EDZL Schedulability Analysis in Real-Time Multicore Scheduling

Jinkyu Lee, *Member*, *IEEE*, and Insik Shin, *Member*, *IEEE*

Abstract—In real-time systems, correctness depends not only on functionality but also on timeliness. A great number of scheduling theories have been developed for verification of the temporal correctness of jobs (software) in such systems. Among them, the Earliest Deadline first until Zero-Laxity (EDZL) scheduling algorithm has received growing attention thanks to its effectiveness in multicore real-time scheduling. However, the true potential of EDZL has not yet been fully exploited in its schedulability analysis as the state-of-the-art EDZL analysis techniques involve considerable pessimism. In this paper, we propose a new EDZL multicore schedulability test. We first introduce an interesting observation that suggests an insight toward pessimism reduction in the schedulability analysis of EDZL. We then incorporate it into a well-known existing Earliest Deadline First (EDF) schedulability test, resulting in a new EDZL schedulability test. We demonstrate that the proposed EDZL test not only has lower time complexity than existing EDZL schedulability tests, but also significantly improves the schedulability of EDZL by up to 36.6 percent compared to the best existing EDZL schedulability tests.

Index Terms—Earliest Deadline first until Zero-Laxity (EDZL), real-time scheduling, schedulability analysis, multicore platform, real-time systems

1 Introduction

A real-time system is one in which its correctness depends not only on logic behavior but also on the time at which results are produced. Real-time systems are increasingly deployed in many safety-critical environments, including avionics, automobiles, space engineering, and medical devices. Over the past several decades, substantial research has been conducted on real-time scheduling theories for temporal behavior analysis of real-time systems [1], [2], [3]. With the increasing popularity of multicore architectures, there is also growing attention to real-time scheduling on multicore platforms.

Many real-time scheduling algorithms have been studied for multicore platforms, and global earliest deadline first until zero-laxity (EDZL) [4] has great potential for effective real-time multicore scheduling because it inherently considers both the urgency of real-time jobs and the parallelism aspect of multicore platform simultaneously [5] with low runtime overheads [6], [7]. Global EDZL scheduling assigns the highest priority to zero-laxity jobs and schedules remaining jobs using earliest deadline first (EDF), where the laxity of a job at any instant is defined as remaining time to deadline minus the amount of remaining execution time. Since EDZL promotes the priority of any job that would otherwise inevitably miss its deadline (i.e., any zero-laxity job), it not only dominates[1] EDF [7], but also is known to have a better scheduling performance than other scheduling algorithms on multicore platforms without incurring many preemptions [6], [7].

However, the true potential of EDZL has not been fully utilized in its schedulability analysis, which determines whether or not all the timing requirements of a given task set can be satisfied under EDZL scheduling. Existing EDZL analysis techniques [8], [9], though outperforming their EDF counterparts significantly, have not matured yet, making unnecessarily pessimistic schedulability decisions.

The goal of this paper is to develop a new, "good" schedulability test of EDZL. We first present an important observation on a task set under EDZL scheduling. This observation mainly comes from the priority promotion of zero-laxity jobs, and therefore does not necessarily hold for other scheduling algorithms (e.g., EDF). Building upon the observation, we derive a new EDZL schedulability test based on an existing EDF schedulability test [10] and the dominance relationship between EDZL and EDF [7]. The proposed EDZL test not only reduces the pessimism of the state-of-the-art EDZL analysis techniques significantly, but also runs in lower time complexity. According to our simulation results, the proposed test can find up to 36.6 percent more task sets schedulable than the best existing EDZL schedulability tests can.

The rest of this paper is organized as follows: Section 2 presents our system model and existing EDZL schedulability tests. Section 3 derives an important condition for a task set to be schedulable by EDZL. Based on the condition, Section 4 develops a new EDZL schedulability test. Section 5 evaluates the performance of the proposed EDZL

1. A scheduling algorithm or a schedulability test A dominates B if any task set schedulable by B is also schedulable by A, but the converse is not true.

• *J. Lee is with the Department of Electrical Engineering and Computer Science, The University of Michigan, 2260 Hayward St., Ann Arbor, MI 48109-2121. E-mail: jinkyul@umich.edu.*
• *I. Shin is with the Department of Computer Science, KAIST, 291 Daehakro, Yuseong-gu, Daejeon 305-701, South Korea. E-mail: insik.shin@cs.kaist.ac.kr.*

Manuscript received 16 Mar. 2012; revised 13 Sept. 2012; accepted 31 Oct. 2012; published online 7 Nov. 2012.
Recommended for acceptance by M. Woodside.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-2012-03-0063.
Digital Object Identifier no. 10.1109/TSE.2012.75.

schedulability test in terms of scheduling performance and time complexity. Section 6 discusses related work, and finally, Section 7 concludes this paper.

2 Background

In this section, we first introduce our system model and notations. Then, we recapitulate existing EDZL schedulability tests [8], [9].

2.1 System Model and Notations

In this paper, we focus on a sporadic task model [11] in which a task $\tau_i \in \mathcal{T}$ is specified as (T_i, C_i, D_i), where T_i is the minimum separation, C_i is the worst-case execution time requirement, and D_i is the relative deadline. We restrict our attention to constrained deadline task sets, i.e., $C_i \le D_i \le T_i, \forall \tau_i \in \mathcal{T}$. A task τ_i invokes a series of jobs, each separated from its predecessor by at least T_i time units. Also, each job of τ_i should finish its execution no later than T_i time units after its release. We let δ_i denote C_i/D_i. In this paper, we let $|A|$ denote the number of elements in A, and thus $|\mathcal{T}|$ means the number of tasks in $\mathcal{T}$.

