2012 IEEE 33rd Real-Time Systems Symposium

Extending Task-level to Job-level Fixed Priority Assignment and Schedulability Analysis Using Pseudo-deadlines

Hoon Sung Chwa[*], Hyoungbu Back[*], Sanjian Chen[†], Jinkyu Lee[‡]
Arvind Easwaran[§], Insik Shin[*] and Insup Lee[†]

[*]Dept. of Computer Science, KAIST, South Korea
[†]Dept. of Computer and Information Science, University of Pennsylvania, U.S.A.
[‡]Dept. of Electrical Engineering and Computer Science, The University of Michigan, U.S.A.
[§]Honeywell Inc., Minneapolis, U.S.A.
insik.shin@cs.kaist.ac.kr; lee@cis.upenn.edu

***Abstract*—In global real-time multiprocessor scheduling, a recent analysis technique for Task-level Fixed-Priority (TFP) scheduling has been shown to outperform many of the analyses for Job-level Fixed-Priority (JFP) scheduling on average. Since JFP is a generalization of TFP scheduling, and the TFP analysis technique itself has been adapted from an earlier JFP analysis, this result is counter-intuitive and in our opinion highlights the lack of good JFP scheduling techniques. Towards generalizing the superior TFP analysis to JFP scheduling, we propose the Smallest Pseudo-Deadline First (SPDF) JFP scheduling algorithm. SPDF uses a simple task-level parameter called pseudo-deadline to prioritize jobs, and hence can behave as a TFP or JFP scheduler depending on the values of the pseudo-deadlines. This natural transition from TFP to JFP scheduling has enabled us to incorporate the superior TFP analysis technique in an SPDF schedulability test. We also present a pseudo-deadline assignment algorithm for SPDF scheduling that extends the well-known Optimal Priority Assignment (OPA) algorithm for TFP scheduling. We show that our algorithm is optimal for the derived schedulability test, and also present a heuristic to overcome the computational complexity issue of the optimal algorithm. Our simulation results show that the SPDF algorithm with the new analysis significantly outperforms state-of-the-art TFP and JFP analysis.**

I. Introduction

Multi-core processing architectures are now being increasingly used in embedded systems with real-time constraints, and as a result real-time scheduling research has been steadily gaining importance. Given a set of tasks with timing requirements (i.e., deadlines), where in each task can potentially generate an infinite sequence of jobs, real-time scheduling determines the order of execution of those jobs in order to satisfy the 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.

Priority-based real-time multi-processor scheduling approaches can be broadly classified into three categories. A Task-level Fixed-Priority (TFP) scheduler assigns a fixed priority to all the jobs of each single task, a Job-level Fixed-Priority (JFP) scheduler assigns a fixed priority to each single job, and a Job-level Dynamic-Priority (JDP) scheduler assigns a priority to each single job that can dynamically change over time. It is easy to see from these definitions that JFP is a generalization of TFP scheduling, and JDP is a generalization of JFP scheduling. In the uniprocessor case several examples of optimal algorithms exist for each of these categories including DM (Deadline-Monotonic) for TFP [1], EDF (Earliest Deadline First) for JFP [2], and LLF (Least Laxity First) for JDP scheduling [3]. For multiprocessor scheduling, depending on how the tasks are mapped to cores, scheduling approaches can also be broadly classified into *partitioned* for many-to-one mappings, *global* for all-to-all mappings, and *clustered* for many-to-many mappings. In this paper, we focus on global JFP scheduling.

Many different global TFP algorithms like RM (Rate-Monotonic) [2], OPA (Optimal Priority Assignment) [4], and RM-US [5], have been proposed in the past. Several analysis techniques have also been derived for these algorithms such as those based on utilization bounds [5], response-time analysis [6], and Deadline-Analysis with Limited Carry-in (DA-LC) [7]. An interesting aspect of the DA-LC analysis is that it was first derived for JFP scheduling [8], and later specialized to TFP scheduling with a key observation of worst-case critical instant for LC. It was also shown that combining OPA with DA-LC leads to a significant improvement in the schedulability of TFP scheduling, even when compared to JFP scheduling, although JFP is a generalization of TFP [9].

JDP scheduling is probably the most widely studied among the three, mainly because all known optimal multiprocessor scheduling algorithms fall in this category. Starting with the optimal algorithm pFair that incurs many task preemptions and migrations [10], most recent work has focused on either preserving the optimality while reducing preemptions and migrations [11], [12], or foregoing optimality and reducing preemptions and migrations even further [13], [14].

This paper is dedicated to the memory of our beloved student, Late Chungdam Kim, who shared a pure passion for research.

Although TFP scheduling incurs very little preemption and migration overhead, it is limited in terms of its ability to meet deadlines. Conversely, although JDP scheduling has proven to be successful in scheduling many task sets, its applicability is limited due to the high number of preemptions and migrations. JFP scheduling seems to offer benefits from both worlds: on account of the per-job priority, it is generally able to schedule more task sets when compared to TFP scheduling, and at the same time, it does not suffer from as many preemptions and migrations as JDP scheduling because the job priorities do not change over time. Similar to the TFP and JDP cases, many different JFP algorithms like EDF [2], EDF-US [15], fp-EDF [16], and EQDF [17], have been proposed in the past. Analysis techniques for JFP algorithms have also matured over the years starting with the utilization based tests for EDF [18] and EDF-US [15]. Tests based on Deadline-Analysis (DA) [19], Response-Time Analysis (RTA) [6], and DA with Limited Carry-in (DA-LC) [8] have all been derived.

Although JFP is a generalization of TFP by definition, and many different TFP and JFP algorithms/analysis have been presented in the past, there is no clear dominance relation between any of them. On the contrary, the TFP technique of OPA with DA-LC analysis seems to outperform many JFP analysis techniques on average [9]. We believe that the reason for this rather counter-intuitive behavior is less a feature of TFP and more a lack of good JFP algorithms and analysis techniques for them. This motivated our research for a better understanding on whether, why not, and how such TFP techniques can be applied to JFP scheduling, considering the fact that TFP is a specialization of JFP.

This paper presents a JFP scheduling algorithm, called SPDF (Smallest Pseudo-Deadline First), that uses a task-level parameter called *pseudo-deadline* to prioritize jobs. To the best of our knowledge, this is the first algorithm that can be controlled to behave either as a TFP or as a JFP scheduler, depending on the values assigned to the pseudo deadlines. As a consequence of this seamless transition from TFP to JFP scheduling, we were able to extend the superior performance benefits of LC-based TFP analysis techniques to JFP schedulers like SPDF. Furthermore, we were also able to extend the well-known TFP-specific OPA algorithm to JFP scheduling, yielding an Optimal Pseudo-Deadline Assignment (OPDA) algorithm subject to some given SPDF analysis. Although we show that the OPDA algorithm is optimal for the derived DA-LC analysis, its runtime complexity in the worst-case is exponential. Therefore, as an alternative we propose a heuristic approach that combines some parametric features of OPDA and a heuristic algorithm called largest slack first (LSF). Our simulation on synthetic workloads show that our heuristic approach outperforms the state-of-the-art TFP and JFP schedulability significantly. Our approach is shown to find 5-21% and 20-100% more schedulable task sets, compared to OPA with DA-LC analysis and EDF with RTA analysis, respectively.

Contributions. The main contributions of this paper can be summarized as follows:

1) We present the first algorithm (SPDF) that generalizes TFP scheduling to JFP scheduling with a simple task-level parameter called pseudo-deadline. We show that SPDF dominates all TFP algorithms when pseudo-deadlines are appropriately determined. We also show that SPDF can easily express other well-known JFP algorithms like EDF, EDF-US and fp-EDF. (Section III)
2) We present a DA-LC analysis technique for SPDF, and this is the first JFP schedulability analysis that incorporates the superior performing TFP-tailored LC technique. (Section IV)
3) We present an Optimal Pseudo-Deadline Assignment (OPDA) algorithm for SPDF, generalizing OPA with DA-LC analysis. (Section V)
4) Given the computational intractability of OPDA, we propose a heuristic approach that also dominates OPA with DA-LC analysis. (Section VI). Our evaluation results show that our heuristic approach is quite effective in advancing the schedulability of JFP schedulers when DA-based analysis is used. (Section VII).

II. System model

This paper studies the global scheduling problem on a homogeneous multiprocessor platform with m identical processors. We consider a sporadic task model τ, in which a task $\tau_i \in \tau$ represents a potentially infinite job release sequence and is characterized by (T_i, C_i, D_i): T_i is the minimum inter-job separation, C_i is the worst-case execution time, and D_i is the relative deadline. All tasks have constrained deadlines, i.e., $(\forall i, C_i \leq D_i \leq T_i)$, and a single job cannot be executed in parallel.

