The Journal of Systems and Software 101 (2014) 15–29

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Capturing urgency and parallelism using quasi-deadlines for real-time multiprocessor scheduling ☆

Hoon Sung Chwa [a], Hyoungbu Back [a], Jinkyu Lee [b], Kieu-My Phan [c], Insik Shin [a,*]

[a] Department of Computer Science, KAIST, Daejeon, South Korea
[b] Department of Computer Science and Engineering, Sungkyunkwan University, Suwon, South Korea
[c] PRECISE Center, University of Pennsylvania, Philadelphia, PA 19104, USA

ARTICLE INFO

Article history:
Received 20 December 2013
Revised 7 November 2014
Accepted 9 November 2014
Available online 20 November 2014

Keywords:
Real-time systems
Multiprocessor scheduling
Quasi-deadline

ABSTRACT

Recent trends toward multi-core architectures in real-time embedded systems pose challenges in designing efficient real-time multiprocessor scheduling algorithms. We believe that it is important to take into consideration both timing constraints of tasks (urgency) and parallelism restrictions of multiprocessor platforms (parallelism) together when designing scheduling algorithms. Motivated by this, we define the *quasi-deadline* of a job as a weighted sum of its absolute deadline (capturing urgency) and its worst case execution time (capturing parallelism) with a system-level control knob to balance urgency and parallelism effectively. Using the quasi-deadline to prioritize jobs, we propose two new scheduling algorithms, called EQDF (earliest quasi-deadline first) and EQDZL (earliest quasi-deadline until zero laxity), that are categorized into job-level fixed-priority (JFP) scheduling and job-level dynamic-priority (JDP) scheduling, respectively. This paper provides a new schedulability analysis for EQDF/EQDZL scheduling and addresses the problem of priority assignment under EQDF/EQDZL by determining a right value of the system-level control knob. It presents optimal and heuristic solutions to the problem subject to our proposed EQDF and EQDZL analysis. Our simulation results show that EQDF and EQDZL can improve schedulability significantly compared to EDF and EDZL, respectively.

1. Introduction

Multi-core architectures have been increasingly adopted in safety-critical, real-time embedded systems to address ever-increasing performance requirements. For example, multi-cores have been growingly deployed in a variety of robots, and AUTOSAR has added support for multi-core architectures in the automotive domain (AUTOSAR, 2009). As a result, real-time scheduling research has been steadily gaining importance. Real-time scheduling determines the order of execution of jobs in order to satisfy their own timing constraints (i.e., deadlines). Two fundamental problems are the focus of most research in this area: algorithm design that aims to derive task and job priorities so as to satisfy all deadlines, and schedulability analysis that aims to provide guarantees of deadline satisfaction.

Over several decades, real-time scheduling has been extensively studied over various scheduling categories. In general, priority-driven preemptive scheduling algorithms fall into one of the following three categories:

- A *task-level fixed-priority (TFP)* algorithm assigns the same priority to all the jobs in each task, and the priority of each task is fixed relative to other tasks. Good examples include RM (rate-monotonic) (Liu and Layland, 1973) and DM (deadline-monotonic) (Leung and Whitehead, 1982).
- A *job-level fixed-priority (JFP)*[1] algorithm can assign different priorities to the individual jobs in each task, but the priority of each individual job is fixed relative to other jobs. A typical example is EDF (earliest deadline first) (Liu and Layland, 1973).
- A *job-level dynamic-priority (JDP)* algorithm assigns to a job a priority that can change dynamically during the job's execution. A good example is LST (least-slack-time first) (Leung, 1989).

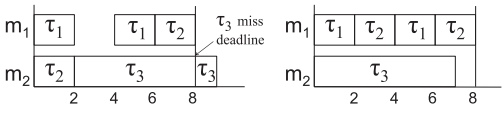
In the uniprocessor case, real-time scheduling has been well understood in each of the above three categories with successful

☆ This paper is an extended version of the RTAS 2012 (the 18th IEEE Real-Time and Embedded Technology and Applications Symposium) paper (Back et al., 2012).

* Corresponding author. Tel.: +82 42 350 3524.
E-mail address: insik.shin@cs.kaist.ac.kr, insik.shin@gmail.com (I. Shin).

[1] This category is also called task-level dynamic-priority in the literature (Liu, 2000). However, we use the term JFP in order to emphasize the static nature of the priority of an individual job.

http://dx.doi.org/10.1016/j.jss.2014.11.019
0164-1212/



(a) EDF schedule (b) Successful schedule

Fig. 1. A set of three tasks is feasible on two processors, but EDF fails to schedule them successfully: $\tau_1 = \tau_2 = (4, 2, 4)$, and $\tau_3 = (8, 7, 8)$, where (T_i, C_i, D_i) specifies the minimum separation, the worst-case execution time, and the relative deadline, respectively.

results, including the optimality of DM, EDF, and LST in each category, respectively. However, such successful results do not simply extend to the multiprocessor case. For example, EDF is no longer optimal but exhibits significantly lower performance in multiprocessor scheduling (Baker, 2003; Goossens et al., 2003), yet with little work reported to explore JFP scheduling beyond EDF on multiprocessors. Under JDP scheduling, several optimal scheduling algorithms have been proposed for a basic task model (i.e., for periodic implicit-deadline tasks in which deadlines are equal to periods). However, such optimality results do not hold for general task models (i.e., for periodic/sporadic constrained-deadline tasks in which deadlines are no larger than periods). This motivates the research described in this paper to design new scheduling algorithms that advance the state-of-the-art in JFP and JDP multiprocessor scheduling, in particular, for sporadic constrained-deadline tasks.

Quasi-deadline. We believe that deadline-based scheduling algorithms (e.g., EDF) perform poorly on multiprocessors because they assign priority with a sole focus on deadline constraints (or "urgency") but neglecting "parallelism" restrictions on multiprocessor platforms. The parallelism restriction makes a task unable to run simultaneously on more than one processor, therefore the task cannot fully use all processors even though more than one are available at the same time. This can lead to a deadline miss of a task even though a total amount of available processor capacity is larger than what the task requires, as shown in Fig. 1(a). Such a parallelism restriction naturally becomes more severe with a growing execution time requirement. On the other hand, a task with unitary minimum execution time requirement is free from such a parallelism restriction. For example, EDF becomes optimal even on multiprocessors if all tasks have the execution time requirement of one unit (Lee et al., 2011b). A few studies (Andersson and Jonsson, 2000; Davis and Burns, 2009; Erickson and Anderson, 2012; Lee et al., 2011b) show that assigning higher priorities to jobs with larger execution time requirements helps to mitigate parallelism restrictions. For non-real-time tasks, the LEF (largest execution time first) scheduling policy is proven to minimize the makespan thus offering the maximum room for future execution. As such, the execution time requirement is considered as effective in capturing the parallelism restriction, and it entails scheduling policies that can consider both urgency and parallelism simultaneously. Inspired by this, we define the *quasi-deadline* (q_i) of an individual job (J_i) as a weighted sum of its absolute deadline (d_i) and its worst-case execution time (C_i) such that $q_i = d_i - k \cdot C_i$, where k is a real number (i.e., $k \in \mathbb{R}$) that the system statically configures for a given task set. The parameter k allows efficient balancing between urgency (captured by d_i) and parallelism (captured by C_i). We then present two new algorithms, *EQDF* (earliest quasi-deadline first) and *EQDZL* (earliest quasi-deadline until zero laxity), that assigns priorities based on quasi-deadlines. We note that the quasi-deadline is a metric that is only used for prioritizing jobs and it does not change any of the original task specification, i.e., jobs are still required to complete by their original deadlines.

EQDF. We first introduce a new job-level fixed-priority scheduling algorithm, called *EQDF* (earliest quasi-deadline first), that assigns priority to jobs according to their quasi-deadlines. EQDF is a generalization of EDF; EQDF becomes EDF with $k = 0$. Then, EQDF is able to yield better schedules than EDF by properly assigning the parameter k; for example, a task set in Fig. 1, which is not schedulable by EDF, is schedulable by EQDF with $k = 1$, as shown in Fig. 1(b).

For the proposed EQDF scheduling, this paper derives new schedulability conditions and addresses quasi-deadline assignment, which is critical to the effectiveness of EQDF. In particular, this paper considers the k-controlled quasi-deadline assignment that determines the value of k to cause a task set to become feasible according to the proposed EQDF schedulability test. A naive approach of examining all possible values of k is prohibitively expensive and even inapplicable to continuous values of k. We thereby present an optimal solution algorithm, called OQDA-k, that finds a feasible value for k, if any exists. Our empirical results show that the proposed EQDF optimal solution not only dominates EDF but also outperforms it significantly. Our EQDF solutions find 40–45% more schedulable task sets than the state-of-the-art EDF analysis. Our empirical results also reveal that the OQDA-k algorithm employs a considerable running time, leaving the algorithm only suitable for design time. Thereby, we present a heuristic solution to the problem as well. A key factor to performance is where and how densely the heuristic algorithm examines k values. Based on thorough understanding of empirical results, we are able to reduce the search space of the heuristic solution quite effectively. Our simulation results show that the proposed heuristic algorithm can find a solution close to optimal (less than 1% loss of optimality) while reducing running time by two orders of magnitude. It is also shown that the heuristic algorithm is able to find 34–37% more schedulable task sets than EDF with a comparable running time.

EQDZL. Building upon such EQDF results, this paper also seeks to explore the effectiveness of quasi-deadline under deadline-based job-level dynamic-priority scheduling. EDZL (earliest deadline until zero laxity) (Lee, 1994) has been introduced as an extension of EDF. Laxity at time t is defined as remaining time to deadline at t ($D_i(t)$) minus the amount of remaining execution at t ($C_i(t)$). EDZL assigns priorities to jobs according to EDF if those jobs do not reach zero laxity or gives the highest priority to a job if it goes to zero laxity. This way, EDZL considers only urgency when no job reaches zero laxity, but starts considering both urgency and parallelism together when a job enters the zero laxity state. Several studies (Baker et al., 2008; Lee et al., 2011b; Lee, 1994) reported that such a zero-laxity-based extension toward EDZL brings significant improvement in schedulability. Given the superiority of EDZL, a question is naturally raised whether incorporating quasi-deadlines into EDZL can improve schedulability any further. To answer the question, this paper extends EDZL to EQDZL (earliest quasi-deadline until zero laxity) and derives EQDZL schedulability tests from EQDF tests. It then considers the problem of finding an optimal value of k subject to the proposed EQDZL test and presents an optimal solution, called OQDA^{ZL}-k, reflecting the unique features of zero laxity. Our simulation results show that the OQDA^{ZL}-k algorithm can improve schedulability by 10% compared to EDZL. Given the high time-complexity of the OQDA^{ZL}-k algorithm, we also present a heuristic solution. According to our simulation results, the heuristic solution is quite effective in finding sub-optimal solutions (1% loss of optimality) while reducing running time by three to four orders of magnitude.

Contribution. This paper extends our work presented earlier in a conference publication (Back et al., 2012). First, this paper elaborates on the concept of quasi-deadline by explaining how the concept captures urgency and parallelism, the most two important aspects of real-time multiprocessor scheduling. Second, in addition to the EQDF algorithm proposed in Back et al. (2012), this paper proposes a new EQDZL algorithm, demonstrating and evaluating the applicability of quasi-deadline in real-time multiprocessor scheduling. The EQDZL algorithm is able to bring a significant improvement in schedulability for general task models. Third, this paper also broadens the related work discussion and expands the evaluation with more simulation results.

The rest of this paper is organized as follows. Section 2 presents related work, and Section 3 describes system models and terminology. Section 4 derives new, interference-based schedulability tests for EQDF and EQDZL scheduling based on the understanding of worst-case inter-task interference scenarios. Section 5 formulates the k-controlled quasi-deadline assignment problem and proposes optimal and heuristic solutions. Section 6 provides empirical results with investigation of the characteristics of optimal solutions thoroughly and evaluates the effectiveness of our EQDF and EQDZL solutions. Section 7 concludes and points out future work.

2. Related work

A considerable amount of work has been made to study TFP multiprocessor scheduling. Many policies were proposed for priority assignment in this category, including RM-US{θ} (Andersson et al., 2001), DM-DS{θ} (Bertogna et al., 2005b), and SM-US{θ} (Andersson, 2008). Those policies share the same principle that they assign the highest priority to tasks with utilization (or density) greater than a threshold θ. They then differ from each other in that they assign priorities to the other tasks according to the rules of rate-monotonic (RM-US{θ}), deadline-monotonic (DM-DS{θ}), and slack-monotonic (SM-US{θ}), respectively. Audsley (1991, 2001) developed an optimal priority assignment (OPA) policy for some given schedulability test on uniprocessor platforms. Davis and Burns (2009) showed that Audsley's OPA algorithm is applicable to the multiprocessor case when a given test satisfies some conditions. There have been some task-level priority assignment schemes that are related to our notion of quasi-deadline. Andersson and Jonsson (2000) designed the *TkC* priority assignment policy which assigns priorities based on $(T_i - k \cdot C_i)$, where T_i is a task period and k is a real value computed on the basis of the number of processors. Davis and Burns (2009) developed the D-CMPO policy that assigns priorities according to $D_i - C_i$, where D_i is a task's relative deadline.