We assume that the platform consists of m identical unit-speed cores. We also assume that any single job cannot be executed in parallel.

2.2 Existing EDZL Schedulability Analysis

In this section, we introduce two existing EDZL schedulability tests: a basic test [8], [9] and its improved version [9]. To the best knowledge of the authors, these are the only existing EDZL schedulability tests which judge whether a given task set is schedulable or not under EDZL, while there have been a few studies that address the scheduling performance of EDZL in a different manner (e.g., the studies [12], [13] have proven that a theoretical upper bound of utilization of EDZL-schedulable task sets is no larger than a certain value).

The existing EDZL schedulability tests use an important property of EDZL as stated in the following lemma.

Lemma 1. *A task set $\mathcal{T}$ is schedulable by EDZL on an m-core platform if there are at most m zero-laxity jobs at any time instant.*

Proof. Since EDZL gives the highest priority to zero-laxity jobs, the lemma trivially holds. □

Then, the remaining step is to judge whether a job of a given task can reach a zero-laxity state or not. To do this, the existing EDZL schedulability tests calculate the time duration in the interval between the release time and deadline of a job of τ_k when any job of τ_i is executed and has higher priority than the job of τ_k. Fig. 1 shows a situation for the maximum duration, and it is calculated by $I_i(\max(0, D_k - S_i))$ [8], [9], where

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

Here, S_i means the slack between the maximum response time and deadline of a job of τ_i. In other words, $T_i - S_i$ is the maximum response time of a job of τ_i.

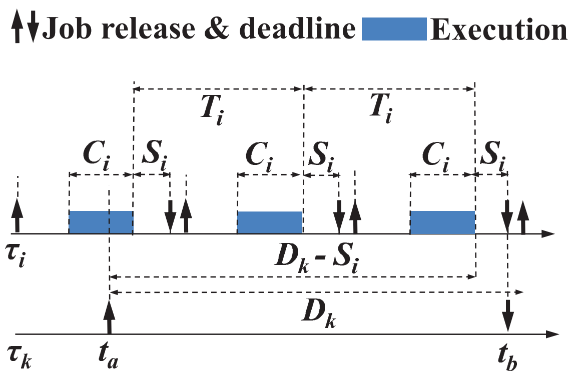


Fig. 1. Situation for the maximum duration in the interval between the release time and deadline of a job of τ_k when any job of τ_i is executed and has higher priority than the job of τ_k.

Using the maximum duration and Lemma 1, the following lemma introduces schedulability tests of EDZL [8], [9].

Lemma 2 (The Existing EDZL Schedulability Tests in [8], [9][2]). *A task set $\mathcal{T}$ is schedulable by EDZL on an m-core platform if the following inequality holds for at most m different tasks $\tau_k \in \mathcal{T}$:*

$$\sum_{\tau_i \in \mathcal{T} - \{\tau_k\}} \min(I_i(\max(0, D_k - S_i)), D_k - C_k) \ge m \cdot (D_k - C_k). \quad (2)$$

Note that a basic test ([8, Theorem 7] or [9, Theorem 3]) does not utilize the slack value (i.e., $S_i = 0$ for all $\tau_i \in \mathcal{T}$), but its improved version ([9, Fig. 3]) calculates the slack value in an iterative manner (details are given in [9, Fig. 3]).

Proof. The proof is given in [8], [9]. In summary, for a job of τ_k to reach a zero-laxity state, time to deadline should be the same as the remaining execution time. And, at each time slot, at least m higher priority jobs' execution is needed to block a job of τ_k's execution. Hence, the sum of higher priority jobs' execution should be at least $m \cdot (D_k - C_k)$ for a job of τ_k to reach a zero-laxity state. By Lemma 1, if (2) holds for at most m tasks, $\mathcal{T}$ is schedulable by EDZL. □

The existing EDZL schedulability tests in Lemma 2 are significant in that they incorporate the property of EDZL in Lemma 1 into schedulability conditions. However, the existing tests can be pessimistic (i.e., deeming a schedulable set unschedulable) due to the overestimation of higher priority executions. When $I_i(\max(0, D_k - S_i))$ is calculated, the worst-case release pattern, where the amount of higher priority execution of jobs of τ_i is maximized, is considered as shown in Fig. 1. Then, these $I_i(\max(0, D_k - S_i))$ for all $\tau_i \in \mathcal{T} - \{\tau_k\}$ are simply added to the LHS of (2), which implies all the worst-case release patterns can occur at the same time. For most task sets, this is not true, so such pessimism grows as the number of tasks increases. We will discuss such a limitation of the existing tests in Section 5 when our proposed EDZL schedulability test is evaluated.

2. A minor error in [8], [9] has been corrected in [14].

3 Properties of EDZL

In this section, we derive an important condition for a given task set to be schedulable by EDZL, which will be a basis for a new EDZL schedulability test to be developed in Section 4.

To do this, we first present a property of EDZL in the following lemma.

