\rceil \tag{8}$$

2.5. Dominance

Dominance is one of the ways to compare scheduling algorithms in terms of schedulable task sets. For scheduling algorithms A and B, let T_A and T_B be the sets of all task sets that are successfully scheduled by A and B, respectively. A is said to *dominate* B if $T_A \supseteq T_B$, that is, A can successfully schedule all task sets that are schedulable by B. Park et al. [8] showed that EDZL strictly dominates EDF. They proved that EDZL can correctly schedule any task set that is schedulable by EDF and showed there is at least one task set schedulable by EDZL but not schedulable by EDF.

On the other hand, dominance can be used to compare schedulability tests. For schedulability tests P and Q, let T_P and T_Q be the sets of all task sets admitted by P and Q, respectively. P is said to *dominate* Q if $T_P \supseteq T_Q$, that is, P can admit all task sets that are admitted by Q. We study the dominance between the EDZL schedulability tests. It is known that the slack-based test and the utilization-based test do not dominate each other [19]. We investigate the dominance between the demand-based test and the other tests in Section 3.2. We also discover the dominance between the EDZL schedulability test and EDF$^{(k)}$ schedulability tests in Section 4.1.

3. Comparing EDZL Schedulability Tests

A schedulability test is a method to verify the schedulability of a task set prior to actual scheduling. If a schedulability test is too pessimistic, it can admit a small number of task sets and the processors may not be utilized enough. Hence, a schedulability test needs to admit as many task sets as possible.

We evaluate and compare three EDZL schedulability tests—the slack-based test, the utilization-based test, and the demand-based test—in terms of the number of task sets admitted by each test. For evaluation, we generate all possible task sets in a restricted range of task parameters. The parameters are the number of tasks in a task set, the period of each task, and the execution time of each task in the domain of integers. The number of tasks in a task set is varied from 3 to 6, i.e., $n = 3, 4, 5, 6$. The period of each task is varied from 2 to 13, i.e., $p_i = 2, 3, \ldots, 13$. And the execution time is varied from 1 to ($period - 1$), i.e., $c_i = 1, 2, \ldots, p_i - 1$. The total number of task sets generated in this range is 406,478,384.

Each task set was tested with the three EDZL schedulability tests, varying the number of processors from 2 to $(n-1)$. We excluded test instances (test instance: the combination

of task set and the number of processors) whose task set utilization exceeds the number of processors. The total number of test instances is 1,000,752,406.

3.1. Admission Ratio

The task set utilization of test instances falls on $(0, m]$, where m is the number of processors. We divide this range into $m \times 100$ intervals of length 0.01. For each interval, all test instances whose task set utilization is within the interval are grouped. Let num_{grp} be the number of test instances belonging to group grp. For each group, we count the number of test instances admitted by each test, denoted by $admit_{grp,test}$ where $test = slack, util, demand$. Then, we calculate *admission ratio* = $admit_{grp,test}$ / num_{grp} for each $test$.

Figures 1–4 show the admission ratio of each EDZL schedulability tests when $m = 2, 3, 4, 5$, respectively. Low-utilization task sets are admitted by all the three tests. Most task sets with utilization less than $0.6 \times m$ are admitted by all tests. This value is close to the Piao's schedulable utilization bound, $(m + 1)/2$. However, as the task set utilization increases, the number of admitted task sets decreases and so does the ratio. When the task set utilization approaches m, the ratio decreases sharply. In all cases, the demand-based test admits high-utilization task sets better than others. The ratio of the demand-based test begins to decrease at a utilization much higher than other tests. The demand-based test outperforms other tests.

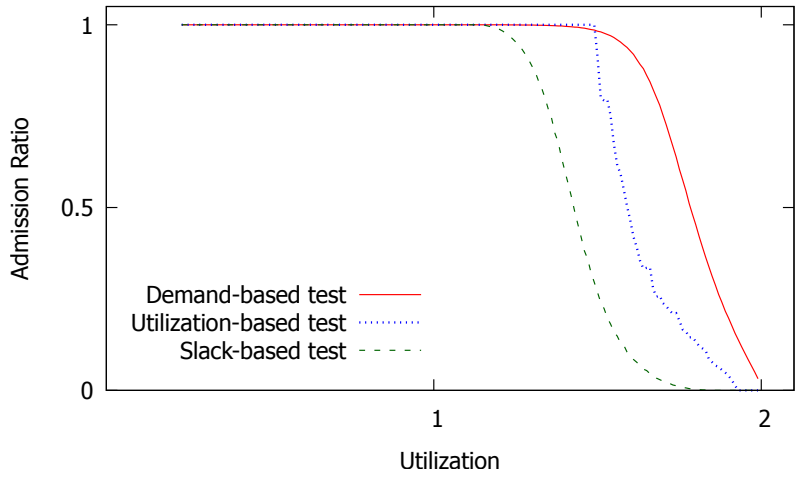

Figure 1. Admission ratio when $m = 2$.