The density of a task is defined as $\delta_i = C_i/D_i$, and the system density δ_{sys} is given by $\sum_{\tau_i \in \tau} \delta_i$. We let J_i^h denote the h-th job of task τ_i, and r_i^h and d_i^h denote its release time and absolute deadline, respectively, where d_i^h is given by $d_i^h = r_i^h + D_i$. The *scheduling window* of a job J_i^h is then defined as the interval $[r_i^h, d_i^h)$. Two jobs J_i^h and J_k^l are said to be *competing* with each other if their scheduling windows overlap.

In this paper, we consider a scheduling algorithm called SPDF (Smallest Pseudo-Deadline First). Each task τ_i is assigned a task-level parameter called relative *pseudo-deadline* (P_i), and the absolute pseudo-deadline p_i^h of each job J_i^h is then determined as $p_i^h = r_i^h + P_i$. We consider pseudo-deadlines as integer values. The SPDF algorithm assigns the highest priority to the job with the smallest p_i^h. It is easy to see that SPDF is a JFP scheduling algorithm and its behavior is very similar to the classic Earliest Deadline First (EDF) algorithm, except that SPDF uses job "pseudo-deadlines" instead of deadlines. Note that the pseudo-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 deadlines.

SPDF scheduling is closely related to the previous work on EQDF (Earliest Quasi-Deadline First) scheduling [17]. In EQDF scheduling, each job J_i^h is assigned a quasi-deadline $q_i^h = d_i^h - k \cdot C_i$, and jobs with earlier quasi-deadlines are given higher priorities. SPDF is a generalization of EQDF, because the quasi-deadlines of tasks are controlled by a system-wide parameter k, while the pseudo-deadlines of tasks are individually controlled by task-level parameters P_i.

In both TFP and JFP scheduling, schedulability analysis generally consists of two sub-problems: 1) determine the priority ordering of jobs and tasks, and 2) perform schedulability test to determine whether a task set with the given priority ordering is schedulable. A task set is said to be *schedulable* under a priority ordering if all job deadlines are met in the resulting schedule. Note that a schedulability test may be sufficient but not always necessary, i.e., task sets that fail the test may still be schedulable. For the SPDF algorithm, since pseudo-deadlines determine job priorities, the two fundamental problems that we address in this paper are: 1) deriving a schedulability test, and 2) assigning pseudo-deadlines to a task set such that it passes the schedulability test and hence is schedulable with SPDF scheduling.

Now we present some relations and optimality concepts that we will use in the paper. The notion of an optimal priority assignment algorithm has been defined for TFP algorithms in the past [9]. A similar notion of an Optimal Pseudo-Deadline Assignment (OPDA) algorithm can be defined for SPDF as follows.

Definition 1 (OPDA algorithm): A pseudo-deadline assignment algorithm A is "optimal" with respect to a schedulability test X and a given task model, if and only if given any task set τ that is compliant with the task model, if there exists a pseudo-deadline assignment such that τ passes test X, then τ can also pass test X using the pseudo-deadline assignment of algorithm A.

The performance of different schedulability tests may be compared using the dominance relation which has been defined in literature [9].

Definition 2 (Dominance [9]): Schedulability test X dominates schedulability test Y, if any given priority ordered task set τ that passes test Y also passes test X, and there exists at least one priority ordered task set that passes test X but fails test Y.

Similar to the above comparison, the performance of different scheduling algorithms may also be compared without referring to specific schedulability tests, using the "better than" relation defined below.

Definition 3 (Better than relation): A scheduling algorithm A is *better than* scheduling algorithm B, if any task set that is schedulable by B is also schedulable by A, and there exist at least one task set that is schedulable by A but not schedulable by B.

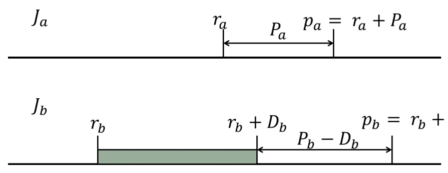


Figure 1. When $P_a \le P_b - D_b$, J_a always preempts J_b.

III. SPDF Scheduling Algorithm

As described previously, the SPDF algorithm assigns priorities to jobs according to their pseudo-deadlines. In this section, we first classify inter-task priority relations and then show that SPDF is better than any existing TFP and JFP algorithms.

Inter-task priority dominance relation. For a task pair (τ_i, τ_k), there are two possible priority dominance relations.

- **Type A.** One task (assuming it is τ_i without loss of generality) is said to be *strictly higher* than the other task (τ_k), denoted by $\tau_i \succ \tau_k$, if every job of τ_i has a higher priority than its competing jobs of τ_k; Lemma 1 gives a necessary and sufficient condition of when this may happen. We express a task pair (τ_i, τ_k) belongs to *Type A1* and *Type A2* dominance relations if $\tau_i \succ \tau_k$ and $\tau_i \prec \tau_k$, respectively.
- **Type B.** Two tasks τ_i and τ_k are said to be *mutual*, denoted by $\tau_i \prec\succ \tau_k$, if some jobs of τ_i have higher priorities than their competing jobs of τ_k but the other jobs of τ_i have lower priorities than their competing jobs of τ_k.

Lemma 1: Any two tasks τ_a and τ_b will have Type A1 relation in the SPDF schedule if and only if $P_a \le P_b - D_b$.

Proof: We show that SPDF always assigns higher priority to jobs of τ_a, if and only if $P_a \le P_b - D_b$.

- "$\leftarrow$" sufficient condition:
  Let J_x denote any job of τ_x. Suppose J_a is released within the scheduling window of J_b, i.e., $r_b \le r_a < r_b + D_b$. The pseudo-deadline of J_a is $p_a = r_a + P_a < (r_b + D_b) + (P_b - D_b) = r_b + P_b = p_b$. Such scenario is depicted in Figure 1[1]. Since $p_a < p_b$, under SPDF schedule, J_a preempts J_b. Note that J_a and J_b are arbitrary jobs, i.e., we show that jobs of τ_a always preempts jobs of τ_b.
  Next we show that no job of τ_b can preempt any job of τ_a. If J_b is released within the scheduling window of J_a, then $r_b \ge r_a$. The pseudo-deadline of J_b is $p_b = r_b + P_b \ge r_a + (P_a + D_b) > r_a + P_a = p_a$. Since $p_b > p_a$, under SPDF, J_b cannot preempt J_a.
  Therefore, if $P_a \le P_b - D_b$, there is Type A1 relation between τ_a and τ_b.

[1]For illustration purpose, P_a and P_b are positive in Figure 1, but it is easy to see the proof applies to arbitrary P_a and P_b.

- "$\rightarrow$" necessary condition:
  Negate the condition and suppose $P_a > P_b - D_b$. We show a case in which J_a competes with J_b but J_a is assigned lower priority by SPDF. Suppose J_a is released during $(\max(r_b, r_b + P_b - P_a), r_b + D_b]$ (the $\max$ operation is to count for the possibility that $P_b < P_a$ and here we want the arrival time of J_a to be later than r_b), and note that such interval is non-empty because $P_b - P_a < D_b$. We have $p_a = r_a + P_a > (r_b + P_b - P_a) + P_a = r_b + P_b = p_b$. Therefore J_a has lower priority than J_b under SPDF. ■

Lemma 1 shows that for any task set τ, SPDF will be able to maintain the priority order $\tau_a \succ \tau_b$, if $P_a \leq P_b - D_b$.

Example 3.1: Let $\tau = \{\tau_1 = \tau_2 = \tau_3 = (3, 2, 3), \tau_4 = (4, 2, 4), \tau_5 = \tau_6 = \tau_7 = (3, 0.1, 3)\}$ and $m = 3$. τ is not schedulable under EDF [2], any TFP algorithm, fp-EDF [16] or EDF-US [15], but it is schedulable under SPDF by setting pseudo-deadlines in the following way: $P_1 = P_2 = P_3 = 3, P_4 = 4, P_5 = P_6 = P_7 = 1000$. Essentially, SPDF can schedule τ by assigning lowest priorities to the last three tasks such that each of the other four tasks is strictly higher than the last three tasks and the remaining four tasks are scheduled by EDF.

Theorem 1: SPDF is better than all TFP algorithms, EDF [2], EDF-US [15], and fp-EDF [16].

Proof: By Lemma 1, any task set that is schedulable by any TFP algorithm is also schedulable by SPDF, because SPDF can generate exactly the same priority ordering.

SPDF schedules task sets in exactly the same way as EDF does when $\forall i, P_i = D_i$. Therefore any task set that is schedulable by EDF is schedulable by SPDF.

Given a task set τ, assuming that the tasks sorted non-increasingly by their utilizations, both EDF-US and fp-EDF assign highest fixed priorities to a subset of "heavy" tasks τ_H and run EDF on the remaining tasks $\tau_L = \tau \backslash \tau_H$. Then SPDF can generate exactly the same schedule by setting $\forall \tau_i \in \tau_L, P_i = D_i$ and enforcing $\forall \tau_a, \tau_b \in \tau_H(\tau_a \succ \tau_b), P_a \leq P_b - D_b$ as well as that $\max_{\tau_x \in \tau_H} P_x \leq \min_{\tau_y \in \tau_L}(P_y - D_y)$. Therefore any task set that is schedulable by EDF-US or fp-EDF will be schedulable by SPDF.

