2012 IEEE 18th Real Time and Embedded Technology and Applications Symposium

Schedulability Analysis and Priority Assignment for Global Job-Level Fixed-Priority Multiprocessor Scheduling

Hyoungbu Back, Hoon Sung Chwa, Insik Shin
Dept. of Computer Science, KAIST, Republic of Korea
insik.shin@cs.kaist.ac.kr

Abstract

Unlike uniprocessor scheduling, EDF (categorized into job-level fixed-priority (JFP) scheduling) shows relatively poor performance on global multiprocessor scheduling. As no other global JFP multiprocessor algorithms are illuminated beyond EDF, this work proposes one, called EQDF (earliest quasi-deadline first), as a generalization of EDF. 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. This paper then seeks to explore how it can improve the schedulability of global JFP scheduling. In addition to providing a new schedulability analysis for EQDF scheduling, it addresses the problem of priority assignment under EQDF by controlling the system-level control knob. It presents optimal and heuristic solutions to the problem subject to our proposed EQDF analysis. Our empirical results show the proposed heuristic solution outperforms EDF significantly, giving close to optimal results.*

1 Introduction

Real-time scheduling has been extensively studied for several decades over various scheduling categories. In general, priority-driven preemptive scheduling algorithms can fall into three categories: task-level fixed-priority (TFP), job-level fixed-priority (JFP)[1], and job-level dynamic-priority (JDP) algorithms. A 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) and DM (deadline-monotonic). A JFP 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 good example is EDF (earliest deadline first). Under JDP scheduling, each job is assigned a priority that can change dynamically during the job's execution. A good example is LST (least-slack-time first).

Those three categories have been well studied in uniprocessor scheduling. DM [23], EDF [24], and LST [18] are proved to be optimal in each of the three categories, respectively. However, they have not been developed as maturely yet in multiprocessor scheduling as in uniprocessor scheduling. In particular, JFP multiprocessor scheduling has not been yet much explored beyond EDF.

A considerable amount of work has been made to study TFP multiprocessor scheduling. Many heuristic schemes were proposed for priority assignment in this category [3, 16], and an optimal priority assignment subject to some given schedulability test was presented under some conditions [16].

There has been a growing attention to JDP multiprocessor scheduling. The pFair algorithm [12] is known as optimal for implicit-deadline task systems (deadline equal to task period) but no longer optimal for constrained-deadline task systems (deadline no larger than task period). Recent studies showed that JDP algorithms, such as EDZL (EDF until zero-laxity) [22, 9], FPZL (fixed-priority until zero-laxity) [17], and LST [20], outperform TFP and JFP algorithms. 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.

A substantial body of research was made to investigate global EDF multiprocessor scheduling. Unlike uniprocessor scheduling, EDF is no longer optimal but exhibits significantly lower performance in multiprocessor scheduling [19, 7]. Despite this, however, little work has been reported to explore JFP scheduling beyond EDF on multiprocessors. This motivates the research described in this paper with a curiosity to understand how effective JFP scheduling algorithms can be on multiprocessors. We believe that EDF performs poor on multiprocessors because it assigns priority with a sole focus on the deadline constraints (or "urgency") but neglecting the "parallelism" aspect of multiprocessor platforms. According to a few studies [3, 16, 20], assigning a higher priority to the job with a larger execution time helps to maximize the potential for concurrency. This is because it is generally easier to schedule on multi-

[1]This category is also called task-level dynamic-priority in the literature [25]. However, we use the term JFP in order to emphasize the static nature of the priority of an individual job.

1080-1812/12 $26.00 © 2012 IEEE
DOI 10.1109/RTAS.2012.33

processors a larger number of jobs with shorter remaining execution times than a smaller number of jobs with longer remaining execution times even though they have the same remaining execution times in total. 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 $q_i = d_i - kC_i$, where k is a real value that the system designer configures statically and/or the system determines dynamically. We then introduce a new JFP 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 $k = 0$.

We present a new schedulability analysis for EQDF scheduling, and this facilitates the comparison between the only, well-known JFP algorithm, EDF, and a new JFP algorithm, EQDF, on the basis of schedulability. A critical factor to improve the effectiveness of EQDF scheduling is quasi-deadline assignment. In this paper, we consider the k-controlled quasi-deadline assignment that determines the value of k to make a given task set deemed schedulable 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 solution value of k, if there exists any. Our empirical results show that the proposed EQDF optimal solution does not simply dominate EDF but outperforms it significantly. Our EQDF solutions find 40%-60% more task sets deemed schedulable than the state-of-the-art EDF analysis. Our empirical results also reveal that the OQDA-k algorithm employs a certain amount of 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 (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 40%-57% more task sets deemed schedulable than EDF with a comparable running time.

Related Work. Several schedulability tests for global EDF scheduling of sporadic task systems have been developed [7, 19, 10, 14, 26]. Baker [7] 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 take the contra-positive of this to derive a sufficient schedulability test. Building upon Baker's work, Bertogna et al. [14, 26] 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.