EDF is the most studied and well-known JFP scheduling algorithm. Several schedulability tests for global EDF scheduling of sporadic task systems have been developed (Baker, 2003; Baker and Baruah, 2009a; Baruah, 2007; Baruah et al., 2009; Bertogna and Cirinei, 2007; Bertogna et al., 2005a, 2009; Goossens et al., 2003). Baker (2003) developed a general strategy using the notion of processor load for determining the schedulability of sporadic task sets. Baker's test computes a necessary amount of processor load to cause a deadline to be missed and takes the contraposition of this to derive a sufficient schedulability test. Building upon Baker's work, Bertogna et al. (2005a, 2009) developed a sufficient schedulability test for any work-conserving algorithms based on bounding the maximum workload in a given interval. Bertogna et al. extended this test via an iterative schedulability test that calculates a slack value for each task, and then uses this value to limit the amount of carry-in workloads. In addition to these studies, many studies have reduced the pessimism of calculating the maximum workload using their own techniques (Baker and Baruah, 2009a; Baruah, 2007; Baruah et al., 2009; Goossens et al., 2003). Empirical schedulability performance of the existing EDF schedulability tests has been evaluated in Bertogna and Baruah (2011). A recent study (Erickson and Anderson, 2012) introduces a priority assignment scheme that is closely related to our notion of quasi-deadline, considering both deadline and execution time together for JFP scheduling. This study suggests that each task τ_i is assigned the priority of $d_i - ((m-1)/m) \cdot C_i$, which uses a particular value for k (i.e., equal to $(m-1)/m$) for all task set instances. It shows that the proposed priority assignment brings a maximum tardiness bound that is no greater than that of global EDF scheduling for soft real-time systems.

There has been a growing attention to JDP multiprocessor scheduling. In the implicit-deadline task systems, where each task has a deadline equal to task period, a class of optimal algorithms (Anderson and Srinivasan, 2000; Baruah et al., 1996; Cho et al., 2006; Funaoka et al., 2008; Levin et al., 2010; Regnier et al., 2011) is proposed. However, such optimality does not extend to more general, constrained-deadline task systems, where a task has a deadline larger than or equal to a task period. For example, those optimal algorithms in the implicit-deadline task case no longer guarantee the schedulability of the constrained-deadline task systems, when the system density[2] exceeds the number of processors (Lee et al., 2012). Recent studies showed that zero-laxity based algorithms (Lee et al., 2011b), which assign the highest priority to zero-laxity jobs, are quite effective in multiprocessor scheduling and able to guarantee the schedulability of the system even when the system density is greater than the number of processors. These algorithms include EDZL (EDF until zero laxity) (Baker et al., 2008; Lee, 1994), FPZL (fixed-priority until zero laxity) (Davis and Kato, 2012; Davis and Burns, 2011), and LST (Lee et al., 2010; Leung, 1989). In general, JDP algorithms incur relatively significant runtime scheduling overheads, for examples, with frequent context switches and/or with keeping track of laxity dynamically, compared to TFP and JFP algorithms.

3. System model and terminology

Task model. In this paper, we assume a sporadic task model, where a task $\tau_i \in \tau$ is specified as (T_i, C_i, D_i) such that T_i is the minimum separation, C_i is the worst-case execution time requirement, and D_i is the relative deadline. Further, we focus our attention on constrained $(C_i \le D_i \le T_i)$ deadline tasks. Let n denote the number of tasks in τ. A task utilization U_i of τ_i is defined as C_i/T_i, and the system utilization U_{sys} is defined as the total utilization of a task set, i.e., $U_{sys} = \sum_{\tau_i \in \tau} U_i$. The density of a task is defined as $\delta_i = C_i/D_i$, and the system density δ_{sys} is given by $\sum_{\tau_i \in \tau} \delta_i$. A task τ_i invokes a series of jobs, each separated from its predecessor by at least T_i time units. A single job is a unit of execution that can only be allocated to a single processor at a time. When a job J_i^h of task τ_i has a release time r_i^h, its absolute deadline d_i^h is given as $d_i^h = r_i^h + D_i$. The *scheduling window* of a job J_i^h is then defined as the interval between its release time and deadline $[r_i^h, d_i^h)$. We define the *quasi-deadline* q_i^h of a job J_i^h as $q_i^h = d_i^h - k \cdot C_i$, where k is a knob that controls the ratio of execution time to deadline. We assume that the *quasi-deadline control knob* k is a real number (i.e., $k \in \mathbb{R}$), and the system statically configures the value of k for a given task set. We also assume quantum-based time and without loss of generality, let one time unit denote the quantum length. All task parameters are assumed to be specified as multiples of this quantum length.

Multiprocessor scheduling. We assume that the system consists of m identical unit-capacity processors, and we consider global preemptive scheduling on multiprocessors. In particular, we focus on two multiprocessor scheduling algorithms with quasi-deadlines: EQDF and EQDZL.

First, the EQDF scheduling algorithm assigns the priority of jobs according to their quasi-deadlines; a job with an earlier quasi-deadline has a higher priority. Compared to EDF, EQDF requires only one extra calculation of quasi-deadline per job (i.e., $q_i = d_i - k \cdot C_i$). Thereby, the operating cost of EQDF is comparable to that of EDF. When it comes to preemption, it has been reported that a single job can preempt other jobs at most once under EDF scheduling (Liu, 2000). In fact, this bound is applicable to all task-level and job-level fixed-priority scheduling algorithms. This is because a single job can preempt another when it is released and is unable to preempt any more as long as its priority remains the same. Hence, EQDF, which is a JFP algorithm, has the same preemption bound as that of EDF.

Second, we consider the EQDZL scheduling algorithm, which is a zero-laxity based extension of EQDF; zero-laxity jobs have the highest

[2] The system density (δ_{sys}) often serves as a measure of the overall processing demands of the systems, which will be formally defined in Section 3.

priority, and other jobs are prioritized by EQDF. Then, similar to the relationship between EQDF and EDF, EQDZL is comparable to EDZL in terms of operating cost and preemption overhead. Like EDZL (Baker and Baruah, 2009b), EQDZL may incur at most one more preemption per job than EQDF when the job enters the zero-laxity state.

In this paper, the costs of job preemption and migration are not directly incorporated into schedulability conditions, as in many other multiprocessor scheduling studies (for example, see Baruah et al., 1996; Cho et al., 2006; Levin et al., 2010; Regnier et al., 2011)[3].

4. Schedulability analysis of EQDF and EQDZL

In this section, we first derive schedulability conditions for the EQDF and EQDZL scheduling algorithms and then present how to exploit slack values for better schedulability of EQDF and EQDZL.

4.1. EQDF schedulability analysis

4.1.1. Interference-based schedulability condition

Existing studies (Baker, 2003; Baker et al., 2008; Bertogna et al., 2005a, 2009; Lee et al., 2010) on multiprocessor global schedulability analysis investigate what happens when the first deadline miss occurs and derive schedulability conditions using the concept of interference—how long the execution of a job of interest can be delayed due to the execution of other higher-priority jobs. The *total interference* on a task τ_j in an interval $[a, b)$ is defined by the cumulative length of all intervals in which τ_j is ready to execute but is not executing due to higher priority jobs of other tasks. We denote such interference with $I_j(a, b)$. We also define the *interference* $I_{j\leftarrow i}(a, b)$ of a task τ_i on a task τ_j over an interval $[a, b)$, as the cumulative length of all intervals in which τ_j is ready to execute but it is not executing since τ_i is executing instead. The relation between $I_j(a, b)$ and $I_{j\leftarrow i}(a, b)$ is as follows (Bertogna et al., 2005a):

$$I_j(a, b) = \frac{\sum_{i\neq j} I_{j\leftarrow i}(a, b)}{m}. \tag{1}$$

The basic strategy used in Baker (2003) and Bertogna et al. (2009) is identifying necessary conditions for a job to miss its deadline. Generally, a deadline miss happens since there is a large amount of higher-priority execution that blocks the remaining execution of a job until its deadline. Let us assume that a job of a task τ_j is the first job that misses its deadline. In order for the job to miss its deadline, it is necessary for the job to be blocked for strictly more than its slack time ($D_j - C_j$ time units) in its scheduling window. On the other hand, a job of a task τ_j always meets its deadline, if the total interference on task τ_j over the job's scheduling window is less than or equal to $D_j - C_j$. Let J_j^* denote the job instance that receives the largest amount of interference among all jobs invoked by τ_j, and $\bar{I}_j$ denote the amount of interference that J_j^* receives. Then, notice that by definition

$$\bar{I}_j = \max_h \left(I_j(r_j^h, d_j^h)\right) = I_j(r_j^*, d_j^*). \tag{2}$$

For notational convenience, we also define

$$\bar{I}_{j\leftarrow i} = I_{j\leftarrow i}(r_j^*, d_j^*). \tag{3}$$

If we are able to precisely calculate $\bar{I}_{j\leftarrow i}$, the necessary and sufficient schedulability condition of global multiprocessor scheduling algorithms is then derived as follows (Bertogna et al., 2005a, 2009).

[3] In the uniprocessor scheduling case, some studies (Bertogna et al., 2010, 2011) are introduced to incorporate preemption cost into schedulability analysis. However, little work has been made to incorporate preemption and migration costs directly into schedulability analysis for multiprocessor scheduling, and this issue is beyond the scope of this paper. Preemption and migration overhead measurements can be accommodated into the worst-case execution time requirements, as mentioned in Levin et al. (2010) and Regnier et al. (2011).

Lemma 1 (from Bertogna et al., 2005a, 2009). *A task set τ is schedulable on a multiprocessor composed of m identical processors, if and only if the following condition holds for each task $\tau_j \in \tau$:*

$$\sum_{i\neq j} \min(\bar{I}_{j\leftarrow i}, D_j - C_j + 1) < m \cdot (D_j - C_j + 1). \tag{4}$$

A key intuition behind Lemma 1 is as follows. By definition, when a job of τ_j is executing in an interval, it cannot be interfered with during the interval. If τ_j misses its deadline, the total interference on τ_j is more than or equal to $D_j - C_j + 1$, and all processors should be busy within each interval where τ_j is not executed. Then, it is sufficient for a task τ_j to miss its deadline if the interfering contribution $\bar{I}_{j\leftarrow i}$ of each task τ_i is at least $D_k - C_k + 1$ time units based on the assumption that a single job cannot be executed upon more than one processor in parallel. Conversely, if the sum of the interfering contributions of the other tasks τ_i is strictly less than $m \cdot (D_j - C_j + 1)$, task τ_j does not miss its deadline, which yields the schedulability condition shown in Eq. (4).

Note that it is difficult to compute $\bar{I}_{j\leftarrow i}$ precisely, so existing approaches (Bertogna et al., 2005a, 2009) use an upper bound on the interference, and therefore the test derived is changed to only a sufficient condition. Then, it is necessary to identify the *worst-case interference scenario* in which a task τ_i has the largest workload to interfere with a job of task τ_j. In the following two sub-sections, we identify two types of worst-case interference scenarios and derive an upper-bound on the interference under EQDF scheduling according to the value of k, which serves as a basis for assigning a schedulable value of k for a given task set (shown in Section 5).

4.1.2. Worst-case interference scenarios

In this sub-section, we identify the worst-case interference scenarios in which the interference of a task τ_i on the job J_j^* over the scheduling window of J_j^* is maximized under EQDF scheduling.

To simplify the presentation, a job is said to be a *carry-in* job of an interval $[a, b)$ if it has a release time before a but a deadline after a, a *body* job if it has a release time and a deadline within $[a, b)$, and a *carry-out* job if it has a release time within $[a, b)$ but a deadline after b.

Under EDF scheduling, the underlying principle behind its worst-case interference scenario is that only the carry-in and body jobs of τ_i over $[r_j^*, d_j^*)$ can interfere with J_j^* and the interference of those jobs can be maximized when they are periodically released and they execute as late as possible (i.e., moving their deadlines as late as possible), as long as their deadlines are in the scheduling window. This is because moving their deadlines later does not affect the interference of body jobs, but it can only increase (and cannot decrease) the interference of a carry-in job (Baker, 2003). Therefore, the worst-case interference scenario of τ_i is the one where one of its jobs has a deadline at the end of the scheduling window of J_j^*.

However, the above principle is not directly applicable to EQDF scheduling. Unlike EDF scheduling, even the carry-out job of τ_i can interfere with J_j^* under EQDF scheduling if the quasi-deadline of the carry-out job is earlier than or equal to that of J_j^*. Furthermore, the worst-case interference scenarios vary depending on the relationship between J_j^* and the carry-out job.