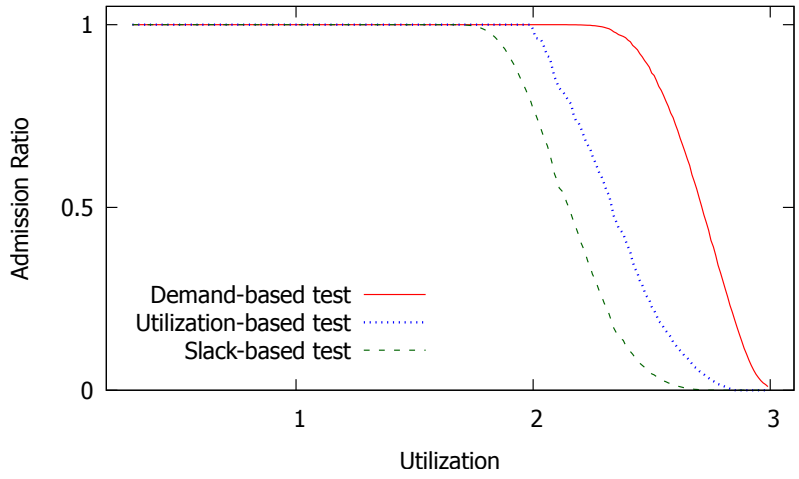

Figure 2. Admission ratio when $m = 3$.

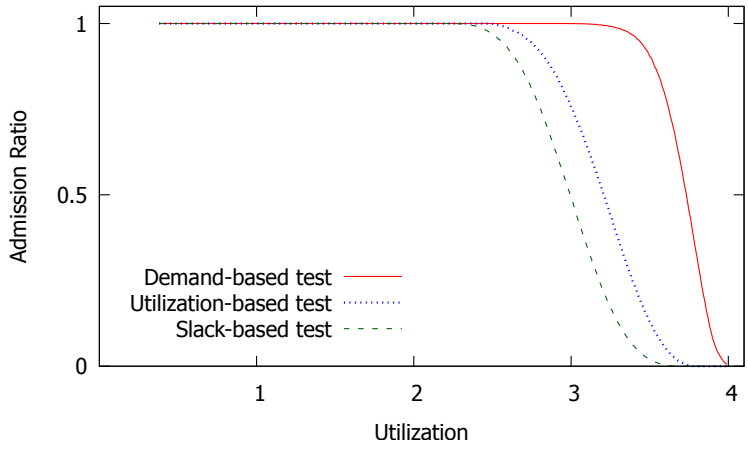

Figure 3. Admission ratio when $m = 4$.

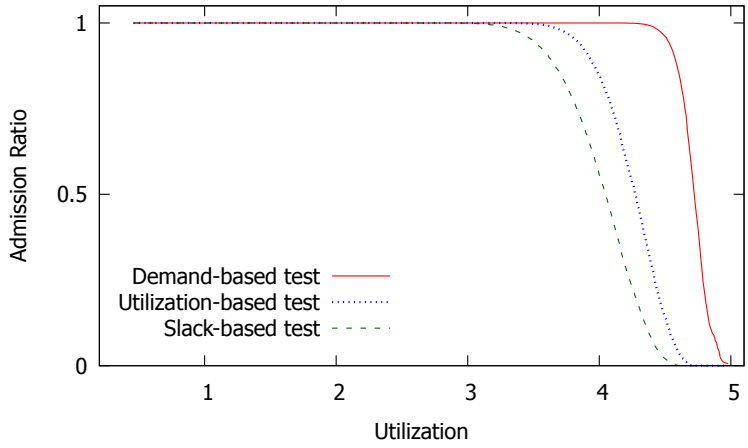

Figure 4. Admission ratio when $m = 5$.

3.2. Dominance between EDZL Schedulability Tests

Figure 5 shows the number of test instances admitted by each test and their intersections. The number of test instances admitted by the demand-based test is 884,159,737 (=181,804,304 + 93,269,942 + 1,280,464 + 607,805,027). The number of test instances admitted by the utilization-based test is 701,454,278 (=379,191 + 93,269,942 + 118 + 607,805,027). The number of test instances admitted by the slack-based test is 609,085,609 (=1,280,464 + 607,805,027 + 118).