Priority assignment has been studied in global TFP multiprocessor scheduling. Audsley [4, 5] developed an optimal priority assignment (OPA) policy for some given test on uniprocessor platforms. Davis and Burns [16] showed that Audsley's OPA algorithm is applicable to the multiprocessor case when a given test satisfies some conditions. Several heuristic priority assignments are devised [15, 1, 3, 16] for global TFP multiprocessor scheduling, and some of them are related to our notion of quasi-deadline. Andersson and Jonsson [3] 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 [16] developed the D-CMPO policy that assigns priorities according to $D_i - C_i$, where D_i is a task's relative deadline. However, no work addresses optimal priority assignment in global JFP multiprocessor scheduling.

Organization. Section 2 presents system models and terminologies. Section 3 derives a new, interference-based schedulability condition for EQDF scheduling on the basis of understanding of worst-case inter-task interference scenarios under EQDF. Section 4 formulates the k-controlled quasi-deadline assignment problem and proposes optimal and heuristic solutions. Section 5 provides empirical results that illuminate the characteristics of optimal solutions thoroughly and evaluates the effectiveness of our EQDF solutions. Section 6 concludes with future work.

2 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 ($C_i \leq D_i \leq T_i$). A task utilization U_i is defined as C_i/T_i, and the system utilization U_{sys} is defined as the total utilization of a task set. A task τ_i invokes a series of jobs, each separated from its predecessor by at least T_i time units. 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 is as $q_i^h = d_i^h - kC_i$, 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 system-wide variable that the system designer determines statically and/or the system sets dynamically. We also assume that a single job of a task cannot execute in parallel.

Multiprocessor scheduling. We consider global job-level fixed-priority preemptive scheduling on m identical unit-capacity processors. In particular, we consider the EQDF scheduling algorithm that assigns priority according to quasi-deadlines; a job with an earlier quasi-deadline has

a higher priority. Like most existing studies in multiprocessor scheduling (for example, see [12]), we assume that the system does not incur any penalty when a job is preempted or when a job is migrated from one processor to another.

Interference. 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 $[a, b)$ is defined 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:

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

3 EQDF Schedulability Analysis

In this section, we derive a schedulability condition for the EQDF scheduling algorithm. We first introduce existing interference-based schedulability analysis of any work-conserving scheduling algorithms (Section 3.1), then present the worst-case interference scenarios between two tasks under EQDF scheduling (Section 3.2), compute an upper bound on the amount of execution that a task can interfere with another task on the basis of those worst-case scenarios under EQDF (Section 3.3), and finally provide a schedulability condition for EQDF scheduling (Section 3.4) and its iterative version (Section 3.5).

3.1 Interference-based Schedulability Condition

Schedulability analysis of global multiprocessor scheduling algorithms is presented based on the concept of worst-case interference [7, 26]. A job of a task τ_j can meet its deadline, if and only if the total interference on task τ_j over the job's scheduling window is less than or equal to its slack time $D_j - C_j$ [26]. Let J_j^* denote the job instance that receives the maximal total interference among the jobs of τ_j, and let $\overline{I}_j$ denote the worst-case interference on task τ_j. Then, notice that by definition

$$\overline{I}_j = \max_h (I_j(r_j^h, d_j^h)) = I_j(r_j^*, d_j^*). \quad (2)$$

For notational convenience, we also define

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

With the above reasoning and notations, the necessary and sufficient schedulability condition of global multiprocessor scheduling algorithms is derived as follows [14, 26].

Lemma 1 (from [14, 26]) *A task set τ is schedulable on a multiprocessor composed by m identical processors iff for each task τ_j*

$$\sum_{i \neq j} \min(\overline{I}_{j \leftarrow i}, D_j - C_j + 1) < m(D_j - C_j + 1) \quad (4)$$

Note that it is known to be difficult to compute $\overline{I}_{j \leftarrow i}$ precisely, so exiting approaches [14, 26] have used an upper bound on the interference, and therefore the test derived is changed to only a sufficient condition. Then, it needs 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.

3.2 Worst-Case Interference Scenarios

In this 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 [7]. 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 us consider the situation in which jobs of τ_i are periodically released and one of the jobs has the same quasi-deadline as that of J_j^*. We refer to this situation as *quasi-deadline alignment*. For notational convenience, we denote by J_i^+ the job of τ_i which has the same quasi-deadline as that of J_j^* at quasi-deadline alignment. We refer to the difference between the deadline of J_j^* and the deadline of J_i^+ as $\phi = d_i^+ - d_j^*$. If the deadline of J_i^+ is before that of J_j^* ($\phi \leq 0$), the carry-out job of the scheduling window of J_j^* has a lower priority than J_j^*, so the carry-out job cannot interfere with J_j^*. Therefore, for the same reason as EDF scheduling, quasi-deadline alignment is the worst-case interference scenario when $\phi \leq 0$ (depicted in Figure 1(a)). If $\phi > 0$, the carry-out job can also interfere with J_j^*, and the interference of the job on J_j^* is maximized until $\phi \leq (D_i - C_i)$. If $0 < \phi \leq (D_i - C_i)$, the carry-out job corresponds with J_i^+ and makes the maximal interference of C_i on J_j^* when it starts execution as soon as it is