Let J_i^+ denote the last job of τ_i which interferes with J_j^*. By definition, the quasi-deadline of J_i^+ is smaller than or equal to that of J_j^*, implying, $d_i^+ - k \cdot C_i \leq d_j^* - k \cdot C_j$. The difference ϕ between the two deadlines can be upper bounded in only k and their execution times as follows:

$$\begin{aligned}\phi &= d_i^+ - d_j^* \\ &\leq k \cdot C_i - k \cdot C_j.\end{aligned} \tag{5}$$

Under the EDF scheduling, if the deadlines of jobs J_i^+ and J_j^* align, then J_i^+ can fully contribute its execution time to the interference on J_j^*. However under the EQDF scheduling, when the quasi-deadlines

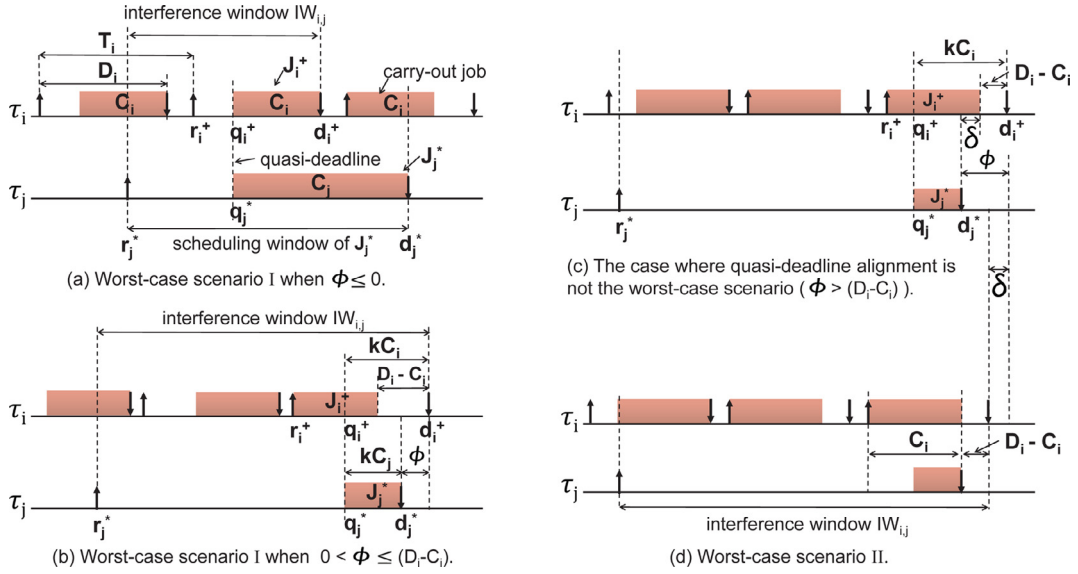


Fig. 2. EQDF worst-case scenarios with $k = 1$.

of J_i^+ and J_j^* align, J_i^+ may not necessarily fully contribute C_i into the interference on J_j^* because J_i^+ has to execute for at least $\delta > 0$ after the deadline of J_j^* as shown in Fig. 2(c). In other words, some execution of J_i^+ can be performed outside the scheduling window of J_j^*. Depending on whether or not J_i^+ can perform its full execution (amount to C_i) within the scheduling window of J_j^*, we consider the following two cases:

Worst-case scenario I. J_i^+ fully contributes C_i to the interference on τ_j when its quasi-deadline aligns with that of J_j^* (shown in Fig. 2(a) and (b)). This case happens when $(k \cdot C_i - k \cdot C_j) \le (D_i - C_i)$. The worst-case interference scenario is that the jobs of τ_i are periodically released in a way that one of the jobs has the same quasi-deadline as that of J_j^* (i.e., identical with quasi-deadline alignment).

Worst-case scenario II. J_i^+ does not fully contribute C_i into the interference on τ_j when its quasi-deadline aligns with that of J_j^* because J_i^+ has to execute for at least $\delta > 0$ after the deadline of J_j^* (shown in Fig. 2(c)). This case happens when $(k \cdot C_i - k \cdot C_j) > (D_i - C_i)$. By moving task τ_i to the left by δ, we can increase the interference of job J_i^+ by exactly δ while decreasing the interference of the carry-in job by at most δ. Thus the total amount of interference increases or at least stays the same. In this case, the worst-case interference scenario is that the jobs of τ_i are periodically released in a way that one of its jobs is released exactly C_i before the deadline of J_j^* (depicted in Fig. 2(d)).

4.1.3. Bounding interference

Define the *interference window* $IW_{i,j}$ of τ_i on J_j^* as the interval from the release time of J_j^* to the deadline of J_i^+, implying $IW_{i,j} = [r_j^*, d_i^+)$. Fig. 2 shows the interference window in each scenario. Note that under EDF scheduling, since the worst-case interference scenario is the one satisfying $d_i^+ = d_j^*$, $IW_{i,j}$ is exactly the same as $[r_j^*, d_j^*)$. Under EQDF scheduling, however, d_i^+ can be before or after, or equal to d_j^*.

The interference $\bar{I}_{j \leftarrow i}$ is then bounded by the largest workload of task τ_i in its interference window on J_j^* according to the worst-case scenarios.

In the worst-case scenario I, J_i^+ has the same quasi-deadline as that of J_j^*, so the deadline d_i^+ is determined by $d_j^* - k \cdot C_j + k \cdot C_i$. If the deadline of J_i^+ is before the release time of J_j^* (i.e., $d_i^+ \le r_j^*$), no single job of τ_i can interfere with J_j^* and the interference of τ_i on J_j^* is thereby zero. Otherwise, all the jobs of τ_i within the interference window $[r_j^*, d_i^+)$ can actually interfere with J_j^*. In this case, the length of the interference window $IW_{i,j}$ is computed as

$$\begin{aligned} |IW_{i,j}| &= d_i^+ - r_j^* = d_j^* - k \cdot C_j + k \cdot C_i - r_j^* \\ &= D_j - k \cdot C_j + k \cdot C_i. \end{aligned} \tag{6}$$

We denote by $\Phi_i(L)$ the maximal number of τ_i's jobs that contribute with the entire execution times (C_i) to the workload within the interval of length L, and it is described as

$$\Phi_i(L) = \left\lfloor \frac{L}{T_i} \right\rfloor. \tag{7}$$

The contribution of the carry-in job can then be bounded by

$$\min(C_i, L - \Phi_i(L) \cdot T_i). \tag{8}$$

Therefore, under the worst-case interference scenario I, the interference $\bar{I}_{j \leftarrow i}$ is bounded by

$$\begin{aligned} \bar{I}_{j \leftarrow i} \le \big[&\Phi_i(D_j - k \cdot C_j + k \cdot C_i) \cdot C_i \\ &+ \min(C_i, D_j - k \cdot C_j + k \cdot C_i - \Phi_i(D_j - k \cdot C_j + k \cdot C_i) \cdot T_i) \big]_0, \end{aligned} \tag{9}$$

where $[X]_Y \triangleq \max(X, Y)$, meaning that the RHS of Eq. (9) is bounded by zero when $D_j - k \cdot C_j + k \cdot C_i < 0$.

In the worst-case scenario II, J_i^+ is released such that its deadline d_i^+ is equal to $d_j^* - C_i + D_i$. Then, the length of the interference window of τ_i on J_j^* is equal to $D_j - C_i + D_i$. Therefore, the interference $\bar{I}_{j \leftarrow i}$ is bounded by

$$\begin{aligned} \bar{I}_{j \leftarrow i} \le\ &\Phi_i(D_j - C_i + D_i) \\ &\cdot C_i + \min(C_i, D_j - C_i + D_i - \Phi_i(D_j - C_i + D_i) \cdot T_i). \end{aligned} \tag{10}$$

4.1.4. EQDF schedulability analysis

In the previous sub-sections, we identified the worst-case interference scenarios and computed the upper bound on the interference $\bar{I}_{j \leftarrow i}$ based on them.

We denote by $I_{j \leftarrow i}^{\mathrm{EQDF}}(L, k)$ an upper bound on the interference $\bar{I}_{j \leftarrow i}$ in any interval of length L under the EQDF scheduling policy with a

system-wide variable k and described as

$$I_{j\leftarrow i}^{\mathrm{EQDF}}(L,k)=\begin{cases}[\Phi_i(L-k\cdot C_j+k\cdot C_i)\cdot C_i+\min(C_i,L-k\cdot C_j+k\cdot C_i \\ \quad-\Phi_i(L-k\cdot C_j+k\cdot C_i)\cdot T_i)]_0, \\ \quad \text{if}(k\cdot C_i-k\cdot C_j)\le(D_i-C_i), \\ \Phi_i(L+D_i-C_i)\cdot C_i+\min(C_i,L+D_i-C_i \\ \quad-\Phi_i(L+D_i-C_i)\cdot T_i), \quad \text{otherwise.}\end{cases} \tag{11}$$

We note that $I_{j\leftarrow i}^{\mathrm{EQDF}}(D_j,k)$ becomes zero when the value of k is $\frac{D_j}{C_j-C_i}$, and $I_{j\leftarrow i}^{\mathrm{EQDF}}(D_j,k)$ has the maximum value when the value of k is $\frac{D_i-C_i}{C_i-C_j}$ (i.e., the worst-case scenario II). Between those two values of k, $I_{j\leftarrow i}^{\mathrm{EQDF}}(D_j,k)$ is monotonically non-decreasing with k.

A schedulability test for EQDF immediately follows.

Theorem 1. *A task set τ is schedulable under EQDF scheduling with a system-wide variable k on a multiprocessor composed of m identical processors, if the following condition holds for each task $\tau_j\in\tau$:*

$$\sum_{i\neq j}\min\left(I_{j\leftarrow i}^{\mathrm{EQDF}}(D_j,k),D_j-C_j+1\right)<m\cdot(D_j-C_j+1). \tag{12}$$

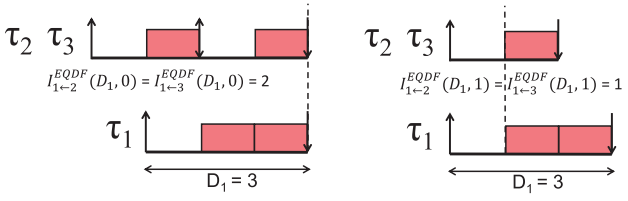
Proof. By Lemma 1, τ is schedulable under EQDF if Eq. (4) holds. We now prove that $\bar{I}_{j\leftarrow i}$ in Eq. (4) is upper-bounded by $I_{j\leftarrow i}^{\mathrm{EQDF}}(D_j,k)$ in Eq. (12).

By the definition of interference, a job can interfere with another job only when the interfering job is executed; more formally, the amount of interference of τ_i on a job of τ_j in an interval is upper-bounded by the amount of execution of jobs of τ_i having a higher priority than the job of τ_j in the interval. Therefore, the maximum amount of interference of τ_i on τ_j among all intervals of length D_j is also upper-bounded by the maximum amount of execution of τ_i among all intervals of length D_j. As shown in Eq. (11), $\bar{I}_{j\leftarrow i}$ is upper-bounded by $I_{j\leftarrow i}^{\mathrm{EQDF}}(D_j,k)$ in any interval of length D_j under the EQDF scheduling policy with a system-wide variable k.

Therefore, by Lemma 1, if Eq. (12) is satisfied for all tasks in a task set τ, the task set is schedulable under global EQDF scheduling on m identical processors. □

Note that the above EQDF schedulability test is a generalization of the existing EDF schedulability test (Bertogna et al., 2005a) in that the condition is equivalent to the EDF test when $k=0$.

Example 4.1. Let $\tau=\{\tau_1=(6,2,3),\tau_2=\tau_3=(2,1,2)\}$ and $m=2$. τ is not deemed schedulable under EDF by Theorem 1 when $k=0$, but it is deemed schedulable under EQDF with $k=1$ by the EQDF schedulability test in Theorem 1. As shown in Fig. 3(a), in the EDF case (when $k=0$), $I_{1\leftarrow 2}^{\mathrm{EQDF}}(D_1,0)=I_{1\leftarrow 3}^{\mathrm{EQDF}}(D_1,0)=2$, and Eq. (12) does not hold for τ_1. However, as shown in Fig. 3(b), in the EQDF schedulability test with $k=1$, $I_{1\leftarrow 2}^{\mathrm{EQDF}}(D_1,1)=I_{1\leftarrow 3}^{\mathrm{EQDF}}(D_1,1)=1$, and Eq. (12) holds for τ_1. When $k=1$, carry-in jobs of τ_2 and τ_3 cannot interfere with a job of task τ_1, which yields τ_1 deemed schedulable under EQDF scheduling.



(a) EDF schedulability test ($k=0$) (b) EQDF schedulability test ($k=1$)

Fig. 3. Illustration of Example 4.1.

4.2. EQDZL schedulability analysis

In this sub-section, we extend the EQDF schedulability analysis toward EQDZL. In order to derive an efficient EQDZL schedulability test, it is necessary to incorporate properties specific to zero-laxity. Under zero-laxity based scheduling, a job entering the zero-laxity state is assigned the highest priority. Thereby, a job misses its deadline, only if there are at least $m+1$ zero-laxity jobs at the same time. Using this property, the following schedulability test has been introduced for any zero-laxity based algorithms.

Lemma 2 (from Baker et al., 2008; Lee et al., 2010, 2011b)**.** *A task set τ is schedulable on a multiprocessor composed of m identical processors under a zero-laxity based algorithm, if at most m tasks $\tau_j\in\tau$ violate the following conditions:*

$$\sum_{i\neq j}\min(\bar{I}_{j\leftarrow i},D_j-C_j)<m\cdot(D_j-C_j). \tag{13}$$

A key intuition behind Lemma 2 is as follows. If Eq. (13) holds, a task τ_j cannot reach the zero-laxity state. So, as long as at most m tasks reach the zero-laxity state (violating Eq. (13)), the entire task set remains schedulable under any zero-laxity based algorithm.

In order to apply Lemma 2 to EQDZL, it is necessary to calculate an upper-bound on the interference $\bar{I}_{j\leftarrow i}$ under EQDZL scheduling. To this end, we seek to define $I_{j\leftarrow i}^{\mathrm{EQDZL}}(L,k)$ similarly to the EQDF case with a certain degree of difference to accommodate the zero-laxity-specific properties.

The priority of jobs under EQDZL is the same as that under EQDF, except the priority promotion of zero-laxity jobs. This way, a job can be interfered by zero-laxity jobs, in addition to the jobs with earlier quasi-deadlines. We consider two cases:

When $\phi\le 0$, the carry-out job has larger quasi-deadline than J_j^*. As shown in Fig. 2(a), the carry-out job, having larger quasi-deadline, cannot interfere with J_j^* under EQDF although its scheduling window overlaps with that of J_j^*. However, the carry-out job may enter the zero-laxity state as shown in Fig. 4(a), and some parts of its execution (that are within the scheduling window of J_j^*) can interfere with J_j^* under EQDZL. By aligning J_j^*'s deadline (not quasi-deadline) and the carry-out job's deadline as shown in Fig. 4(b), we increase the total amount of interference. Then, $I_{j\leftarrow i}^{\mathrm{EQDZL}}(L,k)$ when $\phi\le 0$ (or, equivalently $k\cdot C_i\le k\cdot C_j$ by Eq. (5)) is calculated by $\Phi_i(L)\cdot C_i+\min(C_i,L-\Phi_i(L)\cdot T_i)$, which is calculated in Bertogna et al. (2005a) and Baker et al. (2008) for EDF and EDZL.