Finally, Example 3.1 gives a task set that is schedulable by SPDF but not schedulable by any TFP algorithm, EDF [2], EDF-US [15], or fp-EDF [16]. ■

IV. SPDF Schedulability Analysis

This section derives a schedulability condition for the proposed SPDF scheduling algorithm. We first recapitulate an interference-based analysis, which has served as a basis for the schedulability conditions of various algorithms [20], [19], [7], [21], [17], [22], [13], including EDF and TFP algorithms. We also explain an existing technique, called DA-LC, which can substantially improve the performance of the interfere-based analysis. We then discuss how to apply the interference-based analysis to the SPDF algorithm.

Recapitulation of Interference-based Analysis. The *total interference* on a task τ_k in an interval $[a, b)$ (denoted by $\overline{I}_k(a, b)$) is defined by the cumulative length of all intervals in which τ_k is ready to execute but is not executing due to higher priority jobs of other tasks. We also define the *interference* of a task τ_i on a task τ_k in an interval $[a, b)$ (denoted by $\overline{I}_{i,k}(a, b)$) is defined as the cumulative length of all intervals in which τ_k is ready to execute but it is not executing since τ_i is executing instead. Since a task cannot be scheduled only when m other tasks execute, a relation between $\overline{I}_k(a, b)$ and $\overline{I}_{i,k}(a, b)$ has been derived in Lemma 3 in [19] as follows:

$$\overline{I}_k(a, b) = \frac{\sum_{i \neq k} \overline{I}_{i,k}(a, b)}{m}. \quad (1)$$

Let J_k^* denote the job that receives the maximum total interference among jobs of τ_k, and then the worst-case total interference on the job of τ_k (denoted by I_k^*) can be expressed

$$I_k^* \triangleq \max_h(\overline{I}_k(r_k^h, d_k^h)) = \overline{I}_k(r_k^*, d_k^*). \quad (2)$$

For notational convenience, we also define

$$I_{i,k}^* \triangleq \overline{I}_{i,k}(r_k^*, d_k^*). \quad (3)$$

Using the above definitions, the studies [19], [21] developed the exact schedulability condition of global multiprocessor scheduling algorithms as follows:

Lemma 2 (from [19], [21]): A task set τ is schedulable on a multiprocessor composed by m identical processors if and only if the following condition holds for every task τ_k:

$$\sum_{\tau_i \in \tau \backslash \{\tau_k\}} \min(I_{i,k}^*, D_k - C_k + 1) < m \cdot (D_k - C_k + 1). \quad (4)$$

Since it is generally intractable to compute exact $I_{i,k}^*$ under a given scheduling algorithm, existing approaches [20], [19], [7], [21], [17], [22], [13] have derived upper-bounds on $I_{i,k}^*$ under their target algorithms, resulting in sufficient schedulability tests.

DA-LC Analysis Technique. The DA-LC (Deadline Analysis with Limited Carry-in) analysis technique has been introduced to calculate the maximum interference of *carry-in* jobs. A job is said to be a carry-in job of a given interval when the job is released before the interval, but has a deadline within the interval. The DA-LC technique was initially developed for EDF [8], based on the concept of *busy interval*, which refers to the maximum continuous interval during which all processors are occupied. By definition, there exist at most $m - 1$ carry-in jobs into a busy interval. The DA-LC technique developed for EDF requires an investigation into all possible busy intervals of arbitrary length [8]. It was later specialized to TFP

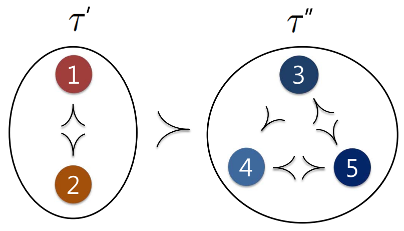


Figure 2. Example of $\tau' \succ \tau''$: τ_1 and τ_2 are strictly higher than τ_3, τ_4, and τ_5, respectively.

scheduling algorithms with a key observation on the worst-case critical instant [7]. The maximum interference on a job J_k comes with at most $m-1$ higher-priority carry-in jobs into the scheduling window of J_k. This is because, in the case of more than m carry-in jobs, J_k would have only larger interference when it is released earlier until it has at most $m-1$ carry-in jobs. It is worth noting that releasing J_k earlier does not affect the execution of all other higher-priority jobs under TFP scheduling. On the other hand, releasing J_k earlier consequently makes its deadline earlier, and this can potentially affect priority ordering between J_k and other tasks under EDF scheduling. This way, it is possible to consider only at most $m-1$ carry-in higher-priority tasks' jobs in a job's scheduling window, resulting in a tighter upper-bound on the maximum interference on the job. Without such a condition for the critical instance, it is required either to consider that all higher-priority jobs can be carry-in in the scheduling window, or to investigate all possible busy intervals of arbitrary length before the scheduling window.

Interference-based Analysis under SPDF. We discuss how to derive an upper-bound on $I^*_{i,k}$ under SPDF scheduling. As described in the previous section, such a dominance relation falls into three types: A1, A2, and B. From the standpoint of τ_k, each task τ_i ($\neq \tau_k$) belongs to one of the three sets, $\tau_k^{(A1)}$, $\tau_k^{(A2)}$ and $\tau_k^{(B)}$, according to its relation with τ_k. Specifically, τ_i belongs $\tau_k^{(A1)}$ if $\tau_i \succ \tau_k$ (Type A1), $\tau_k^{(A2)}$ if $\tau_i \prec \tau_k$ (Type A2), or $\tau_k^{(B)}$ otherwise (i.e., $\tau_i \prec\succ \tau_k$, Type B).

Type A1. We first consider the case of $\tau_i \succ \tau_k$ (Type A1) in deriving an upper-bound on $I^*_{i,k}$. In this case, a tighter bound can be derived with the use of the DA-LC analysis. Observing that the TFP-specific DA-LC technique relies on the inter-task Type A relation, we explore a possibility of employing the technique even under SPDF scheduling, in particular, to some tasks under Type A relation. Towards this, let us consider a task set τ' as strictly higher than another task set τ'' (denoted as $\tau' \succ \tau''$) if it holds that, for all tasks $\tau_i \in \tau'$ and $\tau_j \in \tau''$, $\tau_i \succ \tau_j$ (see Figure 2). We then partition τ into two disjoint subsets, $\tau_k^{(A^*)}$ and $\tau_k^{(B^*)}$, from the standpoint of τ_k such that $\tau_k^{(B^*)}$ must include τ_k and $\tau_k^{(A^*)} \succ \tau_k^{(B^*)}$. It holds by definition that $\tau_k^{(A^*)} \subseteq \tau_k^{(A1)}$, and we here consider having the largest possible $\tau_k^{(A^*)}$ for better schedulability.

Releasing any task $\tau_b \in \tau_k^{(B^*)}$ earlier can change priority ordering between the tasks only belonging to $\tau_k^{(B^*)}$ but brings no impact on the priority dominance relation between $\tau_k^{(A^*)}$ and $\tau_k^{(B^*)}$. The following lemma shows that the TFP-specific DA-LC technique is then applicable to $\tau_k^{(A^*)}$.

Lemma 3: Under SPDF, the amount of execution of jobs of tasks in $\tau_k^{(A^*)}$ in an interval of length l is maximized when there are at most $m-1$ carry-in jobs of tasks in $\tau_k^{(A^*)}$.

Proof: By the definition of $\tau_k^{(A^*)}$, tasks in $\tau_k^{(A^*)}$ have higher priority than any other task in $\tau_k^{(B^*)}$ $(= \tau \setminus \tau_k^{(A^*)})$. Therefore, the execution of jobs of tasks in $\tau_k^{(A^*)}$ is not affected by any other job of tasks in $\tau_k^{(B^*)}$.