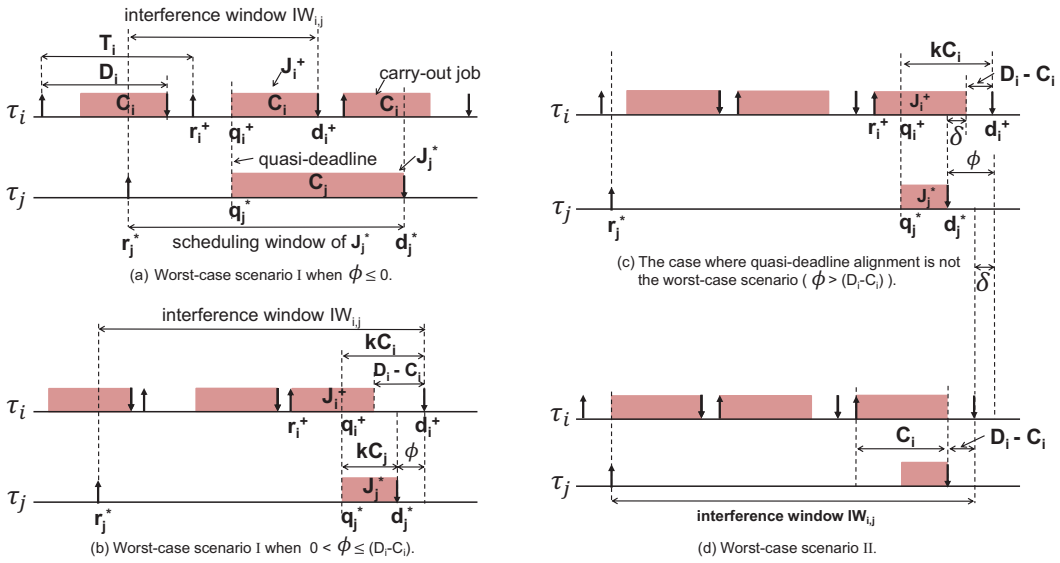


Figure 1. Worst-case scenarios with $k = 1$

released as described in Figure 1(b). Therefore, the quasi-deadline assignment is the worst-case interference scenario in the case of $0 < \phi \leq (D_i - C_i)$. This is because moving deadlines of τ_i's jobs later until one of them meets the quasi-deadline of J_j^* does not incur any loss of the interference of the carry-out job and does maximize the interference of both carry-in and body jobs.

However, if $\phi > (D_i - C_i)$, there can exist a situation (shown in Figure 1(c)) where the interference of the carry-out job is not maximized even though the job has a higher priority than J_j^* at quasi-deadline alignment. This is because the carry-out job is able to have a δ amount of execution remaining, where $\delta > 0$, after the deadline of J_j^* (shown in Figure 1-(c)). We can see that moving the deadlines of τ_i's jobs earlier can increase (and cannot decrease) the total interference of τ_i on J_j^*, compared to that in the quasi-deadline alignment situation. This is because, firstly, moving those deadlines earlier by δ will not affect its priority order with J_j^*, and secondly, it will add the interference of the carry-out job into the total interference exactly as much as δ but decrease the interference of a carry-in job by at most as much as δ. Thereby, 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 Figure 1(d)).

To summarize, we identify two types of the worst-case interference scenarios according to the relationship between ϕ and $(D_i - C_i)$ at the quasi-deadline alignment situation. J_i^+ has the same quasi-deadline as that of J_j^*, that is, $d_i^+ - kC_i = d_j^* - kC_j$, so the deadline of J_i^+ is determined by $d_i^+ = d_j^* - kC_j + kC_i$. The difference ϕ can be expressed in only relative terms as follow:

$$\begin{aligned}\phi &= d_i^+ - d_j^* = d_j^* - kC_j + kC_i - d_j^* \\ &= kC_i - kC_j.\end{aligned} \tag{5}$$

We finally define two types of the worst case scenarios according to the relationship between ϕ and $(D_i - C_i)$.

Worst-case scenario I. In the case where it holds $(kC_i - kC_j) \leq (D_i - C_i)$, 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. In the other case of $(kC_i - kC_j) > (D_i - C_i)$, the worst-case interference scenario is that the jobs of τ_i are periodically released in a way that one of its jobs is released C_i before the deadline of J_j^*.

In both two scenarios, the carry-in and carry-out jobs start executing as close as possible to the scheduling window of J_j^* to maximize their interference.