When $0<\phi\le(D_i-C_i)$ and $\phi>(D_i-C_i)$, the carry-out job has smaller quasi-deadline than J_j^*. The carry-out job already interferes under with J_j^* under EQDF. Therefore, the worst-case scenarios for EQDF can be also applied to EQDZL, implying $I_{j\leftarrow i}^{\mathrm{EQDZL}}(L,k)=I_{j\leftarrow i}^{\mathrm{EQDF}}(L,k)$.

To summarize, $I_{j\leftarrow i}^{\mathrm{EQDZL}}(L,k)$ can be calculated as follows.

$$I_{j\leftarrow i}^{\mathrm{EQDZL}}(L,k)=\begin{cases}\Phi_i(L)\cdot C_i+\min(C_i,L-\Phi_i(L)\cdot T_i), \\ \quad \text{if } k\cdot C_i\le k\cdot C_j, \text{and} \tau_i \text{ can enter the zero-laxity state,} \\ I_{j\leftarrow i}^{\mathrm{EQDF}}(L,k), \quad \text{otherwise.}\end{cases} \tag{14}$$

Incorporating Eq. (14) into Lemma 2, we now present a schedulability test of EQDZL in the following theorem.

Theorem 2. *A task set τ is schedulable under EQDZL scheduling with a system-wide variable k on a multiprocessor composed of m identical processors, if at most m tasks $\tau_j\in\tau$ violate the following conditions:*

$$\sum_{i\neq j}\min\left(I_{j\leftarrow i}^{\mathrm{EQDZL}}(D_j,k),D_j-C_j\right)<m\cdot(D_j-C_j). \tag{15}$$

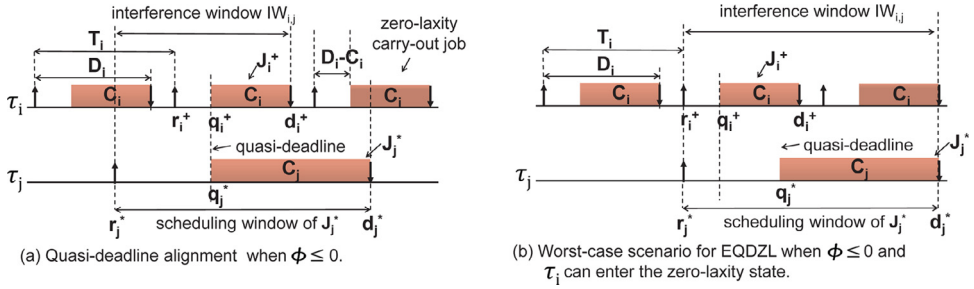


Fig. 4. EQDZL worst-case scenarios with $k = 1$.

Proof. The proof is similar to that of Theorem 1. By Lemma 2, τ is schedulable under EQDZL if Eq. (13) holds.

Similar to the proof of Theorem 1, we can prove that $\bar{I}_{j\leftarrow i}$ in Eq. (13) is upper-bounded by $I^{\text{EQDZL}}_{j\leftarrow i}(D_j, k)$ in Eq. (15).

Therefore, by Lemma 2, if Eq. (15) is violated for at most m tasks in a task set τ, the task set is schedulable under global EQDZL scheduling on m identical processors. □

We note that the zero-laxity state of a task τ_i may be unknown when the schedulability of another task τ_j is checked by Eq. (15). In order to deal with such ambiguity, we check the schedulability of each task in a certain order. According to the definition of $I^{\text{EQDZL}}_{j\leftarrow i}(L, k)$ in Eq. (14), if $k \cdot C_i > k \cdot C_j$, it does not have to identify the zero-laxity state of τ_i. So we check schedulability in an increasing order of $k \cdot C_j$.

For presentational convenience, without loss of generality, let us assume tasks are sorted in an increasing order of $k \cdot C_i$ for all tasks τ_i (i.e., $k \cdot C_i \le k \cdot C_{i+1}$). A task with the smallest value of $k \cdot C_1$ (i.e., τ_1) cannot satisfy $k \cdot C_i < k \cdot C_1$ for any task $\tau_i \in \tau$. That is, $I^{\text{EQDZL}}_{1\leftarrow i}(L, k) = I^{\text{EQDF}}_{1\leftarrow i}(L, k)$ holds regardless of other tasks' capability of entering the zero-laxity state. Therefore, we can test Eq. (15) for τ_1, and then we know whether τ_1 can enter the zero-laxity state (if Eq. (15) does not hold) or not (if Eq. (15) holds). Next, we test Eq. (15) for a task with the second smallest value of $k \cdot C_2$ (i.e., τ_2). In this case, τ_1 is the only task that may satisfy $k \cdot C_1 < k \cdot C_2$, and we already identified τ_1's capability of entering the zero-laxity state. Therefore, we can compute Eq. (15) for τ_2. This procedure is sequentially applied to tasks with the third, fourth, ..., nth smallest value of $k \cdot C_n$, and finally, we finish testing Eq. (15).

We note that the above EQDZL schedulability test in Theorem 2 is a generalization of the existing EDZL schedulability test (Baker et al., 2008). In other words, the condition is equivalent to the EDZL schedulability condition when $k = 0$.

The following example shows that the proposed EQDZL schedulability test can be used to guarantee the schedulability of τ, even when the system density is greater than the number of processors (i.e., $\delta_{sys} > m$). We note that a class of optimal algorithms (Anderson and Srinivasan, 2000; Baruah et al., 1996; Cho et al., 2006; Funaoka et al., 2008; Levin et al., 2010; Regnier et al., 2011) for implicit-deadline task systems is not able to guarantee the schedulability of τ in the following example, since $\delta_{sys} > m$.

Example 4.2. Let $\tau = \{\tau_1 = (4, 1, 4), \tau_2 = (4, 1, 2), \tau_3 = (5, 1, 1), \tau_4 = (7, 4, 7)\}$ and $m = 2$. The system density of τ is 2.32 which exceeds the number of processors ($m = 2$). τ is not deemed schedulable under EDZL by Theorem 2 when $k = 0$, but it is deemed schedulable under EQDZL with $k = 1$ according to Theorem 2. In the EQDZL schedulability test with $k = 1$, τ_2 and τ_3 can enter the zero-laxity state, but, in the EDZL case (when $k = 0$), more than m tasks (i.e., τ_2, τ_3, and τ_4) violate Eq. (15).

4.3. Slack-based iterative test

In general, bounding interference involves much pessimism, particularly, in computing the workload of a carry-in job. Hence, the slack-based iterative approaches (Bertogna and Cirinei, 2007; Bertogna et al., 2009) are introduced to reduce such pessimism effectively. Let S_j denote the slack of a task τ_j and is defined as

$$S_j \triangleq D_j - C_j - \left\lfloor \frac{\sum_{i\neq j} \min(\bar{I}_{j\leftarrow i}, D_j - C_j + 1)}{m} \right\rfloor, \tag{16}$$

when (16) is positive. A lower bound S^{lb}_j on the slack S_j of a task τ_j under EQDF is then given by

$$S^{lb}_j \triangleq D_j - C_j - \left\lfloor \frac{\sum_{i\neq j} \min(I^{\text{EQDF}}_{j\leftarrow i}(D_j, k), D_j - C_j + 1)}{m} \right\rfloor, \tag{17}$$

when this term is positive. Note that if the scheduler is EQDZL, $I^{\text{EQDF}}_{j\leftarrow i}(D_j, k)$ in Eq. (17) is replaced by $I^{\text{EQDZL}}_{j\leftarrow i}(D_j, k)$.

Fortunately, the iterative test exploiting a slack value can be directly adopted into our EQDF and EQDZL schedulability tests. In this paper, the iterative test can be easily incorporated by replacing the bound of the carry-in job's contribution (Eq. (8)) with $\min(C_i, L - S^{lb}_i - \Phi_i(L) \cdot T_i)$.

5. Quasi-deadline assignment for EQDF and EQDZL

In this section, we consider the problem of priority assignment under global EQDF (EQDZL) scheduling. Specifically, we examine the *k-controlled quasi-deadline assignment* problem; given a task set, this problem finds a value of the quasi-deadline control knob k such that each individual job J^h_i is assigned a quasi-deadline q^h_i equal to $d^h_i - k \cdot C_i$ and the task set is deemed schedulable under global EQDF (EQDZL) scheduling by the schedulability test in Theorem 1 (Theorem 2). For presentational convenience, a value of k is referred to as *schedulable* for a given task set τ if τ is deemed schedulable with this k value according to Theorem 1 for EQDF (Theorem 2 for EQDZL). A solution algorithm to the k-controlled quasi-deadline assignment is referred to as *optimal*, if the solution algorithm can find any schedulable value of k for a task set τ if and only if there exists some schedulable value of k for τ.

Section 5.1 presents an optimal solution of EQDF to the problem, and Section 5.2 explains how to derive the optimal solution of EQDZL based on that of EQDF. Then, Section 5.3 discusses heuristic solutions.

5.1. Optimal quasi-deadline assignment under EQDF

In this sub-section, we present an algorithm, called OQDA-k (optimal quasi-deadline assignment by k) that finds a set of all the schedulable values of k for a given task set. A brute-force approach would examine all possible values of k for schedulability. This approach is prohibitively expensive, and it is not even applicable to the case of continuous values of k. Instead, we seek to identify a finite

set of k values that guarantees the discovery of all the schedulable values of k.

The OQDA-k algorithm first carries out efficient discovery of a finite number of certain k values (denoted by $A_{j\leftarrow i}$), taking advantage of interference patterns between every two tasks τ_i and τ_j (step A1). It then aggregates those k values into a single set (denoted by A_j) per each task τ_j (step A2) and constructs a set of intervals (denoted by S_j) from A_j for each task τ_j such that each interval represents a set of continuous schedulable values of k for an individual τ_j (step A3). Finally, the algorithm generates a set of intervals (denoted by S) such that each interval contains the continuous schedulable values of k for an entire task set (step A4). Algorithm 1 summarizes the OQDA-k algorithm, and we describe each step in more details as follows.

Algorithm 1 OQDA-k (τ).

1: **for** each task τ_j **do**
2: **for** each task τ_i **do**
3: if $\tau_j \neq \tau_i$, construct $A_{j\leftarrow i}$
4: **end for**
5: $A_j \leftarrow \bigcup_{\forall \tau_i} A_{j\leftarrow i}$
6: construct S_j from A_j
7: **end for**
8: $S \leftarrow \bigcap_{\forall \tau_j} S_j$
9: if S is empty, **return** unschedulable
10: **return** S

A1. In the first step A1, the algorithm generates a number of discrete k values for all possible pairs of tasks, exploiting the relationship between k and the interference between two tasks (i.e., line 3 in Algorithm 1). The interference of a task τ_i with a job of task τ_j varies with k (i.e., $I^{\mathrm{EQDF}}_{j\leftarrow i}(D_j, k)$ in Eq. (11)), and k makes a different impact on the interference depending on the relationship between τ_i and τ_j.

It is easy to see that each job is assigned an earlier quasi-deadline as k increases. In particular, when a task τ_i has a greater execution time requirement than another τ_j has (i.e., $C_i > C_j$), an increase on k results in a larger reduction to the quasi-deadline of τ_i than that of τ_j. In this case, τ_i is then likely to have a higher priority and thus impose a greater amount of interference on τ_j. This leads to $I^{\mathrm{EQDF}}_{j\leftarrow i}(D_j, k)$ monotonically non-decreasing with k when $C_i > C_j$. Likewise, $I^{\mathrm{EQDF}}_{j\leftarrow i}(D_j, k)$ is monotonically non-increasing as k increases if $C_i < C_j$. When $C_i = C_j$, on the other hand, $I^{\mathrm{EQDF}}_{j\leftarrow i}(D_j, k)$ remains constant because the quasi-deadlines of τ_i and τ_j and their priorities remain relatively the same even though k varies.

Fig. 5 illustrates the impact of k on $I^{\mathrm{EQDF}}_{j\leftarrow i}(D_j, k)$. A value of k is said to be a *turning point* of $I^{\mathrm{EQDF}}_{j\leftarrow i}(D_j, k)$ if $I^{\mathrm{EQDF}}_{j\leftarrow i}(D_j, k)$ changes its slope at this

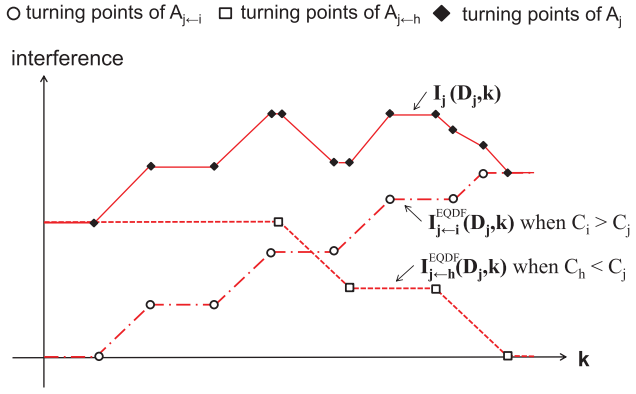


Fig. 5. Example graph representing interference made by higher priority tasks according to value k under EQDF.

k value. The formula $I^{\mathrm{EQDF}}_{j\leftarrow i}(D_j, k)$ defined in Eq. (11) is a combination of linear and constant shape functions as shown in Fig. 5. By definition, $I^{\mathrm{EQDF}}_{j\leftarrow i}(D_j, k)$ is lower-bounded by zero, and it is also upper-bounded by the amount of the interference in the worst-case scenario II. This is because the condition of $k \cdot C_i - k \cdot C_j \leq (D_i - C_i)$ in Eq. (11) makes the interference in the worst-case scenario I always lower than or equal to that in the worst-case scenario II.