Suppose that there are at least m carry-in jobs of tasks in $\tau_k^{(A^*)}$ in $[t, t+l)$. Then, exactly m jobs of tasks in $\tau_k^{(A^*)}$ are executed in $[t-1, t)$ regardless of jobs of tasks in $\tau_k^{(B^*)}$. Since at most m jobs can be executed in $[t+l-1, t+l)$, the amount of execution of jobs of tasks in $\tau_k^{(A^*)}$ will increase or stay if we shift the interval of interest as $[t-1, t+l-1)$. This shift will be repeated until there are at most $m-1$ carry-in jobs of tasks in $\tau_k^{(A^*)}$, and then the amount of execution of jobs of tasks in $\tau_k^{(A^*)}$ in the final interval upper-bounds that in the original interval. ■

In order to apply the TFP-specific DA-LC technique, we need to consider two sub-cases further for calculation of $I^*_{i,k}$: whether a carry-in job of τ_i exists or not. If $\tau_i \in \tau_k^{(A1)}$ has its carry-in job in an interval between the release time and deadline of a job of τ_k, $I^*_{i,k}$ is upper-bounded by the maximum execution of jobs of τ_i in an interval of length D_k (denoted by $W_i^{\mathsf{CI}}(D_k)$) [6], where

$$W_i^{\mathsf{CI}}(l) = \left\lfloor \frac{l + D_i - C_i}{T_i} \right\rfloor \cdot C_i + \min\big(C_i, (l + D_i - C_i) \bmod T_i\big). \tag{5}$$

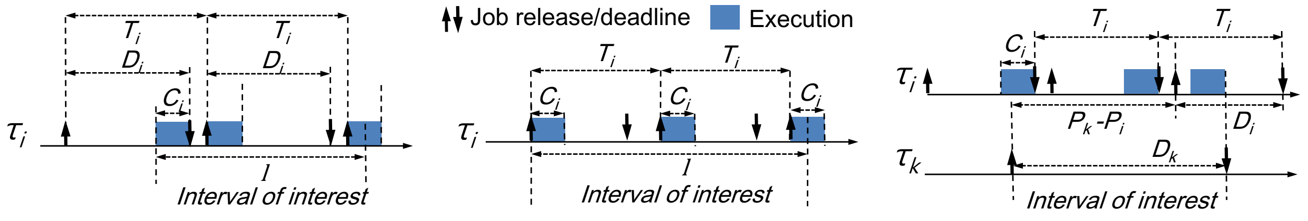
Such a pattern of the maximum execution occurs when the first job of τ_i is executed as late as possible, and thereafter other jobs are scheduled immediately as shown in Fig. 3(a).

On the other hand, if $\tau_i \in \tau_k^{(A1)}$ has no carry-in job in the interval, $I^*_{i,k}$ is upper-bounded by $W_i^{\mathsf{NC}}(D_k)$, the maximum execution of non-carry-in jobs of τ_i in an interval of length D_k [8], where

$$W_i^{\mathsf{NC}}(l) = \left\lfloor \frac{l}{T_i} \right\rfloor \cdot C_i + \min\big(C_i, l \bmod T_i\big), \tag{6}$$

in which all jobs are released and scheduled as soon as possible as shown in Fig. 3(b). Note that $W_i^{\mathsf{CI}}(l) \geq W_i^{\mathsf{NC}}(l)$ holds for any $l > 0$.

Type A2. We then consider another case where $\tau_i \prec \tau_k$ (Type A2). In this case, τ_k has no interference from τ_i, and thereby we have $I^*_{i,k} = 0$.



(a) $W_i^{\mathsf{CI}}(l)$ (b) $W_i^{\mathsf{NC}}(l)$ (c) $W_i^{\mathsf{NC}}(D_i+P_k-P_i)$

Figure 3. The worst-case patterns of interference for different cases

Type B. Now, we derive an upper-bound of $I^*_{i,k}$ when τ_i belongs to $\tau_k^{(B)}$. For J_i^g of $\tau_i \in \tau_k^{(B)}$ to interfere with (have higher-priority than) J_k^h, $r_i^g + P_i \le r_k^h + P_k$ should be satisfied, implying $r_i^g - r_k^h \le P_k - P_i$. Therefore, the amount of higher-priority execution of jobs of τ_i than a job of τ_k is maximized when the pseudo-deadline of a job of τ_i is the same as that of the job of τ_k, i.e., when the release times of a job of τ_i and the job of τ_k are aligned apart from $P_k - P_i$ as shown in Figure 3(c). Then, $I^*_{i,k}$ is upper-bounded by $W_i^{\mathsf{NC}}(D_i+P_k-P_i)$. Note that this upper-bound is pessimistic when $P_k - P_i + C_i \ge D_k$ in that the upper-bound is larger than $W_i^{\mathsf{CI}}(D_k)$, which is an upper-bound of the execution of jobs of τ_i in an interval of length D_k in any case. Therefore, we use the minimum of the two upper-bounds for the upper-bound on $I^*_{i,k}$.

Considering we use Eq. (4) for the schedulability test of SPDF, $\min\left(I^*_{i,k}, D_k - C_k + 1\right)$ under SPDF is upper-bounded as follows:

$$I_{i,k}^{\mathrm{SPDF}} \triangleq \min\left(I^*_{i,k}, D_k - C_k + 1\right) \text{ under SPDF} \le \begin{cases} I_{i,k}^{(A2)} \triangleq 0, \text{ if } P_k - P_i \le -D_i \text{ (i.e., Type A2)}, \\ I_{i,k}^{(A1)} \triangleq \min\left(W_i^{\mathsf{CI}}(D_k), D_k - C_k + 1\right), \\ \qquad \text{if } P_k - P_i \ge D_k \text{ (i.e., Type A1) \& with carry-in}, \\ \hat{I}_{i,k}^{(A1)} \triangleq \min\left(W_i^{\mathsf{NC}}(D_k), D_k - C_k + 1\right), \\ \qquad \text{if } P_k - P_i \ge D_k \text{ (i.e., Type A1) \& no carry-in}, \\ I_{i,k}^{(B)} \triangleq \min\left(W_i^{\mathsf{NC}}(D_i + P_k - P_i), W_i^{\mathsf{CI}}(D_k), D_k - C_k + 1\right), \\ \qquad \text{if } -D_i < P_k - P_i < D_k \text{ (i.e., Type B).} \end{cases} \tag{7}$$

Then, since $I_{i,k}^{(A1)} \ge \hat{I}_{i,k}^{(A1)}$ holds, a safe schedulability test of SPDF can be derived with the use of $I_{i,k}^{(A1)}$ as an upper-bound on $\min\left(I^*_{i,k}, D_k - C_k + 1\right)$ for $\tau_i \in \tau_k^{(A1)}$. Finally, employing Lemma 3 with Eq. (7), we derive a schedulability test of SPDF in the following theorem.

Theorem 2: A task set τ is schedulable under SPDF with given priority assignment $\{P_i\}_{\tau_i \in \tau}$ on a multiprocessor composed of m identical processors if for each task τ_k the following condition holds:

$$\sum_{\tau_i \in \tau \setminus \{\tau_k\}} I_{i,k}^{\mathrm{SPDF}} < m \cdot (D_k - C_k + 1), \tag{8}$$

where the LHS of Eq. (8) is upper-bounded by

$$\sum_{\tau_i \in \tau_k^{(A^*)}} \hat{I}_{i,k}^{(A1)} + \sum_{\text{m-1 largest } \tau_i \in \tau_k^{(A^*)}} \left(I_{i,k}^{(A1)} - \hat{I}_{i,k}^{(A1)}\right) + \sum_{\tau_i \in \tau_k^{(A1)} \setminus \tau_k^{(A^*)}} I_{i,k}^{(A1)} + \sum_{\tau_i \in \tau_k^{(B)}} I_{i,k}^{(B)}. \tag{9}$$

Proof: By Lemma 2, τ is schedulable under SPDF if Eq. (8) holds. We now prove that the LHS of Eq. (8) is upper-bounded by Eq. (9).

As shown in Eq. (7), $I_{i,k}^{\mathrm{SPDF}}$ when τ_i belongs to $\tau_k^{(A1)}$, $\tau_k^{(A2)}$ and $\tau_k^{(B)}$ is upper-bounded by $I_{i,k}^{(A1)}$, $I_{i,k}^{(A2)}$ $(= 0)$ and $I_{i,k}^{(B)}$, respectively. Considering any task in $\tau_k^{(A1)} \setminus \tau_k^{(A^*)}$ belongs to $\tau_k^{(A1)}$, the following inequality holds.

$$\sum_{\tau_i \in \tau_k^{(A1)} \setminus \tau_k^{(A^*)}} I_{i,k}^{\mathrm{SPDF}} + \sum_{\tau_i \in \tau_k^{(A2)}} I_{i,k}^{\mathrm{SPDF}} + \sum_{\tau_i \in \tau_k^{(B)}} I_{i,k}^{\mathrm{SPDF}} \le \sum_{\tau_i \in \tau_k^{(A1)} \setminus \tau_k^{(A^*)}} I_{i,k}^{(A1)} + 0 + \sum_{\tau_i \in \tau_k^{(B)}} I_{i,k}^{(B)} \tag{10}$$

Since $\tau \setminus \{\tau_k\} = \tau_k^{(A^*)} \cup \left(\tau_k^{(A1)} \setminus \tau_k^{(A^*)}\right) \cup \tau_k^{(A2)} \cup \tau_k^{(B)}$, the remaining step is to prove that $\sum_{\tau_i \in \tau_k^{(A1)} \setminus \tau_k^{(A^*)}} I_{i,k}^{\mathrm{SPDF}}$ is upper-bounded by the first two terms in Eq. (9).