3.3 Bounding Interference

For presentational convenience, we call a job of τ_i the *last interfering* job (denoted by $J_i^\diamond$) of τ_i on J_j^* if the job is released latest among those that can interfere with J_j^* over its scheduling window. We also 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 $J_i^\diamond$, that is, $IW_{i,j} = [r_j^*, d_i^\diamond)$. Figure 1 shows the interference window in each scenario. Note that under EDF scheduling, since the worst-case interference scenario is the one where $d_i^\diamond = d_j^*$, $IW_{i,j}$ is exactly the same as $[r_j^*, d_j^*)$. Under EQDF scheduling, however, $d_i^\diamond$ can be before or after, or equal to d_j^*.

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

though $J_i^\diamond$ can be the carry-out job of the scheduling window of J_j^* (i.e., the length of the interference window is greater than that of the scheduling window of J_j^*), $J_i^\diamond$ can make the maximal interference of C_i on J_j^* under the both worst-case interference scenarios.

In the worst-case scenario I, $J_i^\diamond$ has the same quasi-deadline as that of J_j^*, so the deadline $d_i^\diamond$ is determined by $d_j^* - kC_j + kC_i$. If the deadline of $J_i^\diamond$ is before the release time of J_j^* (i.e., $d_i^\diamond \leq r_j^*$), 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 $[r_j^*, d_i^\diamond)$ can actually interfere with J_j^*. In this case, the length of the interference window $IW_{i,j}$ is computed as

$$\begin{aligned} |IW_{i,j}| &= d_i^\diamond - r_j^* = d_j^* - kC_j + kC_i - r_j^* \\ &= D_j - kC_j + kC_i. \end{aligned} \tag{6}$$

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 $\overline{I}_{j \leftarrow i}$ is bounded by

$$\begin{aligned} \overline{I}_{j \leftarrow i} \leq \Big[& \Phi_i(D_j - kC_j + kC_i) \cdot C_i + \\ & \min\left(C_i, D_j - kC_j + kC_i - \Phi_i(D_j - kC_j + kC_i) \cdot T_i\right) \Big]_0 \end{aligned} \tag{9}$$

where it is bounded by zero when $D_j - kC_j + kC_i < 0$.

In the worst-case scenario II, $J_i^\diamond$ is released such that its deadline $d_i^\diamond$ 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 $\overline{I}_{j \leftarrow i}$ is bounded by

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

3.4 EQDF Schedulability Analysis

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

We denote by $I_{j \leftarrow i}^{EQDF}(L, k)$ an upper bound on the interference $\overline{I}_{j \leftarrow i}$ in any interval of length L under the EQDF scheduling policy with a system-wide variable k and described as

$$I_{j \leftarrow i}^{EQDF}(L, k) = \tag{11}$$

$$\begin{cases} \Big[\Phi_i(L - kC_j + kC_i) \cdot C_i + \\ \quad \min\left(C_i, L - kC_j + kC_i - \Phi_i(L - kC_j + kC_i) \cdot T_i\right) \Big]_0 \\ \qquad \text{if } (kC_i - kC_j) \leq (D_i - C_i) \\ \Phi_i(L + D_i - C_i) \cdot C_i + \\ \quad \min\left(C_i, L + D_i - C_i - \Phi_i(L + D_i - C_i) \cdot T_i\right) \\ \qquad \text{otherwise.} \end{cases}$$

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 by m identical processors if for each task τ_j*

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

Note that the above EQDF schedulability test is a generalization of the existing EDF schedulability test [14].

3.5 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 [13, 26] are introduced to reduce such pessimism effectively. Let S_j denote the slack of a task τ_j and is defined as

$$S_j = D_j - C_j - \left\lfloor \frac{\sum_{i \neq j} min(\overline{I}_{j \leftarrow i}, D_j - C_j + 1)}{m} \right\rfloor \tag{13}$$

when (13) is positive. A lower bound S_j^{lb} on the slack S_j of a task τ_j is then given by

$$S_j^{lb} = D_j - C_j - \left\lfloor \frac{\sum_{i \neq j} min(I_{j \leftarrow i}^{EQDF}(D_j, k), D_j - C_j + 1)}{m} \right\rfloor \tag{14}$$

when this term is positive.

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

4 Quasi-Deadline Assignment

In this section, we consider the problem of priority assignment under job-level fixed-priority 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_i^h is assigned a quasi-deadline q_i^h equal to $d_i^h - kC_i$ and the task set is deemed schedulable under global EQDF scheduling by the schedulability test in Eq. (12). 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 Eq. (12). 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 4.1 presents an optimal solution to the problem, and Section 4.2 discusses heuristic solutions.