When $I^{\mathrm{EQDF}}_{j\leftarrow i}(D_j, k)$ is a monotonically increasing function, it starts from the lower-bound point increasing linearly as k increases. It stays constant while the contribution of the carry-in job is bounded by C_i and the number of body and carry-out jobs does not change as k increases. It resumes increasing linearly again when a new body job comes in. The process of increasing linearly and staying constant repeats until it reaches the upper-bound point. Therefore, $I^{\mathrm{EQDF}}_{j\leftarrow i}(D_j, k)$ has a finite number of turning points.

We can easily calculate all turning points with the understanding of the dynamics of $I^{\mathrm{EQDF}}_{j\leftarrow i}(D_j, k)$. According to Eq. (11), the value of $I^{\mathrm{EQDF}}_{j\leftarrow i}(D_j, k)$ is dependent on the length of interference window $|IW_{i,j}|$. The lower-bound point of $I^{\mathrm{EQDF}}_{j\leftarrow i}(D_j, k)$ is placed when $|IW_{i,j}|$ is zero, and the value of such k can be calculated as $\frac{D_j}{C_j - C_i}$ according to Eq. (6). The upper-bound point of $I^{\mathrm{EQDF}}_{j\leftarrow i}(D_j, k)$ is placed when $|IW_{i,j}|$ is $D_j - C_i + D_i$ (i.e., the worst-case scenario II), and the value of such k can be computed as $\frac{D_i - C_i}{C_i - C_j}$. Between those two points, a turning point of $I^{\mathrm{EQDF}}_{j\leftarrow i}(D_j, k)$ is placed at every $|IW_{i,j}|$ of $x \cdot T_i + C_i$ or $x \cdot T_i$ where x is non-negative integer until it reaches the upper-bound point.

Let $A_{j\leftarrow i}$ denote a set of all turning points of $I^{\mathrm{EQDF}}_{j\leftarrow i}(D_j, k)$. In this step, the OQDA-k algorithm constructs $A_{j\leftarrow i}$ for all tasks τ_i and τ_j.

A2. The previous step looked at the relationship between k and the interference of a single task τ_i on τ_j, and this step explores the relationship between k and a total interference imposed on τ_j by an entire task set. Let us define $I_j(D_j, k)$ as the sum of individual interferences ($= \sum_{i\neq j} I^{\mathrm{EQDF}}_{j\leftarrow i}(D_j, k)$). It is also a combination of linear and constant shape functions. Then, $I_j(D_j, k)$ also has a sequence of turning points. Let A_j denote a set of whole turning points of $I_j(D_j, k)$. If a value of k is a turning point of $I^{\mathrm{EQDF}}_{j\leftarrow i}(D_j, k)$, then it is also a turning point of $I_j(D_j, k)$.[4] So we can construct A_j as the union of all the turning points of $I^{\mathrm{EQDF}}_{j\leftarrow i}(D_j, k)$ for all tasks $i \neq j$ (i.e., line 5 in Algorithm 1). Even though $I_j(D_j, k)$ is a combination of linear and constant functions, unlike $I^{\mathrm{EQDF}}_{j\leftarrow i}(D_j, k)$, $I_j(D_j, k)$ does not increase (or decrease) monotonically over all k values. However, $I_j(D_j, k)$ remains constant or increases (or decreases) linearly within an interval between two consecutive turning points.

A3: How to find schedulable k. Recall that a value of k is schedulable for a task τ_j if τ_j is deemed schedulable with this k value according to Eq. (12). The third step finds out all the schedulable values of k (denoted by S_j) out of a set of turning points (A_j) for each task τ_j (i.e., line 6 in Algorithm 1). Consider two consecutive turning points of τ_j ($p^h_j \in A_j$ and $p^{h+1}_j \in A_j$). That is, there does not exist $p' \in A_j$ such that $p^h_j < p' < p^{h+1}_j$. We consider four cases in constructing S_j with p^h_j and p^{h+1}_j according to their schedulability.

Firstly, if both p^h_j and p^{h+1}_j are schedulable for τ_j, then all the values of k within the interval $[p^h_j, p^{h+1}_j]$ are also schedulable. This is because $I_j(D_j, k)$ remains constant or moves linearly within the two

[4] There exists one exceptional case of the statement, where two different interfering tasks (τ_i and $\tau_{i'}$) have the same slopes but different signs in $I^{\mathrm{EQDF}}_{j\leftarrow i}(D_j, k)$ and $I^{\mathrm{EQDF}}_{j\leftarrow i'}(D_j, k)$ (i.e., one with a positive slope and one with a negative slope). In this case, each task's contribution to $I_j(D_j, k)$ between two consecutive turning points cancels out each other, which leads to excluding those turning points from A_j. However, this case rarely occurs, and including those points to A_j does not hurt the correctness of the algorithm. Thereby, we do not handle this exceptional case.

consecutive turning points. So, the interval $[p_j^h, p_j^{h+1}]$ is added into S_j. Secondly, if neither p_j^h nor p_j^{h+1} is schedulable, there is no schedulable value of k within the interval $[p_j^h, p_j^{h+1}]$ and nothing is added into S_j. Thirdly, if p_j^h is schedulable but p_j^{h+1} is not, there must exist $k' \in [p_j^h, p_j^{h+1}]$ such that $[p_j^h, k']$ is the interval of schedulable values of k but $(k', p_j^{h+1}]$ is not. We note that the value of k' can be easily identified since $I_j(D_j, k)$ has a steady slope over $[p_j^h, p_j^{h+1}]$. Then, $[p_j^h, k']$ is inserted into S_j. In the fourth case where p_j^h is not schedulable but p_j^{h+1} is, another interval is added into S_j in a similar way to the third case.

A4. The fourth step finally constructs a set of all the schedulable values of k (denoted by S) for a given task set τ. The set S is indeed the intersection of all S_j; each element $s \in S$ satisfies $s \in S_j$ for all tasks $\tau_j \in \tau$ (i.e., line 8 in Algorithm 1).

Then, we record the optimality of OQDA-k as follows.

Theorem 3. *The OQDA-k algorithm in Algorithm 1 is an optimal quasi-deadline assignment policy with respect to the EQDF schedulability test in Theorem 1.*

Proof. We show this theorem by contradiction. Suppose the set S computed by the OQDA-k algorithm is empty even though there exists a schedulable value of k (denoted as k^*). We consider two cases depending on whether k^* is a turning point.

Suppose k^* is a turning point of the interference function $I_{j\leftarrow i}^{\mathrm{EQDF}}(D_j, k)$ for tasks τ_i and τ_j. By definition of a schedulable value of k, a given task set τ is deemed schedulable with this k^* value by Theorem 1. According to the OQDA-k algorithm, k^* is then placed into $A_{j\leftarrow i}$ (i.e., line 3 in Algorithm 1) and subsequently included in A_j, S_j, and S (i.e., lines 5, 6, and 8 in Algorithm 1). This contradicts the assumption that S is empty.

Consider the other case where k^* is not a turning point. Then, there exist two consecutive turning points of A_j (denoted as p_j^h and p_j^{h+1}) such that $p_j^h < k^* < p_j^{h+1}$. From the assumption that S is empty, neither p_i^h nor p_j^{h+1} is schedulable by Theorem 1. Otherwise, any schedulable turning point should be added into S_j and thereby into S, contradicting the assumption of the empty S. So, p_i^h and p_j^{h+1} must be not schedulable. Then, k^* should not be schedulable either, since $I_{j\leftarrow i}^{\mathrm{EQDF}}(D_j, k)$ is constant or linear within the interval $[p_i^h, p_j^{h+1}]$. This contradicts the assumption that k is schedulable. This concludes the proof of Theorem 3. □

Time-complexity. Recall that we denote the number of tasks in a task set by n. In step A1, identifying $A_{j\leftarrow i}$ for given τ_i and τ_j requires $O(\lceil \frac{D_j+D_i-C_i}{T_i} \rceil)$ operations since the number of turning points in $A_{j\leftarrow i}$ is $\lceil \frac{D_j+D_i-C_i}{T_i} \rceil \times 2$. As discussed in A1, each turning point is placed from $|IW_{i,j}| = 0$ to $|IW_{i,j}| = D_j + D_i - C_i$ at every $x \cdot T_i + C_i$ or $x \cdot T_i$ where x is non-negative integer. In step A2, $|A_j|$ is calculated as $\sum_{i\neq j} \lceil \frac{D_j+D_i-C_i}{T_i} \rceil \times 2$, and it is bounded by $(n-1) \cdot \lceil \frac{2D_{\max}-C_{\min}}{T_{\min}} \rceil \times 2$ where $D_{\max}$, $C_{\min}$, $T_{\min}$ represent the largest D_i, the smallest C_i and the smallest T_i among all tasks $\tau_i \in \tau$, respectively. Then, calculating A_j for all tasks requires $O(n^2 \cdot \lceil \frac{2D_{\max}-C_{\min}}{T_{\min}} \rceil)$ operations for step A2. For step A3, Eq. (12) for each task τ_j is tested for all points in A_j. Since testing Eq. (12) for each task τ_j itself requires $O(n)$, step A3 requires $O(n^3 \cdot \lceil \frac{2D_{\max}-C_{\min}}{T_{\min}} \rceil)$. Lastly, step A4 simply requires $O(\sum_{\tau_j\in\tau} |S_j|)$. Since $|S_j| \le |A_j|$ and $|A_j| \ge 1$ hold for every $\tau_j \in \tau$, the running time of the OQDA-k algorithm is thereby $O(n^3 \cdot \lceil \frac{2D_{\max}-C_{\min}}{T_{\min}} \rceil)$.

5.2. Optimal quasi-deadline assignment under EQDZL

In Section 5.1, OQDA-k for EQDF efficiently finds all schedulable values of k. Similarly, we now develop the OQDA$^{\mathrm{ZL}}$-k algorithm for EQDZL, which finds all schedulable values of k such that Theorem 2 holds. The overall structure of OQDA$^{\mathrm{ZL}}$-k is similar to that of OQDA-k, and therefore we focus on explaining their major differences. The first difference is caused by the interference upper-bound functions, i.e., $I_{j\leftarrow i}^{\mathrm{EQDZL}}(D_j, k)$ and $I_{j\leftarrow i}^{\mathrm{EQDF}}(D_j, k)$. The second one comes from the structure of the two theorems; instead of finding a set of all schedulable values of k which prevent all tasks from causing a deadline miss under EQDF (from Theorem 1), we find a set of all schedulable values of k such that at most m tasks can enter the zero-laxity state (from Theorem 2).

As shown in Eq. (14), $I_{j\leftarrow i}^{\mathrm{EQDZL}}(D_j, k)$ is different from $I_{j\leftarrow i}^{\mathrm{EQDF}}(D_j, k)$ only when $k \cdot C_i \le k \cdot C_j$ holds and τ_i is able to enter the zero-laxity state. Since it varies with k's sign $(+/-)$ that determines whether $k \cdot C_i \le k \cdot C_j$ holds or not, we consider two intervals: $k \in (-\infty, 0)$ and $k \in (0, \infty)$. It is trivial that we need not care for $k = 0$ since $I_{j\leftarrow i}^{\mathrm{EQDZL}}(L, 0)$ is the same as $I_{j\leftarrow i}^{\mathrm{EDZL}}(L, 0)$ for any pair of τ_i and τ_j.

Algorithm 2 OQDA$^{\mathrm{ZL}}$-k (τ).

1: **for** each task τ_j, largest C_j first **do**
2: **for** each task τ_i **do**
3: if $\tau_j \neq \tau_i$, construct $A_{j\leftarrow i}^N$
4: **end for**
5: $A_j^N \leftarrow \bigcup_{\forall \tau_i} A_{j\leftarrow i}^N$
6: construct S_j^N from A_j^N
7: **end for**
8: **for** each task τ_j, smallest C_j first **do**
9: **for** each task τ_i **do**
10: if $\tau_j \neq \tau_i$, construct $A_{j\leftarrow i}^P$
11: **end for**
12: $A_j^P \leftarrow \bigcup_{\forall \tau_i} A_{j\leftarrow i}^P$
13: construct S_j^P from A_j^P
14: **end for**
15: $S_j \leftarrow S_j^N \cup S_j^P$
16: construct S from $\{S_j\}_{\forall \tau_j}$
17: if S is empty, **return** unschedulable
18: **return** S

In case of $k \in (-\infty, 0)$, $k \cdot C_i \le k \cdot C_j$ holds only when $C_i \ge C_j$ holds. Then, as shown in Fig. 6, there may exist a turning point $p_{j\leftarrow i}^{\mathrm{ZL}}$ which is derived by the difference between $I_{j\leftarrow i}^{\mathrm{EQDZL}}(D_j, k)$ and $I_{j\leftarrow i}^{\mathrm{EQDF}}(D_j, k)$, and this turning point occurs only when τ_i can enter the zero-laxity state. To identify each task's capability of being zero-laxity, Section 4.2 already showed that we need a particular order: the smallest value of $k \cdot C_j$, the earlier test. This can be also applied to OQDA$^{\mathrm{ZL}}$-k. Since a task

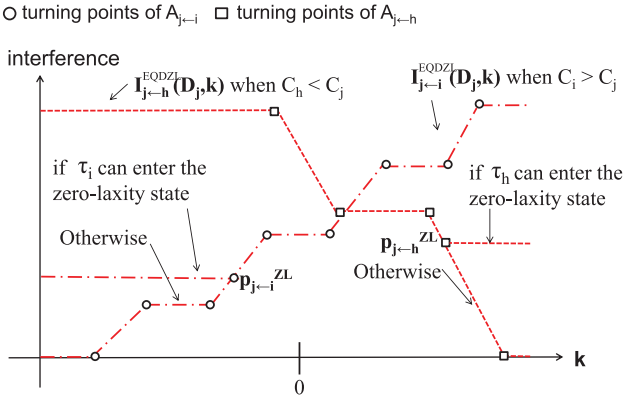


Fig. 6. Example graph representing interference made by higher priority tasks according to value k under EQDZL.

with the smallest value of $k \cdot C_1$ (i.e., τ_1) cannot satisfy $k \cdot C_i \le k \cdot C_1$ for any other task τ_i, the steps A1, A2 and A3 for τ_1 can be successfully performed regardless of other tasks' capability of entering the zero-laxity state. Then, we obtain S_1, a set of all k [values] which satisfy Eq. (15) for τ_1 (i.e., a set of all k such that τ_1 cannot enter the zero-laxity state under EQDZL). Next, for a task with the second smallest value of $k \cdot C_2$ (i.e., τ_2), τ_1 is the only task τ_i ($\neq \tau_j$) that may satisfy $k \cdot C_i \le k \cdot C_2$, and we knew a set of all k such that τ_1 can enter the zero-laxity state (i.e., S_1^C: the absolute complement set of S_1). Therefore, we can perform the steps A1, A2 and A3 for τ_2, obtaining S_2. We can calculate all the other S_j by applying this procedure to tasks with the third, fourth, ..., nth largest value of $k \cdot C_j$, in a sequential manner (i.e., lines 1–7 in Algorithm 2). Then, we obtain $A^N_{j\leftarrow i}$, A^N_j and S^N_j, which correspond to negative elements of $A_{j\leftarrow i}$, A_j and S_j respectively.