Lemma 3. *If a task set $\mathcal{T}$ is schedulable by EDZL on an m-core platform, there are at most m zero-laxity jobs at any time instant.*

Proof. We prove the contraposition. If there are more than m zero-laxity jobs at t, at least one zero-laxity job cannot be executed at t, which results in the job's deadline miss. □

Using the two properties in Lemmas 1 and 3, we derive a novel condition for a given task set to be schedulable by EDZL in the following theorem.

Theorem 1. *If $\mathcal{T}_1$ is schedulable by EDZL on an m-core platform, $\mathcal{T} \triangleq \mathcal{T}_1 \cup \mathcal{T}_2$ is schedulable by EDZL on an $(m + |\mathcal{T}_2|)$-core platform.*

Proof. For notational convenience, we let $\mathcal{T}_1'$ denote $\mathcal{T}_1$ when $\mathcal{T}_1$ is scheduled by EDZL on an m-core platform, and let $\mathcal{T}_1''$ denote $\mathcal{T}_1$ when $\mathcal{T} \triangleq \mathcal{T}_1 \cup \mathcal{T}_2$ is scheduled by EDZL on an $(m+|\mathcal{T}_2|)$-core platform.

We first prove the following claims, and then prove the theorem using the claims:

- C1. The response time of a job, which is invoked by a task in $\mathcal{T}_1''$, is no longer than that of the corresponding job which is invoked by the corresponding task in $\mathcal{T}_1'$.
- C2. At any time instant, there are at most m zero-laxity jobs which are invoked by tasks in $\mathcal{T}_1''$.

We first prove C1. Since we restrict our attention to constrained deadline tasks, each task has at most one unfinished, ready job at any time instant. This means jobs which are invoked by tasks in $\mathcal{T}_2$ do not occupy more than $|\mathcal{T}_2|$ cores at any time instant and, in other words, at least m cores are given to $\mathcal{T}_1''$ at any time instant. Therefore, the number of cores given to $\mathcal{T}_1''$ is always no smaller than that given to $\mathcal{T}_1'$. This proves C1.

Now, we prove C2. By Lemma 3, the number of zero-laxity jobs which are invoked by tasks in $\mathcal{T}_1'$ is at most m, and C1 implies that the number of zero-laxity jobs which are invoked by $\mathcal{T}_1''$ is no larger than that by $\mathcal{T}_1'$ at any time instant. This proves C2.

Since each task has at most one unfinished, ready job at any time instant, there are at most $|\mathcal{T}_2|$ zero-laxity jobs which are invoked by tasks in $\mathcal{T}_2$ at any time instant. By C2, there are at most m zero-laxity jobs which are invoked by tasks in $\mathcal{T}_1''$ at any time instant. Therefore, at any time instant there are at most $(m + |\mathcal{T}_2|)$ zero-laxity jobs which are invoked by tasks in $\mathcal{T} \triangleq \mathcal{T}_1 \cup \mathcal{T}_2$. This proves the theorem by Lemma 1. □

Note that the theorem is valid due to the priority promotion of zero-laxity jobs. That is, since zero-laxity jobs have the highest priority, we do not care for deadline satisfaction of jobs in $\mathcal{T}_2$ as long as there are at most $(m + |\mathcal{T}_2|)$ zero-laxity jobs which are invoked by tasks in $\mathcal{T} \triangleq \mathcal{T}_1 \cup \mathcal{T}_2$. Therefore, the theorem holds for any ZL-based scheduling algorithm which gives the highest priority to zero-laxity jobs [5], and does not necessarily hold for other scheduling algorithms. This property will be detailed in Section 6.

4 New EDZL Schedulability Analysis

In this section, we first recapitulate the dominance relationship between EDZL and EDF and an existing EDF schedulability test [10]. Then, we develop a new schedulability test of EDZL by incorporating the relationship and the EDF test into Theorem 1.

We now present the dominance relationship between EDZL and EDF in the following lemma.

Lemma 4 ([7, Theorem 2]). *If a task set $\mathcal{T}$ is schedulable by EDF on an m-core platform, then it is also schedulable by EDZL on the same platform.*

Proof. The proof is given in [7, Theorem 2]. In summary, at any time instant, EDF chooses a set of m jobs with the earliest deadlines. Then, the set should include all zero-laxity jobs as long as the task set is schedulable by EDF. Otherwise, the zero-laxity jobs are supposed to miss their deadlines. Therefore, EDZL, which gives the highest priority to zero-laxity jobs and schedules remaining jobs by EDF, chooses the same jobs as EDF as long as the task set is schedulable by EDF. □

We present an existing EDF schedulability test in the following lemma.

Lemma 5 ([10, Theorem 3], [15, Theorem 4]). *A task set $\mathcal{T}$ is schedulable by EDF on an m-core platform if it satisfies the following condition:*

$$\sum_{\tau_i \in \mathcal{T}} \delta_i \le m - (m-1) \cdot \max_{\tau_j \in \mathcal{T}} \delta_j. \tag{3}$$

Proof. The proof is given in [10, Theorem 3]. □

Incorporating Lemmas 4 and 5 into Theorem 1, we develop a new schedulability test of EDZL as stated in the following theorem.

Theorem 2. *A task set $\mathcal{T}$ is schedulable by EDZL on an m-core platform if it satisfies one of the following conditions for $m' = 1, 2, \ldots, m$:*

$$\sum_{\tau_i \in \mathcal{T}_1} \delta_i \le m' - (m'-1) \cdot \max_{\tau_j \in \mathcal{T}_1} \delta_j, \tag{4}$$

where

$$\mathcal{T}_1 \triangleq \{\tau_i \in \mathcal{T} | \tau_i \notin m - m' \textit{ tasks with the largest } \delta_i\}.$$

Note (4) is equivalent to (3) when $m' = m$.