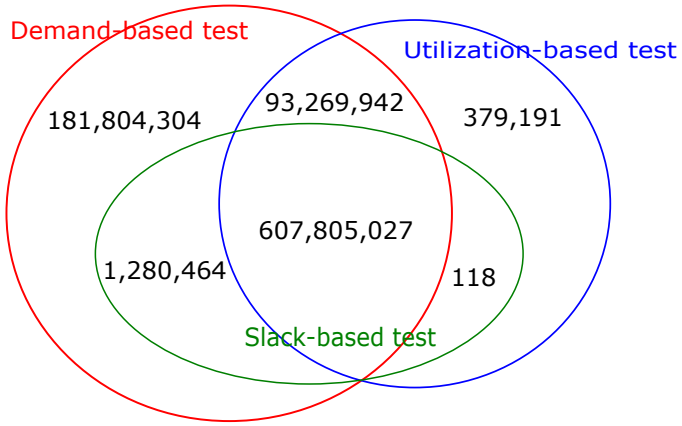

Figure 5. Number of test instances admitted by each test and their intersections.

The figure implies that the three EDZL schedulability tests do not dominate each other. The number of test instances admitted only by the demand-based test is 181,804,304. An example is $\tau^A = \{(1,2),(2,3),(3,4)\}$ on two processors. This test instance is admitted by the demand-based test but not by the utilization-based test nor by the slack-based test. The number of test instances admitted only by the utilization-based test is 379,191. An example is $\tau^B = \{(1,3),(1,6),(6,7),(5,10)\}$ on two processors. This test instance is admitted by the utilization-based test, but not by the demand-based test nor by the slack-based test.

An interesting result is that there is no test instance admitted only by the slack-based test. We explored the experiment results carefully and found out that the demand-based test does not dominate the slack-based test, i.e., there are test instances admitted by the slack-based test but not by the demand-based test (e.g., $\{(1,3),(1,4),(1,4),(3,12),(3,13)\}$ on two processors). And the utilization-based test does not dominate the slack-based test, either, i.e., there are test instances admitted by the slack-based test but not by the utilization-based test (e.g., $\{(1,2),(2,4),(1,7),(3,8)\}$ on two processors). However, any test instance admitted by the slack-based test can be also admitted by the demand-based test or the utilization-based test in our experiments. Of course, this does not imply that the union of the demand-based test and the utilization-based test (we denote this by *Demand + Util*) dominates the slack-based test. But the experimental results imply that Demand + Util is effective for the EDZL schedulability test.

As mentioned in Section 3.1, most task sets whose total utilization is below Piao's schedulable utilization bound are admitted by the three schedulability tests. However, some task sets are admitted by Piao's test but are not admitted by the slack-based test. An example is task set τ = {(3,5), (1,6), (4,8), (1,10), (1,11)} on two processors. Its total utilization is 1.46. This test instance is admitted by Piao's test but not admitted by the slack-based test. There are many test instances that are admitted by the slack-based test but not admitted by Piao's test. The slack-based test and Piao's test do not dominate each other.

Piao's schedulable utilization bound is the same as fpEDF's. As mentioned in Section 2.2, fpEDF can correctly schedule any task set τ if $U(\tau) \leq (m+1)/2$. This implies that any task set admitted by Piao's test is also schedulable by fpEDF. According to Lemma 2.4, the task set is admitted by the EDF$^{(k)}$ schedulability. Since the EDF$^{(k)}$ schedulability test and the utilization-based EDZL schedulability test are equivalent (Theorem 5), the task set can be also admitted by the utilization-based test. Therefore, Theorem 4 follows.

Theorem 4. *The utilization-based schedulability test dominates Piao's schedulability test.*

3.3. Tightness

We introduce a metric for evaluating the performance of the schedulability tests. We define *tightness* as the ratio of the number of admitted test instances to the number of schedulable test instances. An ideal schedulability test for a scheduling algorithm should admit all task sets that are schedulable by the algorithm. The tightness of an ideal schedulability test would be 1.0. For a given set of test instances, a schedulability test that can admit more test instances will have a higher tightness. We can use tightness to compare the performance of schedulability tests.

The tightness is a function of the set of test instances. If a different set of test instances is given, the set of schedulable test instances will be different, and thus the tightness of the same schedulability test will be different. Hence, we can employ tightness to compare the performance of schedulability tests for the same set of test instances. Furthermore, the tightness cannot be used to compare the schedulability test of different scheduling algorithms because the sets of schedulable test instances are different for the different scheduling algorithms. Note that a higher tightness does not imply dominance.

To obtain the number of schedulable test instances, we implemented a custom simulator for EDZL scheduling and performed simulations with the dataset. The simulator was implemented using C language. Given a task set, the simulator computes a hyper-period

of the task set. At every time from 0 to the hyper-period (in the domain of integers), the simulator (1) checks the job release time of every task and generates jobs, (2) gives priority to jobs based on deadline and laxity, (3) selects jobs to execute, (4) decreases the remaining execution time of selected jobs, and (5) checks job completion or deadline miss. Then, it outputs the simulation result. For each test instance in the dataset, EDZL scheduling was simulated and we identified whether each instance was successfully scheduled without missing any deadline during a hyper-period. As a result, the total number of schedulable test instances is 990,451,970.