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 τ_k in an interval is upper-bounded by the amount of execution of jobs of τ_i in the interval. Therefore, the maximum amount of interference of τ_i on τ_k among all intervals of length D_k is also upper-bounded by the maximum amount of execution of τ_i among all intervals of length D_k. Using this relationship with incorporating Lemma 3, we conclude that $\sum_{\tau_k^{(A^*)}} I_{i,k}^{\mathrm{SPDF}}$ is upper-bounded by the amount of execution of jobs of tasks in $\tau_k^{(A^*)}$ in an interval of length D_k when there are at most $m-1$ carry-in jobs of tasks in $\tau_k^{(A^*)}$.

Since we do not know which tasks in $\tau_k^{(A^*)}$ have their carry-in jobs, we choose $m-1$ tasks in $\tau_k^{(A^*)}$, which have the $m-1$ largest difference between $I_{i,k}^{(A1)}$ and $\hat{I}_{i,k}^{(A1)}$. Then, we can safely upper-bound $\sum_{\tau_i \in \tau_k^{(A1)} \setminus \tau_k^{(A^*)}} I_{i,k}^{\mathrm{SPDF}}$ by the first two terms in Eq. (9), regardless of which tasks have carry-in jobs. ■

Note that it requires $O(n \cdot \log(n))$ to calculate Eq. (9) for a given τ_k due to sorting $I_{i,k}^{(A1)} - \hat{I}_{i,k}^{(A1)}$ terms, where n is the number of tasks in τ. Therefore, the SPDF schedulability test in Theorem 2 requires $O(n^2 \cdot \log(n))$.

The following lemma shows the dominance relation between the SPDF schedulability test and other existing tests.

Lemma 4: The SPDF schedulability analysis in Theorem 2 *dominates* the deadline-based EDF schedulability analysis (EDF-DA) in [19], [21] and the deadline-based TFP schedulability analysis with the so-called limited carry-in technique (TFP-DA-LC) in [9].

Proof: The proof is straightforward. If we set P_i to D_i for every $\tau_i \in \tau$, SPDF is the same as EDF and our SPDF schedulability analysis is equivalent to EDF-DA. Also, if we set $P_i, \forall \tau_i \in \tau$ according to Lemma 1, SPDF is the same as TFP with the corresponding priority assignment and the SPDF schedulability analysis is equivalent to TFP-DA-LC. ■

Note that Example 3.1 is deemed schedulable under SPDF scheduling according to Theorem 2, but by neither EDF-DA nor TFP-DA-LC with every possible priority assignment.

V. Optimal Pseudo-Deadline Assignment

As mentioned previously, this paper considers the *pseudo-deadline assignment* problem that, given a task set τ, determines the pseudo-deadline P_i of every task $\tau_i \in \tau$ such that the task set is deemed schedulable according to the SPDF schedulability test given in Theorem 2. In this section, we first discuss the intuition behind the OPA algorithm for TFP scheduling and then extend such intuition towards JFP scheduling, presenting an optimal pseudo-deadline assignment algorithm.

TFP/OPA. The OPA algorithm [4] aims at assigning a priority to each individual task through iterative priority assignment such that an entire task set τ is deemed schedulable by some given OPA-compatible[2] schedulability test X under TFP scheduling.

In the k-th iteration step, the task set τ is divided into two disjoint subsets: $A(k)$ and $R(k)$, where

$A(k)$	denotes a subset of tasks whose priorities have been assigned before the k-th step, and
$R(k)$	denotes a subset of remaining tasks whose priorities must be assigned from the k-th step onwards.

A task τ_e is said to be *TFP-eligible*[3] in the k-th step if τ_e is deemed schedulable by test X under the assumption that τ_e is assigned a priority strictly higher than all the assigned tasks $\tau_a \in A(k)$ but strictly lower than all the remaining tasks $\tau_r \in R(k)$. During the k-th step, OPA then seeks to select one of the eligible tasks for priority assignment. For concise presentation, we introduce additional notations as follows.

$E(k)$	denotes a subset of tasks that are eligible in the k-th step, and
$S(k)$	denotes a subset of eligible tasks that is selected for priority assignment in the k-th step; $S(k) \subseteq E(k)$[4].

Let us discuss the key intuition behind how TFP/OPA works. First, one of the most important properties is that the algorithm builds a solution incrementally without backtracking. Once a task τ_s is selected in an iteration step k, the task has no effect on priority assignment in the next iteration steps. This is because the task τ_s is assigned a priority strictly lower than all the remaining tasks $\tau_r \in R(k+1)$, imposing no interference on them under Type A2 relation.

Second, suppose there exists only one eligible task τ_e in step k. Then, OPA must find it through an exhaustive search of all the remaining tasks $R(k)$, which takes linear time.

Third, suppose there are multiple eligible tasks in the k-th step. Then, it does not matter which eligible task is selected by OPA in the k-th step, because all the other eligible tasks will remain eligible in the next steps and will be eventually selected for priority assignment in a later step. This follows from the fact that interference under Type A1 relation for a task is only smaller when it is selected in a later iteration step (that is, assigned a higher priority). This way, OPA is optimal for finding a priority assignment for a given task set with respect to some given test X under TFP scheduling.

Applicability of OPA to JFP case. Let us discuss the applicability of the OPA algorithm to the JFP category. One of the key differences between TFP and JFP scheduling is that Type B inter-task relation can hold only under JFP scheduling. Targeting TFP scheduling, OPA is not designed to explore such Type B relation in the process of priority assignment, and this is a critical factor in extending OPA towards JFP scheduling.

To illustrate this, let us consider an example. Suppose there is no TFP-eligible task in the k-th iteration step. That is, we assume that for each remaining tasks $\tau_j \in R(k)$, there is no priority assignment to make τ_j schedulable by test X under the assumption that τ_j has only Type A1 relation with all the other remaining tasks. This gives

$$\forall \tau_j \in R(k), \sum_{\tau_r \in R(k), r \neq j} I_{r,j}^{(A1)} > m(D_j - C_j + 1), \qquad (11)$$

In order to emphasize the benefit of Type B inter-task relation over Type A, we make a further assumption in this example that there exist a couple of tasks, τ_p and τ_q, such that they can be deemed schedulable by the same test X when they have Type B relation with each other. This situation can be specified as follows.

[2]Schedulability tests are *OPA-compatible* if for any given task τ_i, its schedulability is insensitive to relative ordering of its higher (and lower) priority tasks and its schedulability is monotonic to its priority (i.e., if it is schedulable (or unschedulable) at a certain priority, then it remains schedulable (or unschedulable) at a higher (or lower) priority.).

[3]We may use only "eligible" omitting "TFP-" (and "JFP-" later) for simplicity when no ambiguity arises.

[4]We note that TFP/OPA always selects a single task for $S(k)$.

$$\sum_{\tau_r \in R(k), r\neq p, r\neq q} I_{r,p}^{(A1)} + I_{q,p}^{(B)} \leq m(D_p - C_p + 1), \text{ and}$$
$$\sum_{\tau_r \in R(k), r\neq p, r\neq q} I_{r,q}^{(A1)} + I_{p,q}^{(B)} \leq m(D_q - C_q + 1). \quad (12)$$

In the above example, there exists a possible scenario to assign job-level priorities to the two tasks τ_p and τ_q that deems them schedulable. However, OPA cannot find such a scenario since it seeks to find a single task in every iteration step. This motivates the design of new priority assignment algorithms for JFP scheduling.

Optimal Pseudo-Deadline Assignment. We now present the Optimal Pseudo-Deadline Assignment (OPDA) algorithm, generalizing the TFP/OPA algorithm towards JFP scheduling. Following the key intuition behind TFP/OPA, our OPDA algorithm (described in Algorithms 1 and 2) also performs pseudo-deadline assignment iteratively. In particular, it partitions the task set τ into two disjoint sets, $A(k)$ and $R(k)$, in each iteration step k, preserving Type A inter-task relation between the two disjoint sets. As in TFP/OPA, enforcing Type A relation between $A(k)$ and $R(k)$ allows OPDA to construct a solution incrementally as well, separating individual iteration steps. On the other hand, the main differences between TFP/OPA and OPDA lie in a couple of factors, including how to enforce Type A relation between $A(k)$ and $R(k)$ and how many tasks can be selected in $S(k)$ for pseudo-deadline assignment in the k-th step.

As previously mentioned, implementing Type A relation between $A(k)$ and $R(k)$ is key to establishing an important property of incremental priority (and pseudo-deadline) assignment. In the TFP case, such Type A relation between those two subsets is automatically implemented simply by assigning different priorities to different tasks. However, this is no longer valid in the JFP case, requiring a different strategy for the separation between them. Under SPDF scheduling, we note that if two tasks satisfy the condition derived in Lemma 1, those tasks have Type A relation. Thereby, we can enforce Type A relation between $A(k)$ and $R(k)$ by assigning relative pseudo-deadlines such that both all tasks $\tau_a \in A(k)$ and all tasks $\tau_r \in R(k)$ satisfy $P_a \leq P_r - D_r$.

In addition, OPDA is designed to explore Type B relation in the process of pseudo-deadline assignment. Unlike TFP/OPA that always seeks to assign a priority to a single task in each iterative step, OPDA has flexibility in finding different numbers of tasks for determining $S(k)$ in a single step and enforces Type B relation for the tasks in the subset $S(k)$.