Proof. We let $\mathcal{T}_2$ denote

$$\{\tau_i \in \mathcal{T} | \tau_i \in m - m' \text{ tasks with the largest } \delta_i\}.$$

Then, it holds that $\mathcal{T}_1 \cup \mathcal{T}_2 = \mathcal{T}$ and $|\mathcal{T}_2| = m - m'$.

Suppose that (4) holds for given m'. Then, by Lemma 5, $\mathcal{T}_1$ is schedulable by EDF on an m'-core platform, and then, by Lemma 4, it is also schedulable by EDZL on the same platform. Then, by Theorem 1, $\mathcal{T}$ is schedulable by EDZL on an m-core (i.e., $m' + (m - m') = m$) platform. □

Note that it can be trivially shown that the EDZL schedulability test in Theorem 2 dominates the EDF schedulability test in Lemma 5.

We give an example that shows how the proposed EDZL schedulability test in Theorem 2 works. We consider a set of three tasks $\mathcal{T} = \{\tau_1(10,9,10), \tau_2(10,6,10), \tau_3(5,2,5)\}$ and a 2-core platform, and thus it holds that $\delta_1 = 0.9$, $\delta_2 = 0.6$, and $\delta_3 = 0.4$. Then, a task set $\mathcal{T}$ is deemed not schedulable by the EDF schedulability test in Lemma 5 since the LHS of (3) is $0.9 + 0.6 + 0.4 = 1.9$, but the RHS is $2 - 1 \cdot 0.9 = 1.1$. If we consider $m' = 1$ in Theorem 2 (i.e., $\mathcal{T}_1 = \{\tau_2, \tau_3\}$), (4) holds since the LHS is $0.6 + 0.4 = 1$ and the RHS is $1 - 0 \cdot 0.6 = 1$. Therefore, by our EDZL schedulability test in Theorem 2, we guarantee that $\mathcal{T}$ is schedulable by EDZL. Note that $\mathcal{T}$ is deemed not schedulable by both existing EDZL schedulability tests in [8], [9] (i.e., the basic test in Lemma 2 with $S_i = 0$ and the improved test in the same lemma with S_i assignment according to [9, Fig. 3]).

5 Evaluation

In this section, we evaluate the performance of the proposed EDZL schedulability test in comparison with the existing EDZL schedulability tests in [8], [9]. We first explain how to generate task sets for simulation, then present simulation results to compare their schedulability performance, and finally analyze their time-complexity.

5.1 Task Set Generation

We generate task sets based on a technique proposed earlier [16] which has also been used in many previous studies (e.g., [17], [18], [19]). We have two input parameters: 1) the number of cores m (2, 4, 8, 16, 32, 48, or 64) and 2) individual task utilization (C_i/T_i) distribution (bimodal with parameter[3]: 0.1, 0.3, 0.5, 0.7, or 0.9, or exponential with parameter[4]: 0.1, 0.3, 0.5, 0.7, or 0.9). For each task, T_i is uniformly distributed in $[1, T_{max} = 1{,}000]$, C_i is chosen based on the given bimodal or exponential parameter, and D_i is set to T_i, which means we focus on implicit deadline task sets.

For each combination of parameters 1 and 2, we repeat the following procedure and generate 100,000 task sets:

1. Initially, we generate a set of $m + 1$ tasks.
2. In order to exclude unschedulable sets, we check whether the generated task set can pass a necessary feasibility condition of implicit deadline task sets [20] (i.e., $\sum_{\tau_i \in \mathcal{T}} \delta_i \leq m$).
3. If it fails to pass the feasibility test, we discard the generated task set and return to Step 1. Otherwise, we include this set for evaluation. Then, this set serves as a basis for the next new set; we create a new set by adding a new task into the old set and return to Step 2.

For any given m, 100,000 task sets are created for each task utilization model, thus generating 1,000,000 task sets in total for simulation.

5.2 Schedulability Performance and Time Complexity

We evaluate the performance of three EDZL schedulability tests: 1) the existing basic EDZL test in Lemma 2 with $S_i = 0$, 2) the existing improved EDZL test in Lemma 2 with S_i assignment according to [9, Fig. 3], and 3) our proposed EDZL test in Theorem 2. These tests are, respectively, annotated as "BCB," "BCB-I," and "OURS." In addition to the three EDZL schedulability tests, we also include the EDF test in Lemma 5 (annotated by "GFB-EDF") since this test serves as a basis for OURS.