Figure 6 shows the tightness of each EDZL schedulability test. The tightness of the demand-based test is much higher than other tests, as expected. The demand-based test can admit 45.15% more test instances than the slack-based test and 26.05% more test instances than the utilization-based test. And the union of the demand-based test and the utilization-based test (Demand + Util) admits 884,539,046 test instances, yielding a tightness of 0.8931.

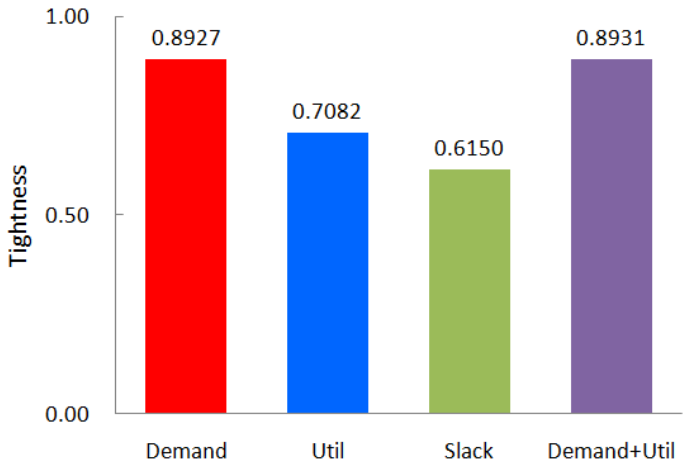

Figure 6. Tightness for each EDZL schedulability test. Demand + Util represents the union of demand-based test and utilization-based test.

4. Comparing EDZL and EDF$^{(k)}$

This section compares the schedulability of EDZL and EDF$^{(k)}$. EDZL and EDF$^{(k)}$ are not optimal, but they are practical with respect to implementation cost, scheduling overhead, and high processor utilization. This section investigates the dominance between their schedulability tests and compares the scheduling performance through simulations.

4.1. Dominance between EDZL and EDF$^{(k)}$ Schedulability Test

We look into the dominance between the schedulability tests for EDZL and EDF$^{(k)}$. We found out that the result of the schedulability test for EDF$^{(k)}$ is always the same as that of the utilization-based test for EDZL in our experiments. Motivated by this, Theorem 5 proves that the utilization-based test for the EDZL (Theorem 1) and EDF$^{(k)}$ schedulability test (Corollary 1) are equivalent.

Theorem 5. *The utilization-based EDZL schedulability test and the EDF$^{(k)}$ schedulability test are equivalent.*

Proof. We rewrite Equation (3) using the definition of $\tau^{(i)}$ and the assumption that $u_i \geq u_{i+1}$ as follows.

$$U(\tau^{(m-m'+1)}) \leq m' - (m'-1) \cdot u_{m-m'+1} \tag{9}$$

Let $k = m - m' + 1$. Then,

$$\begin{aligned} U(\tau^{(k)}) &\leq m - k + 1 - (m - k) \cdot u_k \\ u_k + U(\tau^{(k+1)}) &\leq m - k + 1 - (m - k) \cdot u_k \\ U(\tau^{(k+1)}) &\leq (m - k + 1)(1 - u_k) \\ m &\geq (k - 1) + \frac{U(\tau^{(k+1)})}{1 - u_k} \end{aligned} \tag{10}$$

Since both m and k are integers, the above equation can be rewritten as follows.

$$m \geq (k - 1) + \left\lceil \frac{U(\tau^{(k+1)})}{1 - u_k} \right\rceil \tag{11}$$

If τ can be admitted by the utilization-based EDZL schedulability test, there exists m' satisfying Equation (9). Then, Equation (11) holds for $k = m - m' + 1.$, i.e., τ can be admitted by EDF$^{(k)}$ schedulability. Similarly, if τ can be admitted by EDF$^{(k)}$ schedulability, there exists k such that Equation (11) holds. Then, for $m' = m - k + 1$ Equation (9) holds, i.e., τ can be admitted by the utilization-based EDZL schedulability test. Therefore, the utilization-based EDZL schedulability test and the EDF$^{(k)}$ schedulability test are equivalent and dominate each other. □

Theorem 5 implies that the set of task sets admitted by the utilization-based EDZL test is the same as the set of task sets that are admitted by the EDF$^{(k)}$ schedulability test. This also implies that Demand + Util can admit all task sets admitted by the EDF$^{(k)}$ schedulability test. And, as explained in Section 3.2, there exist task sets that the demand-based test can admit but the utilization-based test cannot. Therefore, Demand + Util strictly dominates the EDF$^{(k)}$ schedulability test.