As described in Algorithm 1, OPDA iteratively finds a subset $S(k)$ in each step k until there is no more remaining task (Lines 2-10 in Algorithm 1). For each step k, it invokes a function FIND-SUBSET(k,i) to find the subset $S(k)$ of the smallest possible size i in the k-th step (Line 4 in Algorithm 1). As described in Algorithm 2, FIND-SUBSET(k,i) basically considers all combinations of i tasks from all the remaining tasks $R(k)$ (Line 2 in Algorithm 2). For each combination of i tasks, it explores any possibility that such i tasks are deemed schedulable according to Theorem 2

Algorithm 1 Optimal Pseudo-Deadline Assignment

```
Require: $k \leftarrow 0$, $Z^H(1) \leftarrow 0$, $R(1) \leftarrow \tau$
 1: repeat
 2:   $k \leftarrow k+1$
 3:   for each $i$ in $\{1, \cdots, |R(k)|\}$ do
 4:     if FIND-SUBSET(k,i) = success then
 5:       break (continue outer loop)
 6:     end if
 7:   end for
 8:   return unschedulable
 9: until $R(k)$ is empty
10: return schedulable
```

Algorithm 2 FIND-SUBSET(k, i)

```
 1: $\mathcal{S}_k \leftarrow$ a set of all $i$-combinations of $R(k)$
 2: for each combination $S(k)$ in $\mathcal{S}_k$ do
 3:   $maxD(k) \leftarrow \max\{D_s\}$ for all $\tau_s \in \mathcal{S}(k)$
 4:   $sumD(k) \leftarrow \sum D_s$ for all $\tau_s \in \mathcal{S}(k)$
 5:   $Z^L(k+1) \leftarrow Z^H(k) + sumD(k)$
 6:   $Z^H(k+1) \leftarrow Z^L(k+1) + maxD(k)$
 7:   $\mathcal{R} \leftarrow R(k) \setminus S(k)$
 8:   for each task $\tau_r \in \mathcal{R}$ do
 9:     $P_r \leftarrow Z^H(k+1)$
10:   end for
11:   $\mathcal{P}_k \leftarrow$ a set of all $i$-permutations with repetition from a set
      $\{Z^H(k), Z^H(k)-1, \cdots, Z^L(k+1)+1, Z^L(k+1)\}$
12:   for each element $< p_1, p_2, \cdots, p_k >$ in $\mathcal{P}_k$ do
13:     $j \leftarrow 1$
14:     for each task $\tau_s \in S(k)$ do
15:       $P_s \leftarrow p_j$
16:       $j \leftarrow j+1$
17:     end for
18:     if an entire task set $\tau$ is deemed schedulable according
        to schedulability test $X$ then
19:       $R(k+1) \leftarrow R(k) \setminus S(k)$
20:       return success
21:     end if
22:   end for
23: end for
24: return fail
```

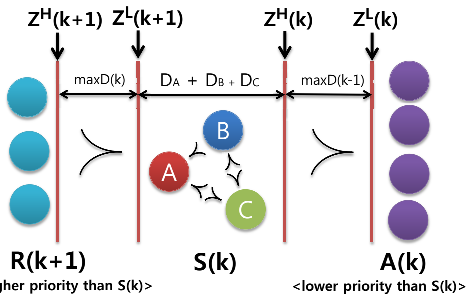


Figure 4. Pseudo-deadline buffer zone in iteration step k

under the assumption that they all have Type A1 relation with each task $\tau_a \in A(k)$, Type A or B relation with each other, and Type A2 relation with all the other remaining tasks $\tau_r \in R(k)$.

To enforce Type A relation between $A(k)$ and $R(k)$, OPDA employs a *pseudo-deadline buffer zone* $[Z^H(k), Z^L(k)]$ between $A(k)$ and $R(k)$ such that no task is assigned a pseudo-deadline inside the buffer zone[5] (see Figure 4). (Lines 3-6 in Algorithm 2):

$$Z^L(k) - Z^H(k) = \max_{\tau_r \in A(k)} D_r \overset{\text{def.}}{=} maxD(k)$$

$$Z^H(k) - Z^L(k+1) = \sum_{\tau_r \in R(k)} D_r \overset{\text{def.}}{=} sumD(k)$$

Following the principle behind Type A relation described in Lemma 1, pseudo-deadlines are assigned such that all the assigned tasks $\tau_a \in A(k)$ have pseudo-deadlines lower (i.e., numerically greater) than $Z^L(k)$ and all the remaining tasks $\tau_r \in R(k+1)$ will be assigned pseudo-deadlines higher (i.e., numerically smaller) than $Z^H(k+1)$ (Line 8-9 in Algorithm 2). We note that the buffer zone size of $maxD(k)$ is large enough to enforce the separation of $A(k)$ and $R(k)$ under Type A relation. Then, OPDA explores all the possible pseudo-deadline assignment of all the eligible tasks $\tau_s \in S(k)$ within $[Z^L(k+1), Z^H(k)]$ in an exhaustive manner (Lines 11-22 in Algorithm 2). It is worth noting that $sumD(k)$ is large enough to explore all the possible combinations of pseudo-deadlines for finding any schedulable Type A or B relations between individual tasks within $S(k)$. This is because only the relative values of pseudo-deadline matter when such relations are determined. If the algorithm succeeds in finding any eligible subset of size i, it goes on to the next iteration step. Otherwise, it tries finding an eligible subset of a different subset size; one size bigger.

Theorem 3: The Optimal Pseudo-Deadline Assignment (OPDA) algorithm is an optimal pseudo-deadline assignment policy with respect to the SPDF schedulability test of Theorem 2.

Proof: We first extend the notion of *eligible* to the JFP case. A subset of tasks Ω is *JFP-eligible* in a step k if there exists a schedulable pseudo-deadline assignment for all tasks $\tau_e \in \Omega$ under the assumption that τ_e has Type A or B relation with all the other tasks within the same eligible subset Ω, but has only Type A2 relation with all the other remaining tasks $\tau_r \in R(k)$.

We then consider two cases depending on how many eligible subsets exist: (1) only one and (2) more than one. First, suppose there is only one eligible subset $\Omega(k)$ in a step k. Since the algorithm tries to find $\Omega(k)$ for all possible size of i from 1 to $R(k)$ (Line 3 in Algorithm 1), it considers all combinations of i tasks with the given size of i (Line 2 in Algorithm 2) and explores all the possible pseudo-deadline assignment of each combination exhaustively (Line 11 in Algorithm 2). Thereby, it fails to find an eligible subset of i, for all $1 \le i \le R(k)$, only if there is no eligible subset in the k-th step. Therefore, the algorithm always find $\Omega(k)$, whenever it exists.

Second, given the optimality of OPDA when there is only one eligible subset, we then investigate whether OPDA can always make an optimal decision even when there are multiple different eligible subsets. For the purpose of simple presentation, let us assume that there exist two eligible subsets, $\Omega_1 = \{\tau_i, \tau_j\}$ and $\Omega_2 = \{\tau_i, \tau_p, \tau_q\}$, in the step k. We also assume that OPDA chooses Ω_1 for pseudo-deadline assignment in the same step. We denote $I_z(k)$ as a bound on the total interference that a task τ_z receives in the k-th step according to the SPDF schedulability test of Theorem 2. By definition, for each task $\tau_z \in \Omega_2$, $I_z(k) \le m(D_z - C_z + 1)$. Similar to the TFP/OPA case, the subset of remaining tasks only becomes smaller as the algorithm proceeds to the next iteration steps, yielding $R(k') \subset R(k)$, where $k < k'$. Here, we consider two sub-cases depending on whether the LC-technique is applied or not.

When the LC technique is not used, we wish to show that the total interference bound $I_q(k')$ that task τ_q receives in the k'-th step, will become only smaller than that in the k-th step. We note that since $\Omega_1 = \{\tau_i, \tau_j\}$ is selected in the k-th step, it immediately follows that $\{\tau_i, \tau_j\} \notin R(k')$ and $\tau_i \notin \Omega_2$ in the k'-th step. And this only decreases $I_q(k')$, leading to the following inequality.

$$I_q(k') \le I_q(k) \le m(D_q - C_q + 1). \quad (13)$$

The above inequality shows that task τ_q can be deemed schedulable if it can keep Type B relation with task τ_p, and Type A2 relation with all the other remaining tasks. That is, it implies that once a task belongs to an eligible subset in step k, it continues to belong to an eligible subset in all steps $k'(k < k')$. This allows us to conclude that it does not matter which eligible subset OPDA selects out of the multiple ones, because any task that is not selected will continue to be in some eligible subset in all future iterations.