In addition, we have one additional curve: "Tot" represents the total number of task sets with each task set utilization (i.e., $\sum_{\tau_i \in \mathcal{T}} \delta_i$). Each plot in Figs. 2 and 3 shows the number of task sets proven schedulable by each test, with task set utilization in

$$\left[\sum_{\tau_i \in \mathcal{T}} \delta_i - 0.01 \cdot m, \sum_{\tau_i \in \mathcal{T}} \delta_i + 0.01 \cdot m\right).$$

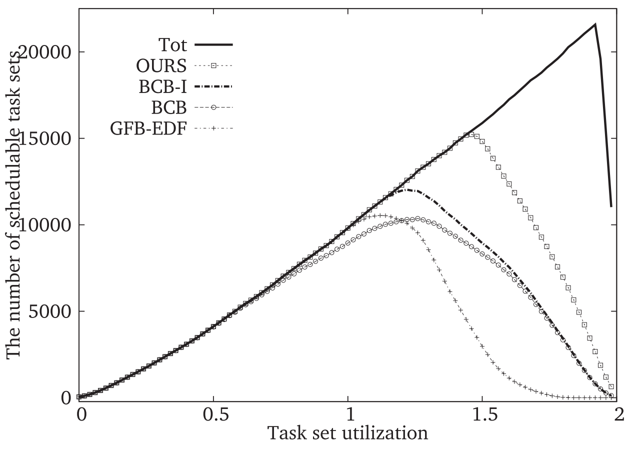
Figs. 2 and 3 show schedulability test results over varying task utilization models and different numbers of cores. We first show the results aggregated over all the task utilization distributions in Figs. 2a and 3a. We then show the results across individual task utilization models to investigate them more closely. Among the 10 task utilization models, we choose to show three representative models, where the average task utilization ($\overline{\delta_i}$) becomes the smallest, the medium, and the largest, respectively. In other words, in those three representative models, the average number of tasks ($\overline{n}$) is the largest, the medium, and the smallest, respectively. Those three models are exponential distribution with 0.1 (Figs. 2b and 3b), exponential distribution with 0.9 (Figs. 2c and 3c), and bimodal distribution with 0.9 (Figs. 2d and 3d), respectively. Among the different values of m, we choose to show the results for $m = 2$ and 4 since schedulability test results with different values of m exhibit similar behaviors.

First, we observe that OURS outperforms both BCB and BCB-I for all cases. On average, OURS finds 32.2 (for $m = 2$) and 27.5 percent (for $m = 4$) more schedulable task sets compared to BCB, as shown in Figs. 2a and 3a. Furthermore, OURS deems 22.7 (for $m = 2$) and 18.3 percent (for $m = 4$) additional task sets schedulable, which are deemed not schedulable by BCB-I. On the other hands, there are only a few task sets (less than 0.2 percent in any case) which are deemed schedulable by BCB or BCB-I, but not schedulable by OURS. One example of such a task set is $\tau \triangleq \{\tau_1(2,1,2), \tau_2(2,1,2), \tau_3(7,1,7), \tau_4(8,3,8)\}$ on a 2-core platform; the set is deemed unschedulable by OURS, but schedulable by BCB-I.

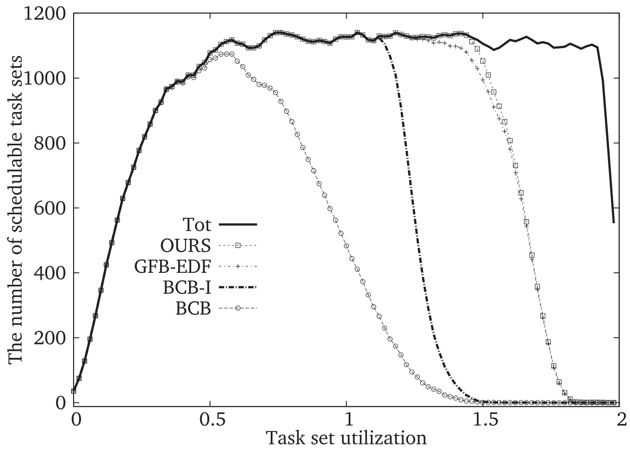
However, the degree of improvement may vary with characteristics of task sets; the improvement of OURS is

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

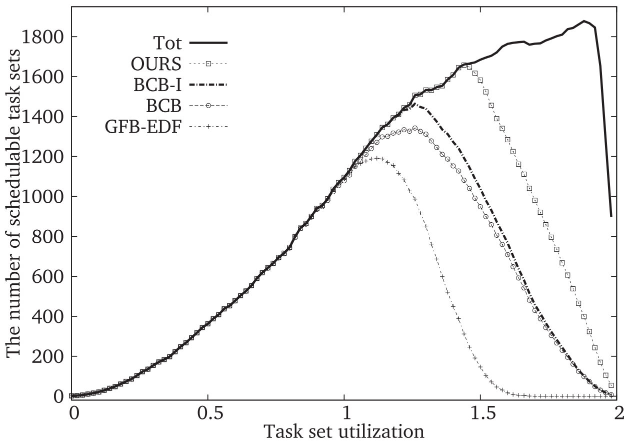
4. For a given exponential parameter $1/\lambda$, a value for δ_i is chosen according to an exponential distribution whose probability density function is $\lambda \cdot \exp(-\lambda \cdot x)$.