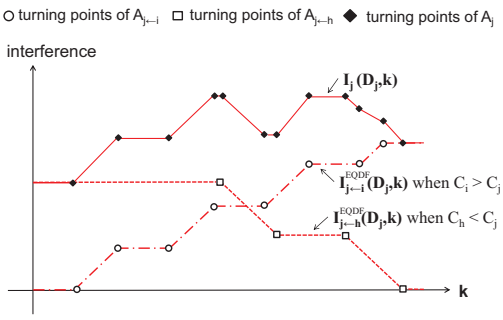


Figure 2. example graph representing interference made by higher priority tasks according to value k

4.1 Optimal quasi-deadline assignment

In this 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 (task set τ)

```
1:  for each task τ_j do
2:      for each task τ_i do
3:          if τ_j ≠ τ_i, construct A_{j←i}
4:      end for
5:  end for
6:  A_j ← ⋃_{∀τ_i} A_{j←i}
7:  for each task τ_j do
8:      construct S_j from A_j
9:  end for
10: S ← ⋂_{∀τ_j} S_j
11: 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. The interference of a task τ_i with a job of task τ_j varies with k (c.f. $I^{EQDF}_{j\leftarrow i}(D_j,k)$ 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 is 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 $I^{EQDF}_{j\leftarrow i}(D_j,k)$ monotonically increasing with a growing value of k when $C_i > C_j$. Likewise, $I^{EQDF}_{j\leftarrow i}(D_j,k)$ decreases monotonically as k increases if $C_i < C_j$. When $C_i = C_j$, on the other hand, $I^{EQDF}_{j\leftarrow i}(D_j,k)$ remains constant because the quasi-deadlines of τ_i and τ_j and their priorities remain relatively the same even though k varies.

Figure 2 illustrates the impact of k on $I^{EQDF}_{j\leftarrow i}(D_j,k)$. A value of k is said to be a *turning point* of $I^{EQDF}_{j\leftarrow i}(D_j,k)$ if $I^{EQDF}_{j\leftarrow i}(D_j,k)$ changes its slope at this k value. The formula $I^{EQDF}_{j\leftarrow i}(D_j,k)$ defined in Eq. (11) is a combination of linear and constant shape functions as shown in Figure 2. By definition, $I^{EQDF}_{j\leftarrow i}(D_j,k)$ 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 $kC_i - kC_j \leq (D_i - C_i)$ (c.f. 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 $I^{EQDF}_{j\leftarrow i}(D_j,k)$ is a monotonically increasing function, it starts from the lower-bound point increasing linearly as k increases. It then becomes staying 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 comes in. The process of increasing linearly and staying constant repeats until it reaches the upper-bound point. Therefore, $I^{EQDF}_{j\leftarrow i}(D_j,k)$ has a finite number of turning points. We can then easily calculate all turning points with this understanding of the dynamics of $I^{EQDF}_{j\leftarrow i}(D_j,k)$.

Let $A_{j\leftarrow i}$ denote a set of all turning points of $I^{EQDF}_{j\leftarrow i}(D_j,k)$. 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 ($I^{EQDF}_{j\leftarrow i}(D_j,k)$). 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 $I^{EQDF}_{j\leftarrow i}(D_j,k)$, then it is also a turning point of $I_j(D_j,k)$. So we can construct A_j as the

union of all the turning points of $I_{j\leftarrow i}^{EQDF}(D_j,k)$ for all tasks $i \neq j$. Even though $I_j(D_j,k)$ is a combination of linear and constant functions, unlike $I_{j\leftarrow i}^{EQDF}(D_j,k)$, $I_j(D_j,k)$ does not increase (or decrease) monotonically over all k values. However, $I_j(D_j,k)$ remains constant or increase (or decrease) 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. Consider two consecutive turning points of τ_j ($p_j^h \in A_j$ and $p_j^{h+1} \in A_j$). That is, there does not exist $p' \in A_j$ such that $p_j^h < p' < p_j^{h+1}$. We consider four cases in constructing S_j with p_j^h and p_j^{h+1} according to their schedulability.

Firstly, if both p_j^h and p_j^{h+1} are schedulable for τ_j, then all the values of k within the interval $[p_j^h, p_j^{h+1}]$ are also schedulable. This is because $I_j(D_j,k)$ remains constant or moves linearly within the two 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. 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 four 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. For each element $s \in S$, there exists $s_j \in S_j$ for all tasks τ_j such that $s \in s_j$.

Theorem 2 *The OQDA-k algorithm is optimal.*

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 $I_{j\leftarrow i}^{EQDF}(D_j,k)$ for tasks τ_i and τ_j. 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 6, 8, and 10 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. 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 $I_{j\leftarrow i}^{EQDF}(D_j,k)$ 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 2. □

Complexity. We denote the number of tasks in a task set by n. For each task τ_j, the OQDA-k algorithm performs as many schedulability tests as $|A_j|$ at most. The complexity of constructing S as the intersection of all S_j is $O(n \cdot |S_j|^2)$. Since $|S_j| \leq |A_j|$, the running time of this algorithm is thereby $O(n \cdot |A_j|^2)$.

Algorithm 2 HQDA-k (task set 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 then
4:         S ← S ∪ {k}
5:         return S
6:     end if
7: end for
8: return S
```

4.2 Heuristic Priority Assignment

The OQDA-k algorithm identifies and explores 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 algorithm has a fast growing number of turning points to explore and thereby its running time also increases rapidly. Thus, the OQDA-k algorithm is good for optimal quasi-deadline assignment at design time. However, it may not be appropriate for reconfiguring quasi-deadlines dynamically at run-time.