4.2. Success Ratio

This section compares the performance of EDZL and EDF$^{(k)}$ scheduling in terms of the number of successfully scheduled test instances. In soft real-time systems, a deadline miss does not cause system failure but merely degrades the system's operation. In such systems, a test instance that does not pass the schedulability test might be executed. In this experiment, all test instances are simulated without a schedulability test. For this experiment, we extended the custom simulator (explained in Section 3.3) to perform EDF$^{(k)}$ scheduling. And the simulation was performed with the same dataset used in Section 3. Then, we compared the results with the EDZL scheduling simulation results.

To begin with, we compare the performance of the EDZL and EDF$^{(k)}$ algorithms from a quantitative perspective. Similar to the evaluation in Section 3.1, we count the number of test instances successfully scheduled by EDZL and EDF$^{(k)}$, respectively, for each task utilization interval. Then, we calculate their ratio, which we define as *success ratio*. Figures 7–10 show the success ratio when $m = 2, 3, 4, 5$, respectively. The figures demonstrate that EDZL can successfully schedule more high-utilization task sets than EDF$^{(k)}$. For example, when $m = 4$, EDF$^{(k)}$ begins to decrease at a utilization of about 3.2, EDZL begins to decrease at a utilization of about 3.8.

As mentioned in Section 3, the number of all test instances is 1,000,752,406 and EDZL successfully scheduled 990,451,970 instances out of these. In our EDF$^{(k)}$ scheduling simulations, EDF$^{(k)}$ successfully scheduled 919,989,568 instances. EDZL can successfully schedule 7.0% more instances than EDF$^{(k)}$. We also discover the dominance between the EDZL and EDF$^{(k)}$ algorithms. We investigate the simulation results of both algorithms and we found out that the two algorithms do not dominate each other. Task set $\{(5, 8), (1, 2), (3, 6), (3, 8)\}$ can be successfully scheduled by EDF$^{(k)}$ on two processors, i.e., all deadlines are met for a hyper-period in the simulation. However, EDZL fails to successfully schedule this task set; in its EDZL scheduling simulation, a deadline miss occurs at time 24. Thus, EDZL does not

dominate EDF$^{(k)}$. On the other hand, task set $\{(2,3),(3,5),(1,3),(2,6)\}$ can be successfully scheduled by EDZL on two processors, but not by EDF$^{(k)}$. EDF$^{(k)}$ does not dominate EDZL.

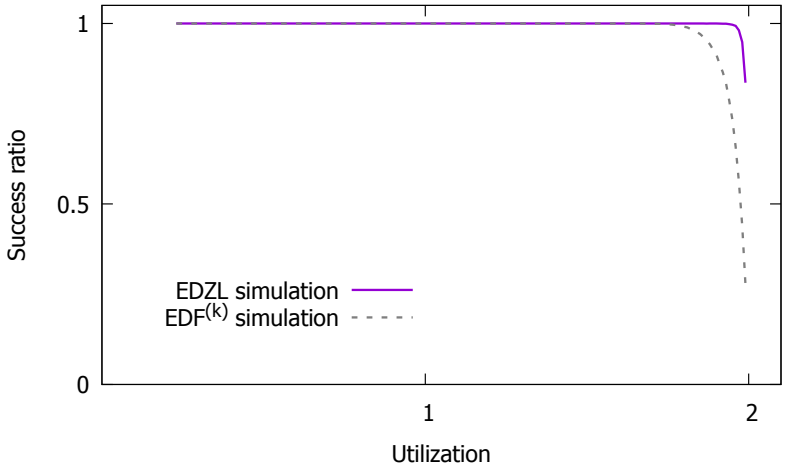

Figure 7. Success ratio when $m = 2$.

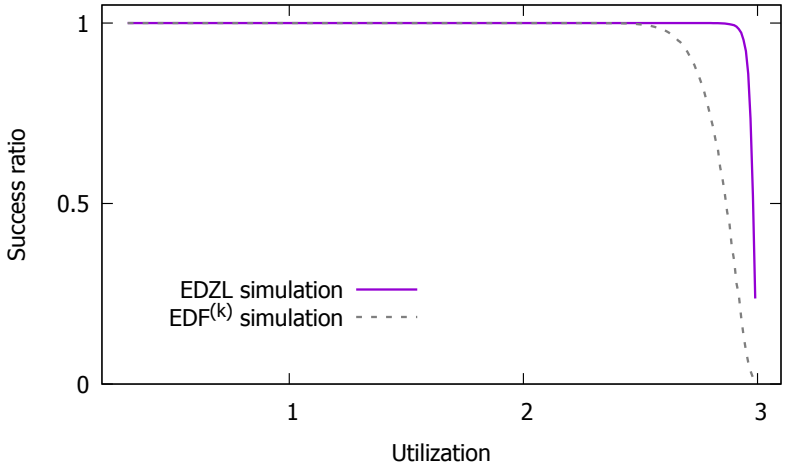

Figure 8. Success ratio when $m = 3$.