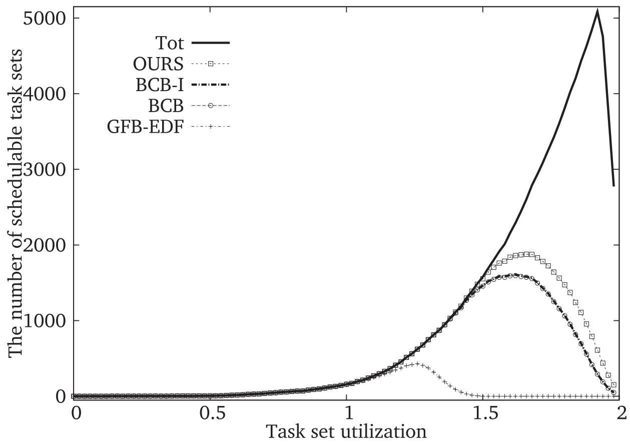
(a) Aggregate of all distributions ($\overline{n} = 5.1$, $\overline{\delta_i} = 0.27$, $\overline{\delta_{max}} = 0.60$)



(b) Exponential distribution with 0.1 ($\overline{n} = 11.6$, $\overline{\delta_i} = 0.09$, $\overline{\delta_{max}} = 0.28$)



(c) Exponential distribution with 0.9 ($\overline{n} = 4.3$, $\overline{\delta_i} = 0.32$, $\overline{\delta_{max}} = 0.64$)



(d) Bimodal distribution with 0.9 ($\overline{n} = 3.1$, $\overline{\delta_i} = 0.55$, $\overline{\delta_{max}} = 0.78$)

Fig. 2. Schedulability performance of schedulability tests for $m = 2$.

more pronounced when $\overline{\delta_i}$ is small, as shown in Figs. 2b and 3b (up to 91.4 and 36.6 percent over BCB and BCB-I, respectively), and is less significant when $\overline{\delta_i}$ is large, as shown in Figs. 2d and 3d. This is because BCB and BCB-I calculate the blocking time of a job of a task by simply adding worst-case blocking time of jobs of all other tasks; since such worst cases do not necessarily occur at the same time, the more tasks the more pessimism. However, OURS does not result in such pessimism in that OURS may depend only on $\sum_{\tau_i \in \mathcal{T}} \delta_i$ and $\max_{\tau_i \in \mathcal{T}} \delta_i$, but does not necessarily depend on the number of tasks. In fact, OURS alleviates the dependency of $\max_{\tau_i \in \mathcal{T}} \delta_i$ because OURS can exclude the effect of at most $m - 1$ tasks on schedulability by setting m' ($1 \leq m' \leq m$) for (4).

Another observation is that the schedulability performance of GFB-EDF highly depends on $\max_{\tau_i \in \mathcal{T}} \delta_i$, which is in accord with (3) itself. When $\overline{\delta_{max}}$ (the average value of $\max_{\tau_i \in \mathcal{T}} \delta_i$) is small, GFB-EDF even outperforms the existing EDZL schedulability tests of BCB and BCB-I, as shown in Figs. 2b and 3b. However, as $\overline{\delta_{max}}$ gets larger, the schedulability performance of GFB-EDF significantly gets lower, as shown in Figs. 2c and 2d and 3c and 3d. On the other hand, OURS, although derived from GFB-EDF, does not highly depend on $\max_{\tau_i \in \mathcal{T}} \delta_i$, as shown in the figures, since OURS can be independent of at most $m - 1$ tasks, as we mentioned in the previous paragraph.

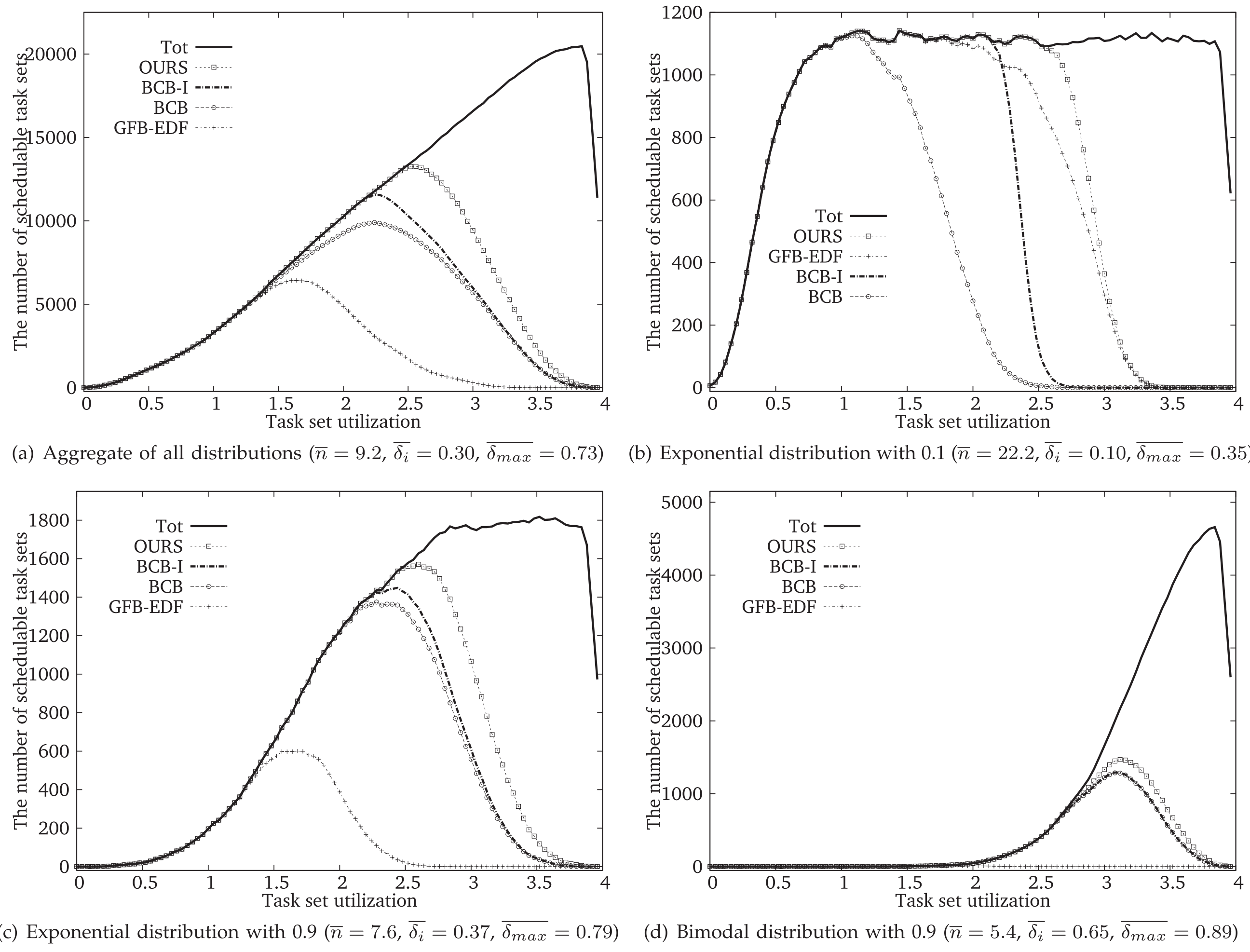
Now, we derive the time-complexity of OURS. To calculate (4) for a given m' requires $O(|\mathcal{T}|)$ computations, and in the worst case, (4) can be tested for all $1 \leq m' \leq m$. Therefore, OURS has $O(|\mathcal{T}| \cdot m)$ time complexity. On the other hand, BCB and BCB-I have $O(|\mathcal{T}|^2)$ and $O(|\mathcal{T}|^3 \cdot \max_{\tau_i \in \mathcal{T}} T_i)$ time complexity, respectively [9]. Since we assume $|\mathcal{T}| > m$ (otherwise, $\mathcal{T}$ is trivially schedulable), OURS has lower time complexity than both BCB and BCB-I.