We now consider the effect of the LC technique. Towards this, let us make a further assumption to Ω_1 and Ω_2 that the task τ_j has Type A1 relation with the task τ_q ($\tau_j \succ \tau_q$), and the task τ_p has Type A1-LC relation (Type A1 relation with the LC technique applied) with τ_q ($\tau_p \succ_{LC} \tau_q$) in the k-th step. In other words, suppose that τ_j incurs non-zero carry-in interference on τ_q, but τ_p has zero carry-in interference on τ_q in step k from the viewpoint of the SPDF analysis. We then wish to show that Eq. (13) holds even though OPDA selects Ω_1 for pseudo-deadline assignment in step k. In this case, it is worth noting that τ_p and τ_q, which are not selected in step

[5]In the OPDA algorithm, the pseudo-deadline buffer zone is initialized as $Z^H(1) = 0$. Note that $Z^L(0)$ is not used. We also note that since Type A relation is determined by the relative difference between pseudo-deadline values, $Z^H(1)$ can be initialized to any arbitrary value.

k, can have a different relation in a later step k' ($k < k'$) such that τ_p has Type A1 (instead of A1-LC) relation with τ_q in step k'. This way, τ_p can impose a larger interference on τ_q in step k', compared to step k. Since we have so far assumed that the total interference bound that τ_q receives will remain the same or become smaller as iteration goes on, we need to investigate this situation carefully.

Note that the LC analysis technique keeps carry-in for m-1 tasks with the largest difference between $I_{i,k}^{(A1)}$ and $\hat{I}_{i,k}^{(A1)}$ and excludes carry-in for all the other tasks within the same Type A1 relation group. From the assumption that τ_j has non-zero carry-in interference but τ_p has no carry-in interference in step k, we can see that τ_j incurs a larger carry-in interference on τ_q than τ_p does, that is,

$$I_{p,q}^{(A1)} - \hat{I}_{p,q}^{(A1)} \leq I_{j,q}^{(A1)} - \hat{I}_{j,q}^{(A1)}. \tag{14}$$

Then, the total interference bound given on task τ_q in step k can be described as follows.

$$\sum_{\tau_r \in R(k), r \neq i,j,p,q} I_{r,q}^{(A1)} + I_{j,q}^{(A1)} + I_{i,q}^{(B)} + \hat{I}_{p,q}^{(A1)} \leq m(D_q - C_q + 1). \tag{15}$$

Then, the total interference bound that τ_q receives in step $k'(k < k')$ can be represented as follows.

$$\begin{aligned}
&\sum_{\tau_r \in R(k'), r \neq i,j,p,q} I_{r,q}^{(A1)} + I_{p,q}^{(A1)} \\
\leq &\sum_{\tau_r \in R(k'), r \neq i,j,p,q} I_{r,q}^{(A1)} + \hat{I}_{p,q}^{(A1)} + I_{j,q}^{(A1)} - \hat{I}_{j,q}^{(A1)} \quad \text{by Eq. (14)} \\
\leq &\sum_{\tau_r \in R(k), r \neq i,j,p,q} I_{r,q}^{(A1)} + \hat{I}_{p,q}^{(A1)} + I_{j,q}^{(A1)} \quad \text{since } R(k') \subset R(k) \\
\leq &m(D_q - C_q + 1). \quad \text{by Eq. (15)}
\end{aligned} \tag{16}$$

The above inequality indicates that even though the interference that τ_q receives from an individual task can increase over iteration steps due to the LC technique, the total interference on task τ_q still becomes smaller. This confirms the optimality of OPDA when there are multiple eligible subsets, even if the DA-LC technique is used in the schedulability test. This completes the proof. ■

Complexity. We denote the number of tasks in a task set by n. At each iteration step k in Algorithm 1, OPDA tries to find a subset $S(k)$ of size i. For each size i in $1 \leq i \leq R(k) \leq n$, it considers all combinations of i tasks, and the number of combinations is $\binom{n}{i}$. Then, for each combination of i tasks, it explores all permutations of possible relative pseudo-deadlines, and the number of permutations is bounded by $i \cdot maxD(k)^i$. For each pseudo-deadline assignment, it checks the SPDF schedulability test in Theorem 2 that requires $O(n^2 \cdot log(n))$. Therefore, OPDA requires $O(n^{n+2} \cdot log(n))$.

VI. Heuristic Pseudo-deadline Assignment

The OPDA algorithm presented in the previous section is optimal with respect to the given schedulability test but its computational complexity is exponential in the number of tasks. Hence, this section presents heuristics with low computational complexity to find sub-optimal solutions.

As described previously, OPDA exhaustively searches an eligible subset of different size k in each iteration step t, from one to the number of remaining tasks (i.e., $1 \leq k \leq |R(t)|$). We investigate tradeoff between performance and complexity for OPDA through different subset size. We let OPDA-k denote the OPDA algorithm that finds an eligible subset of size up to k only. We note that TFP/OPA and OPDA are specializations of OPDA-k, where $k = 1$ and $k = |\tau|$, respectively.

We then also consider a heuristic algorithm, called LSF (Largest Slack First). The intuition behind this algorithm is that increasing the pseudo-deadline of a task τ_s increases only the total interference bound on task τ_s, but decreases the interference bounds of all the other tasks τ_r. This increases the possibility of each task τ_r being schedulable, albeit at the expense of task τ_s.

In this paper, we let σ_j denote the *slack* of a task τ_j and define it as follows.

$$\sigma_j = (D_j - C_j + 1) - \Big\lfloor \frac{\sum_{i \neq j} I_{i,j}^{SPDF}}{m} \Big\rfloor. \tag{17}$$

The LSF algorithm works as follows. It repeats the followings steps until τ is deemed schedulable or it reaches a certain number of iterations. During an iteration, it first finds a task τ_s^* with largest slack among all the remaining tasks. Then, it decreases its slack σ_s^* by increasing its pseudo-deadline P_s^* until the slack σ_s^* does not decrease any more or the task τ_s^* is deemed unschedulable. We note that the relative pseudo-deadline of a task τ_j can be randomly set for the calculation of σ_j at the first iteration. We observe that the initial value of relative pseudo-deadline rarely influences the performance of the LSF algorithm.

We propose a heuristic approach, called HPDA-k (Heuristic Pseudo-Deadline Assignment) combining OPDA-k and LSF. Given a task set τ, HPDA-k employs OPDA-k first to find a pseudo-deadline assignment that makes τ schedulable. If OPDA-k succeeds in finding such a schedulable assignment, HPDA-k stops here. Otherwise, HPDA-k employs LSF over the remaining tasks from OPDA-k to find pseudo-deadline assignments that make τ schedulable.

Complexity. HPDA-k employs OPDA-k that finds an eligible subset of size up to k only. Therefore, it requires $O(n^{k+2} \cdot log(n))$.

VII. Evaluation

This section presents simulation results to evaluate our pseudo-deadline assignment algorithms and compare their performance to existing TFP and JFP algorithms. As a main

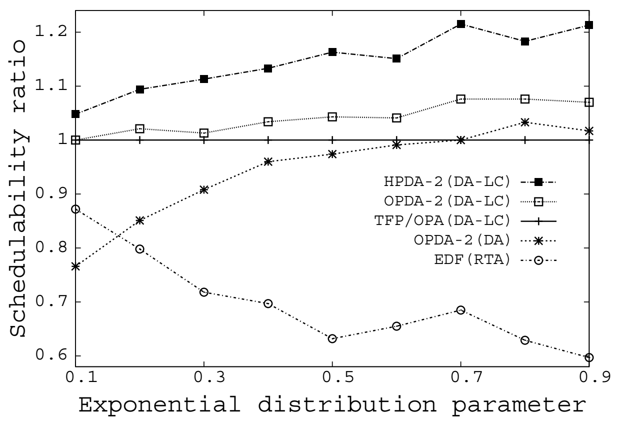


Figure 5. Schedulability ratios of various pseudo-deadline assignment algorithms

metric for comparison we use the *schedulability ratio*, which is defined as the number of task sets deemed schedulable by an algorithm to the total number of generated task sets.

Figure 5. We generate task sets based on a technique proposed earlier [23], which has also been used in many previous studies [21], [13]. We have a density distribution parameter for exponential distribution of individual density: $0.1, 0.3, 0.5, 0.7, 0.9$. For each task τ_i, D_i is uniformly chosen in [1000, 2000], T_i is uniformly chosen in [D_i, 2000], and C_i is chosen based on the exponential density distribution parameter. For each exponential density distribution parameter, we generate 1,000 task sets with $m = 4$ where m is the number of processors. Initially, we generate a set of $m + 1$ tasks, and create a new set by adding a new task into the old set until the system density becomes greater than m.