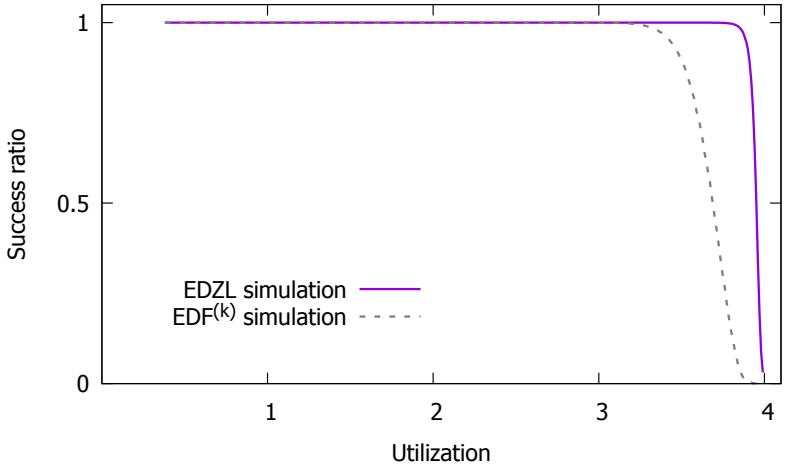

Figure 9. Success ratio when $m = 4$.

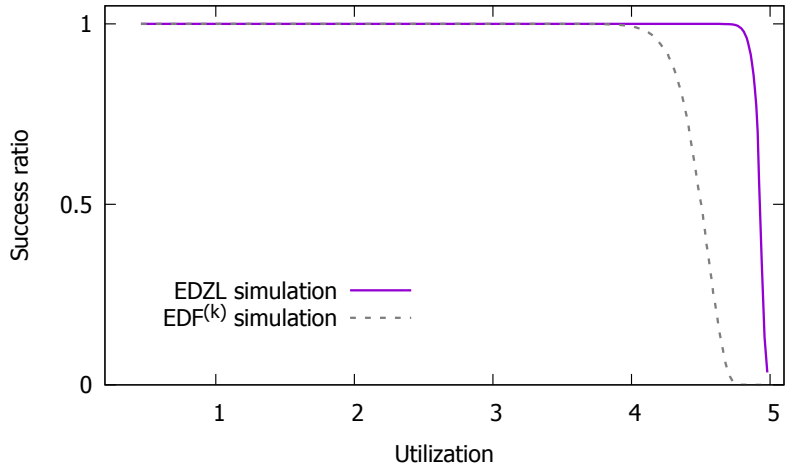

Figure 10. Success ratio when $m = 5$.

5. Conclusions

This work evaluates the schedulability of the EDZL algorithm. First, we compare the performance of EDZL schedulability tests. With respect to the number of admitted test instances, the demand-based test outperforms the utilization-based test and the slack-based test. In our experiments, Demand + Util can admit all test instances that are admitted by the slack-based test. Demand + Util admits 89.31% of EDZL-schedulable test instances.

Second, we compare EDZL and EDF$^{(k)}$. In respect to the schedulability test, this paper shows that the utilization-based EDZL schedulability test is equivalent to the EDF$^{(k)}$ schedulability test. Hence, Demand + Util strictly dominates the EDF$^{(k)}$ schedulability test. Additionally, this paper compares EDZL and EDF$^{(k)}$ scheduling in terms of schedulable test instances. EDZL and EDF$^{(k)}$ do not dominate each other. However, EDZL can accommodate more task sets than EDF$^{(k)}$. In our experiments, EDZL successfully schedules 7.0% more test instances than EDF$^{(k)}$.

In future works, we plan to prove that Demand + Util dominates the slack-based EDZL schedulability test. By disclosing the dominance between schedulability tests, we can determine the feasibility of task sets efficiently. Furthermore, our work assumes implicit deadline systems. We plan to develop EDZL schedulability tests under constrained or arbitrary deadline systems.

Author Contributions: Conceptualization, S.H.; Formal analysis, S.H.; Software, S.H.; Writing—original draft, S.H.; Writing—review and editing, W.P., M.-C.K. and M.P.; Validation, W.P., M.-C.K. and M.P. All authors have read and agreed to the published version of the manuscript.

Funding: This paper was supported by Konkuk University in 2021 (no. 2021-A019-0364).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data is contained within the article.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of the data; in the writing of the manuscript; nor in the decision to publish the results.

References

1. Baruah, S.; Cohen, N.; Plaxton, C.; Varvel, D. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica* **1996**, *15*, 600–625. [CrossRef]
2. Andersson, B.; Tovar, E. Multiprocessor scheduling with few preemptions. In Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06), Sydney, QLD, Australia, 16–18 August 2006; pp. 322–334.