In summary, OURS not only requires fewer computations than BCB and BCB-I, but also significantly improves the schedulability of EDZL by overcoming the weak points of BCB and BCB-I.

6 Related Work

As we discussed in the beginning of Section 2.2, the basic and improved tests in [8], [9] are the only existing schedulability tests for EDZL. However, the analysis technique used in this paper is related to previous studies for different scheduling algorithms, which will be discussed in this section.

In this paper, Theorem 1 presented a key property of EDZL, which is a basis for the new schedulability test of EDZL: If $\mathcal{T}_1$ is schedulable by EDZL on an m-core platform, $\mathcal{T}_1 \cup \mathcal{T}_2$ is schedulable by EDZL on an $(m + |\mathcal{T}_2|)$-core platform. Here, the most important point is that we do not



(a) Aggregate of all distributions ($\overline{n} = 9.2$, $\overline{\delta_i} = 0.30$, $\overline{\delta_{max}} = 0.73$) (b) Exponential distribution with 0.1 ($\overline{n} = 22.2$, $\overline{\delta_i} = 0.10$, $\overline{\delta_{max}} = 0.35$)

(c) Exponential distribution with 0.9 ($\overline{n} = 7.6$, $\overline{\delta_i} = 0.37$, $\overline{\delta_{max}} = 0.79$) (d) Bimodal distribution with 0.9 ($\overline{n} = 5.4$, $\overline{\delta_i} = 0.65$, $\overline{\delta_{max}} = 0.89$)

Fig. 3. Schedulability performance of schedulability tests for $m = 4$.

need to care for the schedulability of tasks in $\mathcal{T}_2$ as long as additional $|\mathcal{T}_2|$ cores are provided. A similar property has been identified for some scheduling algorithms such as EDF-US [21], EDF$^{(k)}$ [10], and fpEDF [22]. Those studies share an approach of task-level priority promotion. That is, they assign the highest task-level fixed-priority to at most $m - 1$ tasks. This way, a clear separation is made statically between at most $m - 1$ highest priority tasks and the other remaining lower priority tasks. This allows us to carry out schedulability analysis for the remaining tasks in isolation, excluding at most $m - 1$ highest priority tasks with the same number of cores.

However, despite potential improvement of schedulability tests, it has not been proven that such a technique is applicable to the scheduling algorithms that do not employ such task-level priority promotion. For example, EDZL promotes the priorities of jobs dynamically according to their laxity values. Thereby, the contribution of this paper is to extend the technique of analyzing the schedulability of a task subset in isolation toward the job-level dynamic-priority scheduling case from the task-level fixed-priority scheduling case. The technique, incorporated into the dominance relationship between EDZL and EDF, derived a new EDZL schedulability test based on a popular schedulability test for EDF, which is another contribution of this paper.

7 Conclusion

In this paper, we have developed a new schedulability test of EDZL. Compared to existing EDZL schedulability tests, the proposed test remarkably improves schedulability of implicit deadline task sets under EDZL with low time complexity.

Our future work includes development of new techniques to improve the schedulability of EDZL for more general task systems (e.g., arbitrary deadline task sets in which the relative deadline of each task is determined regardless of its period). This would require investigating new inherent EDZL properties for more general task systems and deriving efficient EDZL schedulability analysis based on the properties.