Therefore, we introduce a heuristic algorithm (HQDA-k) that finds a schedulable value of k in a sub-optimal but efficient way. Algorithm 2 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 stepsize K_s, and the algorithm checks with $k = K_1, K_1 + K_s, K_1 + 2K_s, \cdots, K_2$. The evaluation of this heuristic algorithm is provided in the next section.

5 Experimental results

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

Simulation environment. Task sets are generated based on a technique proposed earlier [6], which has also been used in many previous studies (e.g., see [2, 26, 21]). We

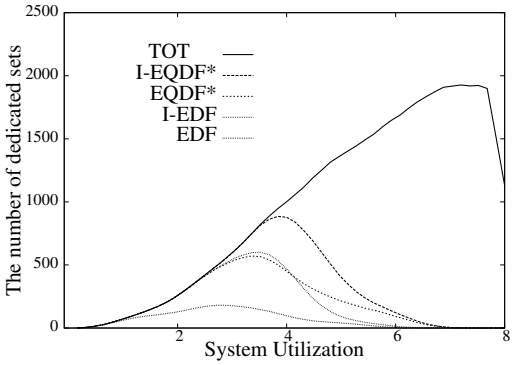


Figure 3. Schedulability of EDF and EQDF

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[2]. Although our proposed EQDF analysis is applicable to both implicit and constraint deadline models, because of page limit, we only show the results of the implicit deadline model: D_i is set equal to T_i. For each task utilization model, we repeat the following procedure to generate 1,000 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 [8, 11].
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 1,000 task sets for an individual task utilization model, resulting in 10,000 task sets in total.

EDF vs. EQDF. Our first simulations were run to compare the EQDF schedulability tests derived in this paper with the existing tests of the only, well-known job-level fixed-priority (JFP) scheduling algorithm, EDF. Aiming at seeing how effectively the schedulability of JFP scheduling can improve with EQDF, the quasi-deadline control knob k is set to an optimal value for each task set through the OQDA-k algorithm. The following schedulability tests were considered:

- the EDF test in [26] (EDF)
- the iterative EDF test in [26] (I-EDF)
- our EQDF test in Theorem 1 with an optimal value k^* (EQDF*)
- our iterative EQDF test in Eq. (14) with an optimal value k^* (I-EQDF*)

[2]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.

Figure 3 compares EDF and EQDF on the basis of schedulability with $m = 8$. Each line represents the number of task sets deemed schedulable by one specific test, except that the curve labeled with TOT; it represents an upper bound on the feasible task sets. Figure 3 shows that EQDF* significantly outperforms EDF. In particular, I-EQDF* dominates all the other tests, making significant differences when the system utilization is between $m/3$ and $2m/3$. Table 1 also shows that EQDF* finds 160%-320% more task sets schedulable than EDF does, and I-EQDF* detects 40%-60% more task sets schedulable than I-EDF does on 4 and 8 processors, respectively. Our simulation results indicate that the schedulability of JFP scheduling can improve significantly when the priorities of individual jobs are well assigned. That is, it is important to determine a good value of k for the effectiveness of EQDF scheduling. The OQDA-k algorithm is able to find an optimal value of k. However, as shown in Table 1, it is computationally expensive. Its running time is four orders of magnitude greater than that of EDF tests, leaving the OQDA-k algorithm inappropriate for online priority assignment. This entails good, cost-effective, alternative solutions to online quasi-deadline assignment.

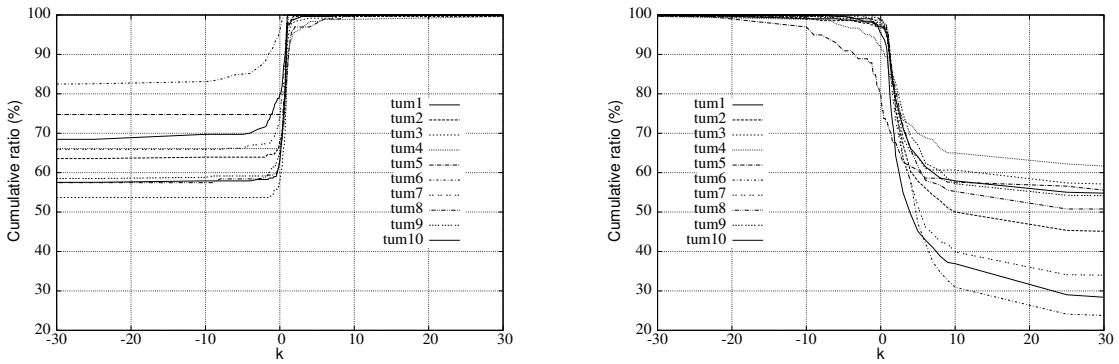
	m	EDF	I-EDF	EQDF*	I-EQDF*
Schedulability	4	11.1	26.7	30.4	37.8
(%)	8	6.5	18.3	20.7	29.1
Running	4	0.4	5.3	5.4×10^3	1.42×10^5
Time(μs)	8	0.7	23.4	2.26×10^4	1.85×10^6

Table 1. EDF and EQDF tests

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

To simply 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 $N(-\infty, \infty)$.