3. Cho, H. An Optimal Real-Time Scheduling Algorithm for Multiprocessors. In Proceedings of the 27th IEEE Real-Time Systems Symposium, Rio de Janeiro, Brazil, 5–8 December 2006.
4. Anderson, J.H.; Srinivasan, A. Mixed Pfair/ERfair scheduling of asynchronous periodic tasks. *J. Comput. Syst. Sci.* **2004**, *68*, 157–204. [CrossRef]
5. Baruah, S.K.; Gehrke, J.E.; Plaxton, C.G. Fast scheduling of periodic tasks on multiple resources. In Proceedings of the 9th International Parallel Processing Symposium, Santa Barbara, CA, USA, 25–28 April 1995; pp. 280–288.
6. Funk, S.; Levin, G.; Sadowski, C.; Pye, I.; Brandt, S. DP-Fair: A unifying theory for optimal hard real-time multiprocessor scheduling. *Real-Time Syst.* **2011**, *47*, 389–429. [CrossRef]
7. Cho, S.; Lee, S.K.; Ahn, S.; Lin, K.J. Efficient real-time scheduling algorithms for multiprocessor systems. *IEICE Trans. Commun.* **2002**, *85*, 2859–2867.
8. Park, M.; Han, S.; Kim, H.; Cho, S.; Cho, Y. Comparison of deadline-based scheduling algorithms for periodic real-time tasks on multiprocessor. *IEICE Trans. Inf. Syst.* **2005**, *88*, 658–661. [CrossRef]
9. Alhussian, H.; Zakaria, N. An Efficient Zero-Laxity Based Real-Time Multiprocessor Scheduling Algorithm. In Proceedings of the 5th International Conference on IT Convergence and Security (ICITCS), Kuala Lumpur, Malaysia, 24–27 August 2015; pp. 1–4.
10. Choi, S.; Cho, S.; Park, J.; Nam, B.G. Earliest virtual deadline zero laxity scheduling for improved responsiveness of mobile GPUs. *J. Semicond. Technol. Sci.* **2017**, *17*, 162–166. [CrossRef]
11. Alhussian, H.; Zakaria, N.; Patel, A.; Jaradat, A.; Abdulkadir, S.J.; Ahmed, A.Y.; Bahbouh, H.T.; Fageeri, S.O.; Elsheikh, A.A.; Watada, J. Investigating the schedulability of periodic real-time tasks in virtualized cloud environment. *IEEE Access* **2019**, *7*, 29533–29542. [CrossRef]
12. Jung, N.; Baek, H.; Lim, D.; Lee, J. Incorporating zero-laxity policy into mixed-criticality multiprocessor real-time systems. *IEICE Trans. Fundam.* **2018**, *101*, 1888–1899. [CrossRef]
13. Chen, T.Y.; Wei, H.W.; Leu, J.S.; Shih, W.K. EDZL scheduling for large-scale cyber service on real-time cloud. In Proceedings of the 2011 IEEE International Conference on Service-Oriented Computing and Applications (SOCA), Irvine, CA, USA, 12–14 December 2011; pp. 1–3.
14. Wu, P.; Majeed, S.; Ryu, M. Two approaches towards EDZL scheduling for performance asymmetric multiprocessors. In Proceedings of the 2016 IEEE International Conference on Network Infrastructure and Digital Content (IC-NIDC), Beijing, China, 23–25 September 2016; pp. 120–123.
15. Baek, H.; Lee, J. Task-level re-execution framework for improving fault tolerance on symmetry multiprocessors. *Symmetry* **2019**, *11*, 651. [CrossRef]
16. Piao, X.; Han, S.; Kim, H.; Park, M.; Cho, Y.; Cho, S. Predictability of earliest deadline zero laxity algorithm for multiprocessor real-time systems. In Proceedings of the Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'06), Gyeongju, Republic of Korea, 24–26 April 2006; p. 6.
17. Wei, H.W.; Chao, Y.H.; Lin, S.S.; Lin, K.J.; Shih, W.K. Current Results on EDZL Scheduling for Multiprocessor Real-Time Systems. In Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2007), Daegu, Republic of Korea, 21–24 August 2007; pp. 120–130. [CrossRef]
18. Baker, T.P.; Cirinei, M.; Bertogna, M. EDZL scheduling analysis. *Real-Time Syst.* **2008**, *40*, 264–289. [CrossRef]
19. Lee, J.; Shin, I. EDZL schedulability analysis in real-time multicore scheduling. *IEEE Trans. Softw. Eng.* **2012**, *39*, 910–916. [CrossRef]
20. Lee, J.; Shin, I. Demand-based schedulability analysis for real-time multi-core scheduling. *J. Syst. Softw.* **2014**, *89*, 99–108. [CrossRef]
21. Goossens, J.; Funk, S.; Baruah, S. Priority-driven scheduling of periodic task systems on multiprocessors. *Real-Time Syst.* **2003**, *25*, 187–205. [CrossRef]
22. Liu, C.L.; Layland, J.W. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *J. ACM* **1973**, *20*, 46–61. [CrossRef]
23. Audsley, N.C.; Burns, A.; Richardson, M.F.; Wellings, A.J. Real-Time scheduling: the deadline-monotonic approach. In Proceedings of the IEEE Workshop on Real-Time Operating Systems and Software, Atlanta, GA, USA, 15–17 May 1991.
24. Carpenter, J.; Funk, S.; Holman, P.; Srinivasan, A.; Anderson, J.; Baruah, S. Chapter 2. In *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*; Chapman Hall/CRC Press: Boca Raton, FL, USA, 2004.
25. Baruah, S.K.; Goossens, J. Rate-monotonic scheduling on uniform multiprocessors. *IEEE Trans. Comput.* **2003**, *52*, 966–970. [CrossRef]
26. Baruah, S.; Goossens, J. Deadline monotonic scheduling on uniform multiprocessors. In Proceedings of the International Conference on Principles of Distributed Systems, Brussels, Belgium, 15–18 December 2008; pp. 89–104.
27. Mok, A. Task Management Techniques for Enforcing ED Scheduling on a Periodic Task Set. In Proceedings of the 5th IEEE Workshop on Real-Time Software and Operating Systems, Washington, DC, USA, 12–13 May 1988; pp. 42–46.
28. Funk, S.H. EDF Scheduling on Heterogeneous Multiprocessors. Ph.D. Thesis, University of North Carolina at Chapel Hill, Chapel Hill, NC, USA, 2004.
29. Dhall, S.K.; Liu, C.L. On a Real-Time Scheduling Problem. *Oper. Res.* **1978**, *26*, 127–140. [CrossRef]
30. Srinivasan, A.; Baruah, S. Deadline-based scheduling of periodic task systems on multiprocessors. *Inf. Process. Lett.* **2002**, *84*, 93–98. [CrossRef]