For $k \in (0, \infty)$, $k \cdot C_i \le k \cdot C_j$ holds only when $C_i \le C_j$. Therefore, the order of executing individual tasks' steps A1, A2 and A3 is the opposite; the same procedure is sequentially applied to tasks with the first, second, ..., and nth smallest value of $k \cdot C_j$ (i.e., lines 8–14 in Algorithm 2). Then, we obtain $A^P_{j\leftarrow i}$, A^P_j and S^P_j, which correspond to positive elements of $A_{j\leftarrow i}$, A_j and S_j respectively.

Finally, we successfully finish calculating S_j for all $\tau_j \in \tau$ by performing steps A1, A2 and A3 of individual tasks in a particular order. Then, using S_j for all $\tau_j \in \tau$, we can perform the final step A4 as follows: a set of all k which makes the task set schedulable (denoted by S) is a set of all elements s such that s belongs to S_j for at least $n - m$ tasks $\tau_j \in \tau$ (i.e., lines 15–16 in Algorithm 2).

Similar to OQDA-k, we now present the optimality of OQDA^{ZL}-k in the following theorem.

Theorem 4. *The OQDA^{ZL}-k algorithm in Algorithm 2 is an optimal quasi-deadline assignment policy with respect to the EQDZL schedulability test in Theorem 2.*

Proof. The proof is similar to that of Theorem 3. We show this theorem by contradiction. Suppose the set S computed by the OQDA^{ZL}-k algorithm is empty even though there exists a schedulable value of k (denoted as k^*) according to Theorem 2. We consider two cases depending on whether k^* is a turning point.

Suppose k^* is a turning point of the interference function $I^{EQDZL}_{j\leftarrow i}(D_j, k)$ for tasks τ_i and τ_j. By definition of a schedulable value of k, a given task set τ is deemed schedulable with this k^* value by Theorem 2. According to the OQDA^{ZL}-k algorithm, k^* is then placed into $A^N_{j\leftarrow i}$ or $A^P_{j\leftarrow i}$ for at least $n - m$ tasks τ_j (i.e., line 3 or 10 in Algorithm 2) and subsequently included in A^N_j (or A^P_j), S_j, and S (respectively by lines 5 and 6 (or 12 and 13), 15, and 16 in Algorithm 2). This contradicts the assumption that S is empty.

Similar to the proof of Theorem 3, we can derive a contradiction for the other case where k^* is not a turning point. This concludes the proof of Theorem 4. □

Time-complexity. Compared to OQDA-k, OQDA^{ZL}-k additionally needs to sort tasks by their execution time, which requires $O(n \log n)$ computations. Therefore, the total time complexity of OQDA^{ZL}-k is the same as OQDA-k: $O(n^3 \cdot \lceil \frac{2D_{max} - C_{min}}{T_{min}} \rceil)$.

Algorithm 3 HQDA-k (τ, K_1, K_2, K_s).

```
1: S ← ∅
2: for k = K1 to K2 step Ks do
3:     if τ is schedulable with k according to Theorem 1 then
4:         S ← S ∪ {k}
5:         return S
6:     end if
7: end for
8: return unschedulable
```

5.3. Heuristic priority assignment

The OQDA-k and OQDA^{ZL}-k algorithms identify and explore all the turning points of an entire task set to find out all optimal values of k. As the number of tasks increases, the optimal algorithms have a fast growing number of turning points to explore and thereby their running time also increases rapidly.

Therefore, we introduce a heuristic algorithm (HQDA-k) for EQDF that finds a schedulable value of k in a sub-optimal but efficient way. Algorithm 3 summarizes this heuristic algorithm. This algorithm is given a set of k values and repeats the process of examining whether there exists a schedulable value of k in the given set until it finds any solution or there is no more element in the set to examine. The set is given as an interval $[K_1, K_2]$ with a step size K_s, and the algorithm checks with $k = K_1, K_1 + K_s, K_1 + 2K_s, \ldots, K_2$. To increase the performance of the heuristic algorithm, the interval and the step size are determined by analyzing the property of our interference-based analysis in the next section. The evaluation of this heuristic algorithm is also provided in the next section.

HQDA^{ZL}-k, a heuristic algorithm for EQDZL, is the same as HQDA-k, except that Theorem 2 is used instead of Theorem 1 in line 3 of Algorithm 3.

6. Evaluation

This section presents simulation results to evaluate the proposed EQDF and EQDZL schedulability tests and quasi-deadline assignment algorithms.

Simulation environment. Task sets are generated based on a technique proposed earlier (Baker, 2005), which has also been used in many previous studies (e.g., see Andersson et al., 2008; Bertogna et al., 2009; Lee et al., 2011a). We have two input parameters. One is the number of processors ($m = 4$ or 8), and the other is a task utilization parameter. For each task τ_i, T_i is uniformly chosen in [100, 1000], and C_i is chosen based on a bimodal or exponential task utilization parameter.[5] Although our proposed EQDF and EQDZL analyses are applicable to both implicit and constrained deadline models, because of page limit, we only show the results of the constrained deadline model: D_i is set less than or equal to T_i. For each task utilization model, we repeat the following procedure to generate 1000 task sets.

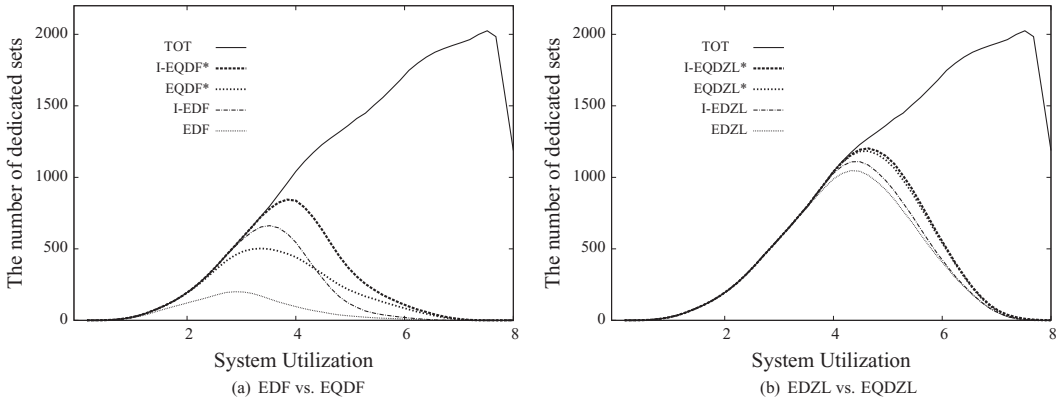
1. Initially, we generate a set of $m + 1$ tasks.
2. We check whether the generated task set can pass a necessary feasibility condition (Baker and Cirinei, 2006; Baruah et al., 2009).
3. If it fails to pass the feasibility test, we discard the generated task set and return to Step 1. Otherwise, we include this set for evaluation. Then, this set is used as a basis for the next task set; we create a new set by adding a new task into the old set and return to Step 2.

For any given m, we create 1000 task sets for an individual task utilization model, resulting in 10,000 task sets in total.

6.1. Deadline vs. quasi-deadline

Our first simulations were performed to evaluate the maximum benefit that can be achieved by the use of quasi-deadlines in deadline-based scheduling. To this end, we compare the schedulability performances of EQDF and EQDZL with that of EDF and EDZL respectively, in terms of how many task sets are deemed schedulable by their

[5] For a given bimodal parameter p, a value for C_i/T_i is uniformly chosen in $[0, 0.5)$ with probability p, where $p = 0.1, 0.3, 0.5, 0.7$, or 0.9. For a given exponential parameter $1/\lambda$, a value for C_i/T_i is chosen according to the exponential distribution whose probability density function is $0.5 \cdot \lambda \cdot \exp(-\lambda \cdot x)$, where $\lambda = 0.1, 0.3, 0.5, 0.7$ or 0.9.



(a) EDF vs. EQDF (b) EDZL vs. EQDZL

Fig. 7. Schedulability of algorithms when $m = 8$.

own schedulability tests. In order to find the maximum schedulability performance of EQDF and EQDZL, the quasi-deadline control knob k is set to an optimal value for each task set through the OQDA-k and OQDA^{ZL}-k algorithms, respectively. The following schedulability tests were used in simulation:

- the EDF test in Bertogna et al. (2009) (EDF);
- the iterative EDF test in Bertogna et al. (2009) (I-EDF);
- our EQDF test in Theorem 1 with an optimal value k^* (EQDF*);
- our iterative EQDF test in Eq. (17) with an optimal value k^* (I-EQDF*);
- the EDZL test in Baker et al. (2008) (EDZL);
- the iterative EDZL test in Baker et al. (2008) (I-EDZL);
- our EQDZL test in Theorem 4 with an optimal value k^* (EQDZL*);
- our iterative EQDZL test in Eq. (17) with $I_{j\leftarrow i}^{\mathrm{EQDF}}(D_j, k)$ replaced with $I_{j\leftarrow i}^{\mathrm{EQDZL}}(D_j, k)$ for an optimal value k^* (I-EQDZL*).

Fig. 7 compares EDF vs. EQDF and EDZL vs. EQDZL on the basis of schedulability with $m = 8$. Each line represents the number of task sets deemed schedulable by one specific test, except for the curve labeled with TOT which represents an upper bound on the feasible task sets. Fig. 7 shows the use of quasi-deadlines can significantly improve the schedulability of deadline-based scheduling. It is shown in Fig. 7(a) that I-EQDF* dominates all the other tests, substantially outperforming the EDF schedulability tests particularly when the system utilization is between $m/3$ and $2m/3$. Similarly, Fig. 7(b) shows that I-EQDZL* also outperforms the other tests.

Table 1 also shows that EQDF* finds 160–300% more task sets schedulable than EDF does, and I-EQDF* detects 40–45% more task sets schedulable than I-EDF does on 4 and 8 processors, respectively. Table 2 also shows that EQDZL* and I-EQDZL* outperform the corresponding EDZL schedulability tests, finding at least 10% more task sets schedulable. We note that unlike EDF, EDZL already has a good performance in multiprocessor scheduling since it is able to consider both urgency and parallelism via zero laxity. The figure indicates that schedulability can be even improved when quasi-deadlines are considered together with zero laxity. Unlike significant improvement by I-EQDF* over EQDF*, the performance improvement of I-EQDZL* is not significant compared to EQDZL*. However, this behavior is widely observed by zero-laxity based algorithms (Lee et al., 2012).

Table 1
EDF and EQDF tests.

	m	EDF	I-EDF	EQDF*	I-EQDF*
Schedulability (%)	4	11.3	25.4	28.9	35.8
	8	6.1	18.3	18.1	26.4
Running Time (μs)	4	0.3	2.1	5.54×10^2	1.42×10^4
	8	0.8	4.7	2.89×10^3	1.85×10^5

Table 2
EDZL and EQDZL tests.

	m	EDZL	I-EDZL	EQDZL*	I-EQDZL*
Schedulability (%)	4	45.1	46.7	51.3	51.7
	8	39.4	40.9	44.7	45.1
Running Time (μs)	4	0.5	3.1	1.84×10^3	2.01×10^5
	8	1.1	7.9	2.09×10^4	2.41×10^6

Our simulation results indicate that the schedulability of deadline-based scheduling can improve significantly when the quasi-deadlines of individual jobs are well assigned. That is, it is important to determine a good value of k for the effectiveness of EQDF and EQDZL scheduling. The OQDA-k and OQDA^{ZL}-k algorithms are able to find an optimal value of k. However, as shown in Tables 1 and 2, they are computationally expensive. Their running times are four to six orders of magnitude greater compared to the EDF and EDZL cases, leaving the OQDA-k and OQDA^{ZL}-k algorithms inappropriate for online priority assignment. This complexity calls for good, cost-effective, and alternative solutions to online quasi-deadline assignment.

6.2. Understanding the effect of quasi-deadline assignment

We seek to understand the impact of k on schedulability in order to gain good insights toward online quasi-deadline assignment.

To simplify the presentation, we define $N(a, b)$ as the number of task sets in which there exists a schedulable value $k^* \in [a, b]$. We also define $F(a, b)$ as the ratio of the number of task sets that have a schedulable value $k^* \in [a, b]$ to the number of schedulable task sets with any schedulable value $k^* \in [-\infty, \infty]$. That is, $F(a, b)$ is defined as the ratio of $N(a, b)$ to $N(-\infty, \infty)$.