Acknowledgments

Insik Shin is the corresponding author. This work was supported in part by BRL (2009-0086964), BSRP (2010-0006650, 2012-R1A1A1014930), NCRC (2012-0000980), NIPA (H0503-12-1041), KEIT (2011-10041313), DGIST CPS Global Center, and KIAT (M002300089) funded by the Korea Government (MEST/MKE).

REFERENCES

[1] C. Liu and J. Layland, "Scheduling Algorithms for Multi-Programming in a Hard-Real-Time Environment," *J. ACM,* vol. 20, no. 1, pp. 46-61, 1973.

[2] C.-G. Lee, K. Lee, J. Hahn, Y.-M. Seo, S.L. Min, R. Ha, S. Hong, C.Y. Park, M. Lee, and C.S. Kim, "Bounding Cache-Related Preemption Delay for Real-Time Systems," *IEEE Trans. Software Eng.,* vol. 27, no. 9, pp. 805-826, Sept. 2001.

[3] L. Sha, T. Abdelzaher, K.-E. Arzen, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A.K. Mok, "Real Time Scheduling Theory: A Historical Perspective," *Real-Time Systems,* vol. 28, no. 2/3, pp. 101-155, 2004.

[4] S.K. Lee, "On-Line Multiprocessor Scheduling Algorithms for Real-Time Tasks," *Proc. IEEE Region 10's Ninth Ann. Int'l Conf.,* pp. 607-611, 1994.

[5] J. Lee, A. Easwaran, I. Shin, and I. Lee, "Zero-Laxity Based Real-Time Multiprocessor Scheduling," *J. Systems and Software,* vol. 84, no. 12, pp. 2324-2333, 2011.

[6] S. Cho, S.-K. Lee, S. Ahn, and K.-J. Lin, "Efficient Real-Time Scheduling Algorithms for Multiprocessor Systems," *IEICE Trans. Comm.,* vol. E85-B, no. 12, pp. 2859-2867, 2002.

[7] M. Park, S. Han, H. Kim, S. Cho, and Y. Cho, "Comparison of Deadline-Based Scheduling Algorithms for Periodic Real-Time Tasks on Multiprocessor," *IEICE Trans. Information and Systems,* vol. E88-D, pp. 658-661, 2005.

[8] M. Cirinei and T.P. Baker, "EDZL Scheduling Analysis," *Proc. Euromicro Conf. Real-Time Systems,* pp. 9-18, 2007.

[9] T.P. Baker, M. Cirinei, and M. Bertogna, "EDZL Scheduling Analysis," *Real-Time Systems,* vol. 40, pp. 264-289, 2008.

[10] J. Goossens, S. Funk, and S. Baruah, "Priority-Driven Scheduling of Periodic Task Systems on Multiprocessors," *Real-Time Systems,* vol. 25, no. 2/3, pp. 187-205, 2003.

[11] A. Mok, "Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment," PhD dissertation, Massachusetts Inst. of Technology, 1983.

[12] 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," *Proc. IEEE Int'l Conf. Embedded and Real-Time Computing Systems and Applications,* pp. 120-130, 2007.

[13] Y.-H. Chao, S.-S. Lin, and K.-J. Lin, "Schedulability Issues for EDZL Scheduling on Real-Time Multiprocessor Systems," *Information Processing Letters,* vol. 107, pp. 158-164, 2008.

[14] S. Kato and N. Yamasaki, "Global EDF-Based Scheduling with Efficient Priority Promotion," *Proc. IEEE Int'l Conf. Embedded and Real-Time Computing Systems and Applications,* pp. 197-206, 2008.

[15] M. Bertogna, M. Cirinei, and G. Lipari, "Improved Schedulability Analysis of EDF on Multiprocessor Platforms," *Proc. Euromicro Conf. Real-Time Systems,* pp. 209-218, 2005.

[16] T.P. Baker, "Comparison of Empirical Success Rates of Global vs. Paritioned Fixed-Priority and EDF Scheduling for Hand Real Time," Technical Report TR-050601, Dept. of Computer Science, Florida State Univ., Tallahasee, 2005.

[17] M. Bertogna, M. Cirinei, and G. Lipari, "Schedulability Analysis of Global Scheduling Algorithms on Multiprocessor Platforms," *IEEE Trans. Parallel and Distributed Systems,* vol. 20, no. 4, pp. 553-566, Apr. 2009.

[18] J. Lee, A. Easwaran, and I. Shin, "Laxity Dynamics and LLF Schedulability Analysis on Multiprocessor Platforms," *Real-Time Systems,* vol. 48, no. 6, pp. 716-749, 2012.

[19] J. Lee, A. Easwaran, and I. Shin, "Maximizing Contention-Free Executions in Multiprocessor Scheduling," *Proc. IEEE Real-Time Technology and Applications Symp.,* pp. 235-244, 2011.

[20] S. Baruah, N.K. Cohen, C.G. Plaxton, and D.A. Varvel, "Proportionate Progress: A Notion of Fairness in Resource Allocation," *Algorithmica,* vol. 15, no. 6, pp. 600-625, 1996.

[21] A. Srinivasan and S. Baruah, "Deadline-Based Scheduling of Periodic Task Systems on Multiprocessors," *Information Processing Letters,* vol. 84, no. 2, pp. 93-98, 2002.

[22] S.K. Baruah, "Optimal Utilization Bounds for the Fixed-Priority Scheduling of Periodic Task Systems on Identical Multiprocessors," *IEEE Trans. Computers,* vol. 53, no. 6, pp. 781-784, June 2004.