31. Baruah, S.K. Optimal utilization bounds for the fixed-priority scheduling of periodic task systems on identical multiprocessors. *IEEE Trans. Comput.* **2004**, *53*, 781–784. [CrossRef]
32. Davis, R.I.; Burns, A. FPZL schedulability analysis. In Proceedings of the 17th IEEE Symposium Real-Time and Embedded Technology and Applications, Chicago, IL, USA, 11–14 April 2011; pp. 245–256.
33. Davis, R.I.; Kato, S. FPSL, FPCL and FPZL schedulability analysis. *Real-Time Syst.* **2012**, *48*, 750–788. [CrossRef]
34. Lee, J.; Easwaran, A.; Shin, I.; Lee, I. Zero-laxity based real-time multiprocessor scheduling. *J. Syst. Softw.* **2011**, *84*, 2324–2333. [CrossRef]
35. Rubio-Anguiano, L.; Ramírez-Treviño, A.; Chils, A.; Briz, J. Real time scheduler for multiprocessor systems based on continuous control using Timed Continuous Petri Nets. *IFAC-PapersOnLine* **2020**, *53*, 371–377. [CrossRef]
36. Kato, S.; Yamasaki, N. Global EDF-based scheduling with laxity-driven priority promotion. *J. Syst. Architect.* **2011**, *57*, 498–517. [CrossRef]
37. Anderson, J.H.; Srinivasan, A. Early-release fair scheduling. In Proceedings of the 12th Euromicro Conference on Real-Time Systems (Euromicro RTS 2000), Stockholm, Sweden, 19–21 June 2000; pp. 35–43.
38. Levin, G.; Funk, S.; Sadowski, C.; Pye, I.; Brandt, S. DP-FAIR: A simple model for understanding optimal multiprocessor scheduling. In Proceedings of the 2010 22nd Euromicro Conference on Real-Time Systems, Brussels, Belgium, 6–9 July 2010; pp. 3–13.
39. Nelissen, G.; Berten, V.; Nélis, V.; Goossens, J.; Milojevic, D. U-EDF: An unfair but optimal multiprocessor scheduling algorithm for sporadic tasks. In Proceedings of the 2012 24th Euromicro Conference on Real-Time Systems, Pisa, Italy, 11–13 July 2012; pp. 13–23.
40. Regnier, P.; Lima, G.; Massa, E.; Levin, G.; Brandt, S. Run: Optimal multiprocessor real-time scheduling via reduction to uniprocessor. In Proceedings of the 2011 IEEE 32nd Real-Time Systems Symposium, Washington, DC, USA, 29 November–2 December 2011; pp. 104–115.
41. Leung, J.Y.T. A new algorithm for scheduling periodic, real-time tasks. *Algorithmica* **1989**, *4*, 209–219. [CrossRef]