Figs. 8 and 9 respectively show how schedulable values k^* of I-EQDF* and I-EQDZL* are distributed over k on $m = 4$ and $m = 8$ processors. More specifically, Figs. 8(a) and 9(a) plot $F(-\infty, k')$, representing the cumulative distribution of schedulable k values in a direction from $-\infty$ to the value of k'. On the other hand, Figs. 8(b) and 9(b) plot $F(k', \infty)$ in the other direction from k' to ∞. As shown in Fig. 8, there is a sharp increase in a short range of k' values before and after one, reaching at the 95th percentile or higher. Fig. 9 also shows the same trend before and after zero. This implies that we can find a schedulable value k^* with a high probability by looking at a small interval of k values, instead of exploring the whole range of $[-\infty, \infty]$, if any schedulable value k^* exists.

This motivates investigation into how likely a schedulable value of k belongs to an interval of length L. For a given length L, we find a real value t^* such that $N(t^*, t^* + L)$ is maximized. We define EQDF(L) and EQDZL(L) as I-EQDF* and I-EQDZL* tests with values of $k^* \in (t^*, t^* + L)$ respectively.

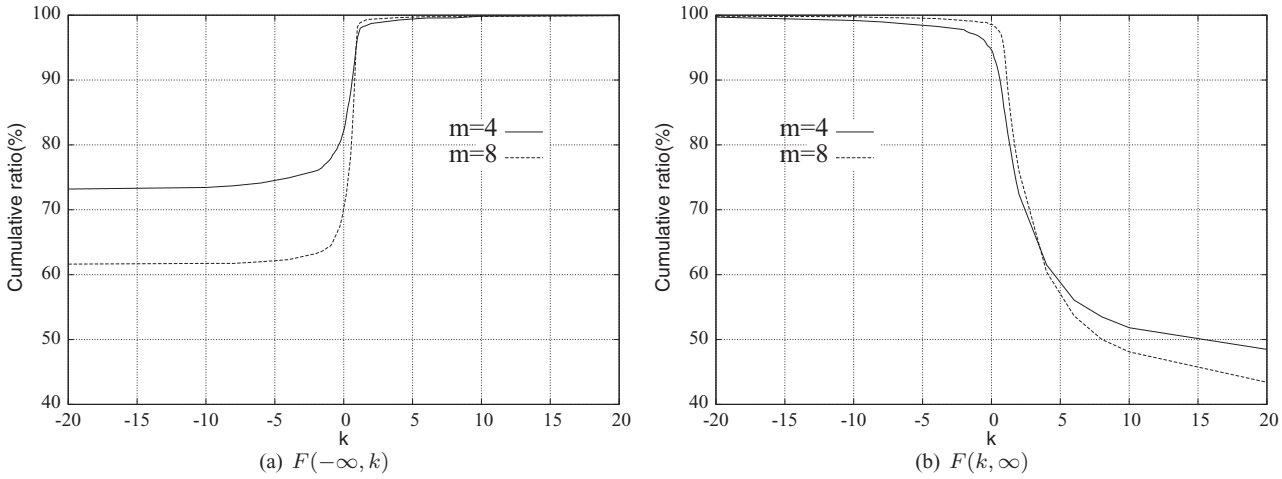


Fig. 8. Cumulative distribution of schedulable k values of EQDF.

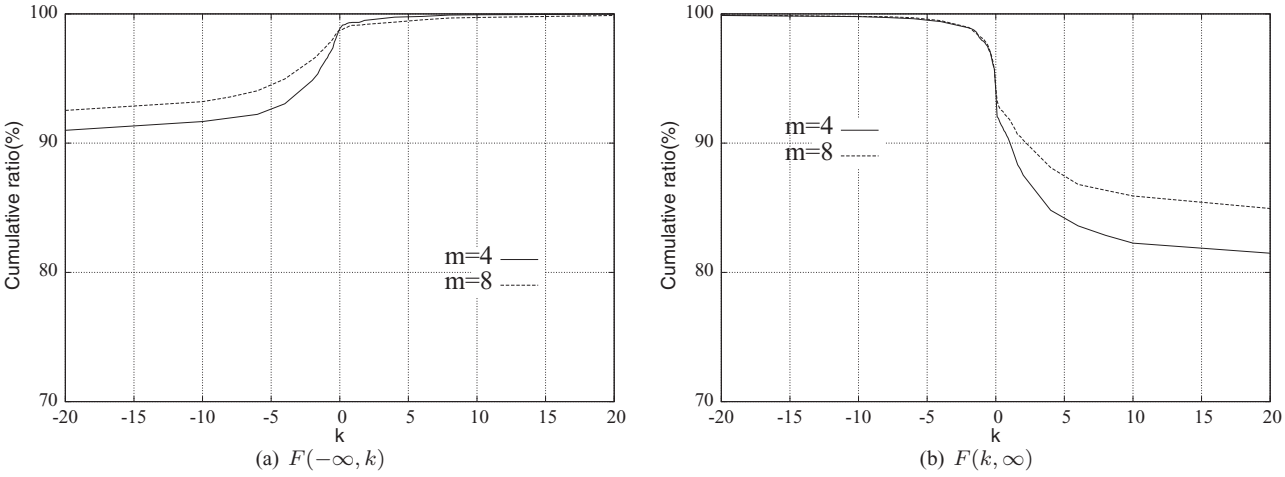


Fig. 9. Cumulative distribution of schedulable k values of EQDZL.

Table 3
Intervals ($[t^*, t^* + L]$).

m	Interval length (L) (EQDF)				
	1	2	4	8	16
4	[0.2,1.2]	[−0.7,1.3]	[−2.0,2.0]	[−5.4,2.6]	[−10.1,5.9]
8	[0,1.0]	[−0.7,1.3]	[−2.6,1.4]	[−5.2,2.8]	[−11.2,4.8]

Table 4
Intervals ($[t^*, t^* + L]$).

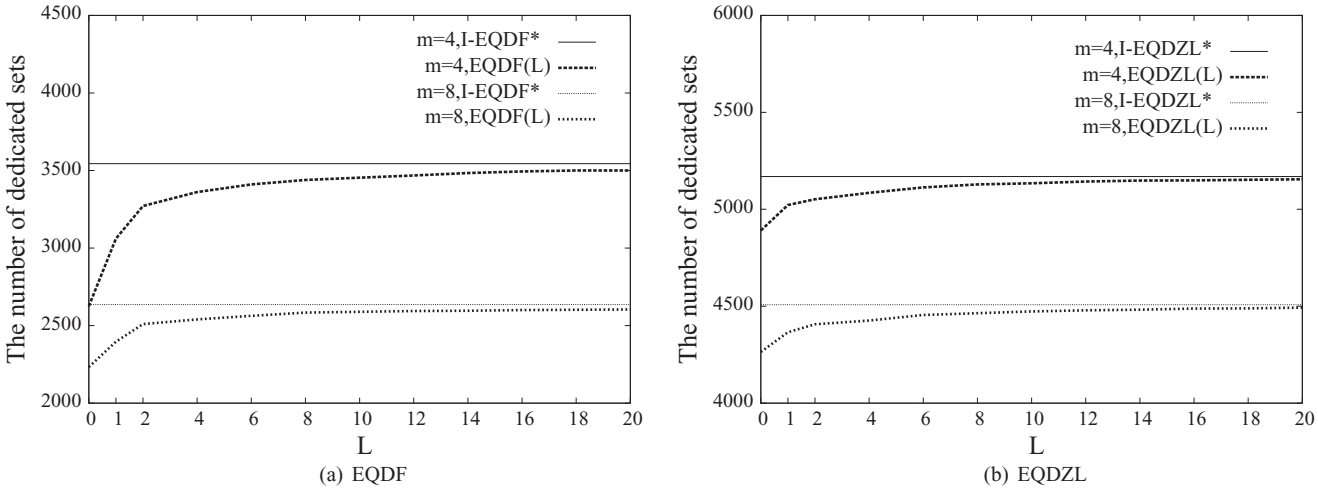
m	Interval length (L) (EQDZL)				
	1	2	4	8	16
4	[−1.0,0]	[−1.8,0.2]	[−1.8,2.2]	[−4.5,3.5]	[−8.3,7.7]
8	[−1.0,0]	[−1.3,0.7]	[−1.4,2.6]	[−3.4,4.6]	[−8.7,7.4]

Fig. 10 plots the number of task sets deemed schedulable under EQDF(L) and EQDZL(L) with different values of L with $m = 4$ and $m = 8$, and Tables 3 and 4 show corresponding intervals $[t^*, t^* + L]$. As shown in Fig. 10(a) and Table 3, even when length of L is one, such as [0.2,1.2] on $m = 4$ and [0,1.0] on $m = 8$, EQDF(L) finds higher than 84% and 86% schedulable task sets of those of I-EQDF* respectively. Fig. 10(b) and Table 4 also show that when length of L is one, such as [−1.0,0] on $m = 4$ and $m = 8$, EQDZL(L) finds higher than 97% and 96% schedulable task sets of those of I-EQDZL* respectively. Such a ratio increases sharply with a small value of L and grows slowly going beyond the 99th percentile when L is close to 20. This presents a good intuition into how long an interval of interest should be enough to include a schedulable value of k with a certain degree of probability.

Tables 3 and 4 show where those intervals $[t^*, t^* + L]$ are located, and these provide an idea into which interval of k should be examined for the efficient discovery of schedulable k value. We are then interested in how densely to sample a given interval. Recall that the OQDA-k and OQDA^{ZL}-k algorithms generate a set S that contains all schedulable values of k for a given task set. Each element $s \in S$ is an interval that holds a series of continuous k schedulable values. Let us define $S(L)$ as a subset of S such that $S(L)$ includes $s \in S$ if $|s| \geq L$. Fig. 11 shows the ratio of the size of $S(L)$ to the size of S. As shown in the figure, every interval $s \in S$ has a length greater than or equal to 0.001, and more than 99% of the intervals of S have a length greater than or equal to 0.005. The percentage drops significantly when L becomes larger than 0.1. This gives an insight into how densely our heuristic algorithm samples a given interval to locate a schedulable value of k.

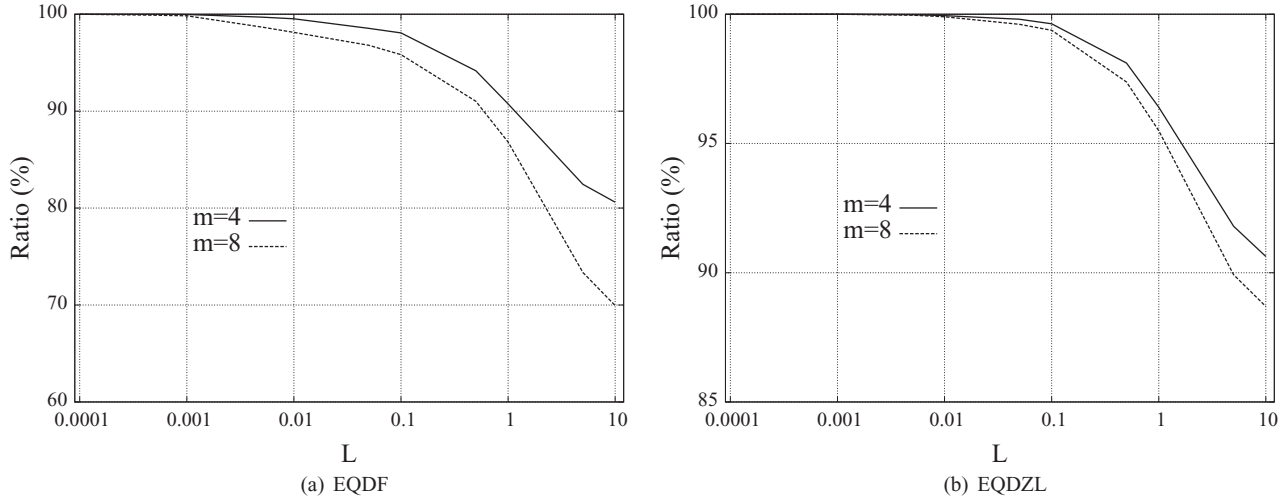
6.3. Optimal vs. heuristic solutions

Based on understanding of the characteristics of optimal solutions, we determine where and how densely the HQDA-k and HQDA^{ZL}-k algorithms examine k values. Tables 5 and 6 show the interval that



(a) EQDF (b) EQDZL

Fig. 10. The number of schedulable task sets with $[t^*, t^* + L]$ and those with $[-\infty, \infty]$.



(a) EQDF (b) EQDZL

Fig. 11. Ratio of the size of $S(L)$ to the size of S.

Table 5
The ratio of heuristics to optimal (EQDF).

	m	Length (L) and interval				
		1 [0,1]	4 [−2,2]	16 [−8,8]	32 [−16,16]	64 [−32,32]
Schedulability Ratio (%)	4	83.1	94.7	97.9	98.9	99.2
	8	85.1	95.3	97.5	98.6	99.0
Running time Ratio (10^{-3})	4	1.5	5.3	19.4	36.1	79.7
	8	0.2	1.0	3.7	9.0	17.7

Table 6
The ratio of heuristics to optimal (EQDZL).

	m	Length (L) and interval				
		1 [−1,0]	4 [−2,2]	16 [−8,8]	32 [−16,16]	64 [−32,32]
Schedulability Ratio (%)	4	95.7	98.1	99.3	99.6	99.8
	8	95.5	97.7	99.1	99.4	99.7
Running time Ratio (10^{-4})	4	2.0	7.4	24.5	50.6	93.4
	8	0.3	1.5	5.9	9.4	17.1

HQDA-k and HQDA^{ZL}-k examine for a given value of L, and we set the sampling step to 0.1. For example, when $L = 1$, HQDA-k examines the interval $[0, 1]$ with the step size of 0.1. This way, we can effectively reduce the search space of HQDA-k. Tables 5 and 6 also show the ratio of heuristic solutions to optimal in terms of schedulability and running time. As shown in Table 5, when $L = 4$, HQDA-k algorithm finds a solution 95% close to optimal with a shorter running time by three orders of magnitude. This translates into that HQDA-k has a comparable running time with EDF analysis but produces 34–37% better results than EDF. Table 6 describes similar trend of HQDA^{ZL}-k algorithm. Even when $L = 4$, HQDA^{ZL}-k algorithm finds a solution 98% close to optimal with a shorter running time by four orders of magnitude. This implies that HQDA^{ZL}-k has a comparable running time with EDZL analysis but produces 8%–9% better results than EDZL. When $L = 32$, heuristic solutions have only less than 1% loss of optimality reducing running time by two to four orders of magnitude.