Figure 5 compares the schedulability ratios of three pseudo-deadline assignment schemes (TFP/OPA, OPDA-2, and HPDA-2) with DA, DA-LC, or RTA (response time analysis) [6] analysis, relative to the case of TFP/OPA with DA-LC. Here, HPDA-2 was allowed to run the LSF algorithm up to 1,000 times for each individual task set until a schedulable pseudo-deadline assignment is found. The complexity of OPDA grows exponentially as the subset size k increases, so OPDA-2 and HPDA-2 is only shown in Figure 5. A more detailed comparison between OPDA-k and HPDA-k with different subset size k will be presented later in this section. As the value of the exponential distribution parameter increases, it generates task sets with a smaller number of tasks and tasks are more likely to have larger densities. In the figure, the effect of the LC analysis can be noticed by the difference between OPDA-2(DA) and OPDA-2(DA-LC). Such a difference is significant when the exponential parameter is small (i.e., when the number of tasks is large). This is because a larger number of tasks can get benefit from the limited carry-in feature. The difference between OPDA-2(DA-LC) and TFP/OPA(DA-LC) shows the improvement of allowing Type B inter-task priority relation in priority (pseudo-deadline) assignment, compared to the TFP/OPA case where only Type A relation is granted. Such an improvement grows when the exponential parameter increases (i.e., when tasks have higher densities). This is because there are more cases where tasks with higher densities cannot accept full interference from others under Type A relation, but can accommodate some partial interference from each other under Type B relation. Finally, the contribution of the LSF algorithm is shown by the difference between OPDA-2(DA-LC) and HDPA-2(DA-LC). Due to its significant contribution, HPDA-2(DA-LC) can outperform TFP/OPA (DA-LC) by 5-21% over all exponential distributions. The figure also shows that TFP/OPA(DA-LC) significantly outperforms EDF(RTA), which is consistent with the results in [9], and so does HPDA-2(DA-LC).

	Schedulability ratio relative to optm (%)	Running time (ms)
OPDA-1	86.6	4.2×10^2
HPDA-1	91.6	4.5×10^3
OPDA-2	97.6	6.2×10^2
HPDA-2	98.2	5.9×10^3
OPDA-3	99.4	5.7×10^3
HPDA-3	99.6	9.1×10^3
OPDA-4	99.9	1.7×10^5
HPDA-4	99.9	1.7×10^5
OPDA-5	100.0	4.9×10^6
HPDA-5	100.0	4.9×10^6
optm	100.0	6.0×10^6

Table I
TRADEOFF BETWEEN SCHEDULABILITY AND RUNNING TIME OF OPDA VIA DIFFERENT SUBSET SIZE k

Table I. Our second simulations were performed to investigate the tradeoff between performance and complexity of our OPDA algorithm through different subset size k. The complexity of OPDA grows exponentially as the number of tasks (n) and/or the range of D_i increase, becoming easily intractable. Thus, we generated task sets with different simulation parameters from those for Figure 5, with a smaller number of tasks ($n = 5$) and a smaller range of deadlines ($T_i = D_i$ uniformly chosen in $[1, 10]$). We then ran simulations on 10,000 task sets with $m = 2$. In the simulations, an exhaustive search (optm) was conducted to find optimal solutions and serves as a baseline to compare the results of OPDA-k, for all $1 \leq k \leq 5$. Table I shows the schedulability ratio and running time of OPDA-k relative to optm. For example, OPDA-1 (HPDA-1) finds 86.8% (91.6%) of all schedulable task sets with a running time four (three) orders of magnitude shorter than optm. We note that OPDA-1 is equivalent to TFP/OPA, and in this case, OPDA-5 and HPDA-5 produce the same optimal result as optm. This table also shows that as k increases, the schedulability

ratio gap between optm and OPDA-k rapidly decreases at the expense of an exponential increase in running time. This implies that the subset size k is a good control knob to balance the schedulability vs. complexity tradeoff of the OPDA algorithm, establishing a good basis for our heuristic approach HPDA-k.

VIII. Conclusion

The motivation for our work was to advance JFP scheduling techniques by adapting high-performing TFP scheduling techniques. One of the main difficulties in applying TFP-specific techniques to JFP scheduling was a perception that JFP scheduling can support tasks only with Type B priority dominance relations, while TFP-specific techniques can support Type A relation only. To overcome this, we identified a condition under which JFP scheduling allows a subset of tasks to behave under Type A relation. Building upon it, this paper has introduced a JFP scheduling algorithm, called SPDF, that is better than all TFP scheduling algorithms and well-known JFP algorithms when pseudo-deadlines are properly assigned. This paper has generalized the high-performing LC analysis technique and the OPA algorithm, which are highly TFP-oriented, for SPDF.

In this paper, we have focused on understanding a fundamental difference between TFP and JFP scheduling. Based on the understanding, we have improved priority assignment algorithms and schedulability analysis for JFP scheduling by generalizing TFP techniques to JFP. However, we also believe JFP scheduling techniques can be further enhanced when JFP-centric properties are appropriately exploited, which is for our future research.

Acknowledgement

The authors would like to thank the anonymous reviews for their helpful and constructive comments. This work was supported in part by ARO W911NF-11-1-0403 and NSF CNS-1117185. It was also partially funded by BRL (2009-0086964), BSRP (2010-0006650, 2012-R1A1A1014930), NCRC (2012-0000980), NIPA (2010-C1810-1102-0003), KEIT (2011-10041313), and KIAT (M002300089) funded by the Korea Government (MEST/MKE).

References

[1] J. Leung and J. Whitehead, “On the complexity of fixed-priority scheduling of periodic real-time tasks,” *Performance Evaluation*, vol. 2, pp. 237–250, 1982.

[2] C. Liu and J. Layland, “Scheduling algorithms for multi-programming in a hard-real-time environment,” *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[3] M. L. Dertouzos and A. K. Mok, “Multiprocessor on-line scheduling of hard-real-time tasks,” *IEEE Transactions on Software Engineering*, vol. 15, pp. 1497–1506, 1989.

[4] R. Davis and A. Burns, “Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems,” in *RTSS*, 2009.

[5] B. Andersson, S. Baruah, and J. Jonsson, “Static-priority scheduling on multiprocessors,” in *RTSS*, 2001.

[6] M. Bertogna and M. Cirinei, “Response-time analysis for globally scheduled symmetric multiprocessor platforms,” in *RTSS*, 2007.

[7] N. Guan, M. Stigge, W. Yi, and G. Yu, “New response time bounds of fixed priority multiprocessor scheduling,” in *RTSS*, 2009.

[8] S. Baruah, “Techniques for multiprocessor global schedulability analysis,” in *RTSS*, 2007.

[9] R. Davis and A. Burns, “Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems,” *RTSJ*, vol. 47, no. 1, pp. 1–40, 2011.

[10] S. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, “Proportionate progress: a notion of fairness in resource allocation,” *Algorithmica*, vol. 15, no. 6, pp. 600–625, 1996.

[11] G. Levin, S. Funk, C. Sadowski, I. Pye, and S. A. Brandt, “Dp-fair: A simple model for understanding optimal multi-processor scheduling,” in *ECRTS*, 2010.

[12] P. Regnier, G. Lima, E. Massa, G. Levin, and S. A. Brandt, “Run: Optimal multiprocessor real-time scheduling via reduction to uniprocessor,” in *RTSS*, 2011.

[13] J. Lee, A. Easwaran, and I. Shin, “Maximizing contention-free executions in multiprocessor scheduling,” in *RTAS*, 2011.

[14] S. Cho, S.-K. Lee, S. Ahn, and K.-J. Lin, “Efficient real-time scheduling algorithms for multiprocessor systems,” *IEICE Trans. on Communications*, vol. E85–B, no. 12, pp. 2859–2867, 2002.

[15] A. Srinivasan and S. Baruah, “Deadline-based scheduling of periodic task systems on multiprocessors,” *Information Processing Letters*, vol. 84, no. 2, pp. 93–98, 2002.

[16] S. K. Baruah, “Optimal utilization bounds for the fixed-priority scheduling of periodic task systems on identical multiprocessors,” *IEEE Transactions on Computers*, vol. 53, pp. 781–784, 2004.

[17] H. Back, H. S. Chwa, and I. Shin, “Schedulability analysis and priority assignment for global job-level fixed-priority multiprocessor scheduling,” in *RTAS*, 2012.

[18] J. Goossens, S. Funk, and S. Baruah, “Priority-driven scheduling of periodic task systems on multiprocessors,” *Real-Time Systems*, vol. 25, no. 2–3, pp. 187–205, 2003.

[19] M. Bertogna, M. Cirinei, and G. Lipari, “Improved schedulability analysis of EDF on multiprocessor platforms,” in *ECRTS*, 2005.

[20] T. P. Baker, “Multiprocessor EDF and deadline monotonic schedulability analysis,” in *RTSS*, 2003.

[21] M. Bertogna, M. Cirinei, and G. Lipari, “Schedulability analysis of global scheduling algorithms on multiprocessor platforms,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, pp. 553–566, 2009.

[22] J. Lee, A. Easwaran, and I. Shin, “LLF Schedulability Analysis on Multiprocessor Platforms,” in *RTSS*, 2010.

[23] T. Baker, “An analysis of EDF schedulability on a multiprocessor,” *IEEE Transactions on Parallel Distributed Systems*, vol. 16, no. 8, pp. 760–768, 2005.