Figure 4 shows how schedulable values k^* are distributed over k on $m = 8$ processors. More specifically, Figure 4(a) plots $F(-\infty, k')$, representing the cumulative distribution of schedulable k values in a direction from $-\infty$ to the value of k'. On the other hand, Figure 4(b) plots $F(k', \infty)$ in the other direction from ∞ to k'. Each curve represents the cumulative distribution of one specific task utilization model (with either a bimodal or exponential parameter). It is consistently shown across those 10 different task utilization models that there is a sharp increase in a short range of k' values before and after zero, reaching at the 95th percentile or higher. This implies that we can find a schedulable value k^* with a high probability by looking at a short range of k values, instead of exploring the whole range of $[-\infty, \infty]$, if any schedulable value k^* exists.



(a) $F(-\infty, k)$ (b) $F(k, \infty)$

Figure 4. Cumulative distribution of schedulable k values on $m = 8$ processors

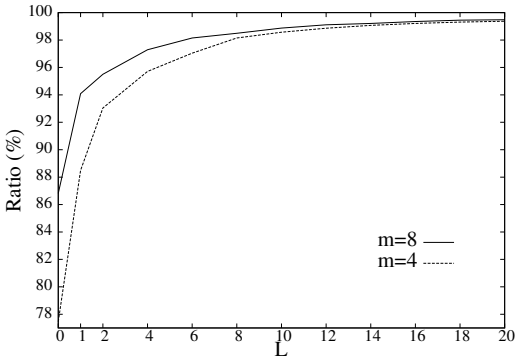


Figure 5. Ratio of schedulable task sets with $[t^*, t^* + L]$ to those with $[-\infty, \infty]$

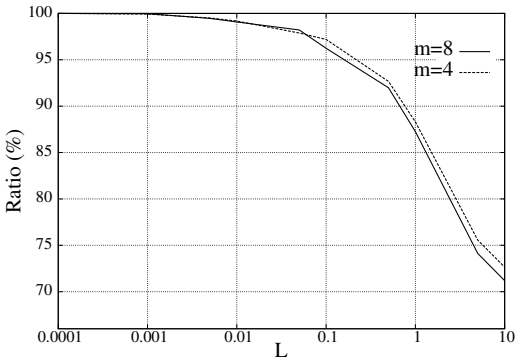


Figure 6. Ratio of the size of $S(L)$ to the size of S

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. Figure 5 plots $F(t^*, t^* + L)$ over different values of L with $m = 4$ and $m = 8$, and Table 2 shows corresponding intervals $[t^*, t^* + L]$. It is shown that even a single value of k, such as $k = 0.7$ on $m = 4$ and $k = 1$ on $m = 8$, makes a schedulable value k^* with a higher probability than 78% and 87%, respectively. Such a probability 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.

m	Interval length (L)				
	0	1	4	8	16
4	[0.7,0.7]	[0.2,1.2]	[-2.4,1.6]	[-4.9,3.1]	[-9.0,7.0]
8	[1.0, 1.0]	[0.3,1.3]	[-2.5,1.5]	[-5.3,2.7]	[-10.8,5.2]

Table 2. Intervals ($[t^*, t^* + L]$)

Table 2 show where those intervals $[t^*, t^* + L]$ are located, and this provides 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 algorithm generates 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 $|s| \geq L$. Figure 6 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 0.01. 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.

	m	Length (L) & Interval				
		0 [1,1]	4 [-2,2]	16 [-8,8]	32 [-16,16]	64 [-32,32]
Schedulability ratio (%)	4	77.4	95.1	98.2	98.7	99.2
	8	86.6	96.9	98.6	99.1	99.2
Running time ratio (10^{-3})	4	0.3	2.7	12.2	24.4	47.4
	8	0.06	0.6	4.1	5.1	15.6

Table 3. The ratio of heuristics to optimal

Optimal vs. heuristic solutions. Based on understanding of the characteristics of optimal solutions, we determine where and how densely the HQDA-k algorithm examines k values. Table 3 shows the interval that HQDA-k examines for a given value of L, and we set the sampling step to 0.1. For example, when $L = 4$, HQDA-k examines the interval $[-2, 2]$ with the step size of 0.1. This way, we can effectively reduce the search space of HQDA-k. Table 3 also shows that the ratio of heuristic solutions to optimal in terms of schedulability and running time. It is shown that when $L = 4$, the heuristic algorithm finds a solution 95% close to optimal with a shorter running time by three to four orders of magnitude. This translates into that the heuristic solution has a comparable running time with EDF analysis but produces 40%-57% better results than EDF. When $L = 64$, heuristic solutions have only less than 1% loss of optimality reducing running time by two orders of magnitude on 4 and 8 processors, respectively.