6.4. Preemption and migration overheads

Under the EDF scheduling, a single job can preempt other jobs at most once when released (Liu, 2000); it cannot preempt after that since its priority remains the same. Following the same reasoning, a job can preempt at most once when released under the EQDF scheduling. Under the EQDZL scheduling, a job can preempt at most twice when released and when entering the zero-laxity state. Since the number of migrations is upper-bounded by the number of preemptions, the upper-bounds of the number of preemptions a job can cause

under EQDF and EQDZL can be used for those of the number of migrations.

7. Conclusion

This paper introduced a new concept of the quasi-deadline which can serve as a more effective scheduling parameter than deadline on multiprocessors. We proposed two new scheduling algorithms, EQDF (categorized into JFP scheduling) and EQDZL (categorized into JDP scheduling), that assign priority to jobs according to their quasi-deadlines ($d_i - k \cdot C_i$), and the control knob k allows to balance efficiently between urgency and parallelism in quasi-deadline assignment. We also presented an optimal solution to the quasi-deadline assignment subject to the proposed EQDF (EQDZL) schedulability analysis for design time, and a heuristic solution for runtime. We performed an extensive empirical study to gain good insights into how the search space of the heuristic solution can be effectively reduced. Based on our understanding of empirical results, we can reduce the running time of our proposed heuristic algorithm significantly (two to three orders of magnitude) at the expense of 1–5% optimality loss.

Incorporating such a balanced consideration into job-level priority assignment offers a significant improvement over the state-of-the-art deadline-based scheduling algorithms. However, there could exist some other effective ways of capturing urgency and parallelism than quasi-deadlines. This suggests our future research direction of advancing understanding of real-time multiprocessor scheduling and translating new understanding into better scheduling strategies.

Acknowledgments

This work was supported in part by BSRP (NRF-2010-0006650, NRF-2012R1A1A1014930), NCRC (2012-0000980), IITP (2011-10041313, 14-824-09-013) and KIAT (M002300089) funded by the Korea Government (MEST/MSIP/MOTIE). This work was also supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (NRF-2014R1A1A1035827).

References

AUTOSAR, 2009. AUTOSAR Release 4.0 Specification.
Anderson, J.H., Srinivasan, A., 2000. Early-release fair scheduling. In: Proceedings of Euromicro Conference on Real-Time Systems (ECRTS), pp. 35–43.
Andersson, B., 2008. Global static-priority preemptive multiprocessor scheduling with utilization bound 38%. In: Proceedings of International Conference on Principles of Distributed Systems, pp. 73–88.
Andersson, B., Baruah, S., Jonsson, J., 2001. Static-priority scheduling on multiprocessors. In: Proceedings of IEEE Real-Time Systems Symposium (RTSS), pp. 193–202.
Andersson, B., Bletsas, K., Baruah, S., 2008. Scheduling arbitrary-deadline sporadic task systems on multiprocessor. In: Proceedings of IEEE Real-Time Systems Symposium (RTSS), pp. 385–394.
Andersson, B., Jonsson, J., 2000. Fixed-priority preemptive multiprocessor scheduling: to partition or not to partition. In: Proceedings of IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), pp. 337–346.
Audsley, N., 1991. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times: Technical Report YCS164. Department of Computer Science, University of York.
Audsley, N., 2001. On priority assignment in fixed priority scheduling. Inf. Process. Lett. 79, 39–44.
Back, H., Chwa, H.S., Shin, I., 2012. Schedulability analysis and priority assignment for global job-level fixed-priority multiprocessor scheduling. In: Proceedings of IEEE Real-Time Technology and Applications Symposium (RTAS), pp. 297–306.
Baker, T., 2005. An analysis of EDF schedulability on a multiprocessor. IEEE Trans. Parallel Distrib. Syst. 16, 760–768.
Baker, T.P., 2003. Multiprocessor EDF and deadline monotonic schedulability analysis. In: Proceedings of IEEE Real-Time Systems Symposium (RTSS), pp. 120–129.
Baker, T.P., Baruah, S., 2009a. An analysis of EDF schedulability for arbitrary sporadic task systems. Real-Time Syst. 43, 3–24.
Baker, T.P., Baruah, S.K., 2009b. Sustainable multiprocessor scheduling of sporadic task systems. In: Proceedings of Euromicro Conference on Real-Time Systems (ECRTS), pp. 141–150.
Baker, T.P., Cirinei, M., 2006. A necessary and sometimes sufficient condition for the feasibility of sets of sporadic hard-deadline tasks. In: Proceedings of IEEE Real-Time Systems Symposium (RTSS), pp. 178–190.
Baker, T.P., Cirinei, M., Bertogna, M., 2008. EDZL scheduling analysis. Real-Time Syst. 40, 264–289.
Baruah, S., 2007. Techniques for multiprocessor global schedulability analysis. In: Proceedings of IEEE Real-Time Systems Symposium (RTSS), pp. 119–128.
Baruah, S., Bonifaci, V., Marchetti-Spaccamela, A., Stiller, S., 2009. Implementation of a speedup-optimal global EDF schedulability test. In: Proceedings of Euromicro Conference on Real-Time Systems (ECRTS), pp. 259–268.
Baruah, S., Cohen, N.K., Plaxton, C.G., Varvel, D.A., 1996. Proportionate progress: a notion of fairness in resource allocation. Algorithmica 15, 600–625.
Bertogna, M., Baruah, S., 2011. Tests for global EDF schedulability analysis. J. Syst. Archit. 57, 487–497.
Bertogna, M., Buttazzo, G., Marinoni, M., Caccamo, M., 2010. Preemption points placement for sporadic task sets. In: Proceedings of Euromicro Conference on Real-Time Systems (ECRTS), pp. 251–260.
Bertogna, M., Cirinei, M., 2007. Response-time analysis for globally scheduled symmetric multiprocessor platforms. In: Proceedings of IEEE Real-Time Systems Symposium (RTSS), pp. 149–160.
Bertogna, M., Cirinei, M., Lipari, G., 2005a. Improved schedulability analysis of EDF on multiprocessor platforms. In: Proceedings of Euromicro Conference on Real-Time Systems (ECRTS), pp. 209–218.
Bertogna, M., Cirinei, M., Lipari, G., 2005b. New schedulability tests for real-time task sets scheduled by deadline monotonic on multiprocessors. In: Proceedings of International Conference on Principles of Distributed Systems, pp. 306–321.
Bertogna, M., Cirinei, M., Lipari, G., 2009. Schedulability analysis of global scheduling algorithms on multiprocessor platforms. IEEE Trans. Parallel Distrib. Syst. 20, 553–566.
Bertogna, M., Marinoni, O.X.M., Esposito, F., Buttazzo, G., 2011. Optimal selection of preemption points to minimize preemption overhead. In: Proceedings of Euromicro Conference on Real-Time Systems (ECRTS), pp. 217–227.
Cho, H., Ravindran, B., Jensen, E.D., 2006. An optimal real-time scheduling algorithm for multiprocessors. In: Proceedings of IEEE Real-Time Systems Symposium (RTSS), pp. 101–110.
Davis, R., Burns, A., 2009. Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. In: Proceedings of IEEE Real-Time Systems Symposium (RTSS), pp. 398–409.
Davis, R., Kato, S., 2012. FPSL, FPCL and FPZL schedulability analysis. Real-Time Syst. 48, 750–788.
Davis, R.I., Burns, A., 2011. FPZL schedulability analysis. In: Proceedings of IEEE Real-Time Technology and Applications Symposium (RTAS), pp. 245–256.
Erickson, J.P., Anderson, J.H., 2012. Fair lateness scheduling: reducing maximum lateness in G-EDF-like scheduling. In: Proceedings of Euromicro Conference on Real-Time Systems (ECRTS), pp. 3–11.
Funaoka, K., Kato, S., Yamasaki, N., 2008. Work-conserving optimal real-time scheduling on multiprocessors. In: Proceedings of Euromicro Conference on Real-Time Systems (ECRTS), pp. 13–22.
Goossens, J., Funk, S., Baruah, S., 2003. Priority-driven scheduling of periodic task systems on multiprocessors. Real-Time Syst. 25, 187–205.
Lee, J., Easwaran, A., Shin, I., 2010. LLF schedulability analysis on multiprocessor platforms. In: Proceedings of IEEE Real-Time Systems Symposium (RTSS), pp. 25–36.
Lee, J., Easwaran, A., Shin, I., 2011a. Maximizing contention-free executions in multiprocessor scheduling. In: Proceedings of IEEE Real-Time Technology and Applications Symposium (RTAS), pp. 235–244.
Lee, J., Easwaran, A., Shin, I., 2012. Laxity dynamics and LLF schedulability analysis on multiprocessor platforms. Real-Time Syst. 48, 716–749.
Lee, J., Easwaran, A., Shin, I., Lee, I., 2011b. Zero-laxity based real-time multiprocessor scheduling. J. Syst. Softw. 84, 2324–2333.
Lee, S.K., 1994. On-line multiprocessor scheduling algorithms for real-time tasks. In: Proceedings of IEEE Region 10's Ninth Annual International Conference, pp. 607–611.
Leung, J., Whitehead, J., 1982. On the complexity of fixed-priority scheduling of periodic real-time tasks. Perform. Eval. 2, 237–250.
Leung, J.Y.T., 1989. A new algorithm for scheduling periodic, real-time tasks. Algorithmica 4, 209–219.
Levin, G., Funk, S., Sadowski, C., Pye, I., Brandt, S., 2010. DP-FAIR: a simple model for understanding optimal multiprocessor scheduling. In: Proceedings of Euromicro Conference on Real-Time Systems (ECRTS), pp. 3–13.
Liu, C., Layland, J., 1973. Scheduling algorithms for multi-programming in a hard-real-time environment. J. ACM 20, 46–61.
Liu, J., 2000. Real-Time Systems. Prentice Hall.
Regnier, P., Lima, G., Massa, E., Levin, G., Brandt, S., 2011. RUN: optimal multiprocessor real-time scheduling via reduction to uniprocessor. In: Proceedings of IEEE Real-Time Systems Symposium (RTSS), pp. 104–115.

Hoon Sung Chwa received B.S. and M.S. degrees in Computer Science in 2009 and 2011, respectively, from KAIST (Korea Advanced Institute of Science and Technology), South Korea. He is currently working toward the Ph.D. degree in Computer Science from KAIST. His research interests include system design and analysis with timing guarantees and resource management in real-time embedded systems and cyber-physical systems. He won two best paper awards from the 33rd IEEE Real-Time Systems Symposium (RTSS) in 2012 and from the IEEE International Conference on Cyber-Physical Systems, Networks, and Applications (CPSNA) in 2014.

Hyoungbu Back is a Ph.D. student at the Department of Computer Science at KAIST (Korea Advanced Institute of Science and Technology), South Korea. He received B.S.

degree in Computer Science and Engineering from Konkuk University, South Korea in 2010 and M.S. degree in Computer Science from KAIST (Korea Advanced Institute of Science and Technology), South Korea in 2012. His research interests include system design and analysis with timing guarantees and resource management in real-time embedded systems and cyber-physical systems. He won the best paper award from the 33rd IEEE Real-Time Systems Symposium (RTSS) in 2012.

Jinkyu Lee is an assistant professor in Department of Computer Science and Engineering, Sungkyunkwan University (SKKU), Republic of Korea, where he joined in 2014. He received the B.S., M.S., and Ph.D. degrees in computer science from the Korea Advanced Institute of Science and Technology (KAIST), Republic of Korea, in 2004, 2006, and 2011, respectively. He has been a research fellow/visiting scholar in the Department of Electrical Engineering and Computer Science, University of Michigan until 2014. His research interests include system design and analysis with timing guarantees, QoS support, and resource management in real-time embedded systems and cyber-physical systems. He won the best student paper award from the 17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) in 2011, and the Best Paper Award from the 33rd IEEE Real-Time Systems Symposium (RTSS) in 2012.

Kieu-My Phan received B.S. degrees in Computer Science in 2013 from KAIST (Korea Advanced Institute of Science and Technology), South Korea. Her research interests include system design and analysis with timing guarantees and resource management in real-time embedded systems and cyber-physical systems.

Insik Shin is currently an associate professor in Department of Computer Science at KAIST, South Korea, where he joined in 2008. He received a B.S. from Korea University, an M.S. from Stanford University, and a Ph.D. from University of Pennsylvania all in Computer Science in 1994, 1998, and 2006, respectively. He has been a post-doctoral research fellow at Malardalen University, Sweden, and a visiting scholar at University of Illinois, Urbana-Champaign until 2008. His research interests lie in cyber-physical systems and real-time embedded systems. He is currently a member of Editorial Boards of Journal of Computing Science and Engineering. He has been co-chairs of various workshops including satellite workshops of RTSS, CPSWeek and RTCSA and has served various program committees in real-time embedded systems, including RTSS, RTAS, ECRTS, and EMSOFT. He received best paper awards, including Best Paper Awards from RTSS in 2003 and 2012, Best Student Paper Award from RTAS in 2011, and Best Paper runner-ups at ECRTS and RTSS in 2008.