6 Conclusion

In this paper, we presented the EQDF algorithm (categorized into job-level fixed-priority (JFP) scheduling) to overcome poor performance of the existing JFP algorithm on multiprocessor platforms. EQDF assigns priority to jobs according to their quasi-deadlines ($d_i - kC_i$), 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 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 four orders of magnitude) at the expense of 1$\sim$5% optimality loss.

In this paper, the notion of quasi-deadline has been explored for job-level fixed-priority scheduling. We plan to extend this notion towards job-level dynamic-priority algorithms such as EDZL.

Acknowledgements

This work was supported in part by the IT R&D Program of MKE/KEIT [2011-KI002090], Basic Research Laboratory (BRL) Program (2009-0086964), Basic Science Research Program (2011-0005541), and NCRC (2011-0018245) through the National Research Foundation of Korea (NRF) funded by the Korea Government (MEST), and KAIST-Microsoft Research Collaboration Center.

References

[1] B. Andersson. Global static-priority preemptive multiprocessor scheduling with utilization bound 38%. In *ICPDS*, 2008.
[2] B. Andersson, K. Bletsas, and S. Baruah. Scheduling arbitrary-deadline sporadic task systems on multiprocessor. In *RTCSA*, 2008.
[3] B. Andersson and J. Jonsson. Fixed-priority preemptive multiprocessor scheduling: To partition or not to partition. In *RTCSA*, 2000.
[4] N. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical Report YCS164, Department of Computer Science, University of York, 1991.
[5] N. Audsley. On prioirty assignment in fixed priority scheduling. *Information Processing Letters*, 79(1):39–44, 2001.
[6] T. Baker. An analysis of EDF schedulability on a multiprocessor. *IEEE Transactions on Parallel Distributed Systems*, 16(8):760–768, 2005.
[7] T. P. Baker. Multiprocessor EDF and deadline monotonic schedulability analysis. In *RTSS*, 2003.
[8] T. P. Baker and M. Cirinei. A necessary and sometimes sufficient condition for the feasibility of sets of sporadic hard-deadline tasks. In *RTSS*, pages 178–190, 2006.
[9] T. P. Baker, M. Cirinei, and M. Bertogna. EDZL scheduling analysis. *Real-Time Systems*, 40:264–289, 2008.
[10] S. Baruah. Techniques for multiprocessor global schedulability analysis. In *RTSS*, 2007.
[11] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, and S. Stiller. Implementation of a speedup-optimal global EDF schedulability test. In *ECRTS*, 2009.
[12] S. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. Proportionate progress: a notion of fairness in resource allocation. *Algorithmica*, 15(6):600–625, 1996.
[13] M. Bertogna and M. Cirinei. Response-time analysis for globally scheduled symmetric multiprocessor platforms. In *RTSS*, 2007.
[14] M. Bertogna, M. Cirinei, and G. Lipari. Improved schedulability analysis of EDF on multiprocessor platforms. In *ECRTS*, 2005.
[15] M. Bertogna, M. Cirinei, and G. Lipari. New schedulability tests for real-time task sets scheduled by deadline monotonic on multiprocessors. In *ICPDS*, 2005.
[16] R. Davis and A. Burns. Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. In *RTSS*, 2009.
[17] R. I. Davis and A. Burns. FPZL schedulability analysis. In *RTAS*, 2011.
[18] M. L. Dertouzos and A. K. Mok. Multiprocessor on-line scheduling of hard-real-time tasks. *IEEE Transactions on Software Engineering*, 15:1497–1506, 1989.
[19] J. Goossens, S. Funk, and S. Baruah. Priority-driven scheduling of periodic task systems on multiprocessors. *Real-Time Systems*, 25(2–3):187–205, 2003.
[20] J. Lee, A. Easwaran, and I. Shin. LLF Schedulability Analysis on Multiprocessor Platforms. In *RTSS*, 2010.
[21] J. Lee, A. Easwaran, and I. Shin. Maximizing contention-free executions in multiprocessor scheduling. In *RTAS*, 2011.
[22] S. K. Lee. On-line multiprocessor scheduling algorithms for real-time tasks. In *IEEE Region 10's Ninth Annual International Conference*, 1994.
[23] J. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic real-time tasks. *Performance Evaluation*, 2:237–250, 1982.
[24] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.
[25] J. Liu. Real-time systems. *Prentice Hall*, 2000.
[26] M. Bertogna, M. Cirinei, and G. Lipari. Schedulability analysis of global scheduling algorithms on multiprocessor platforms. *IEEE Transactions on Parallel and Distributed Systems*, 20:553–566, 2009.