Improved Conditions for Bounded Tardiness under EPDF Fair Multiprocessor Scheduling *

UmaMaheswari C. Devi and James H. Anderson
Department of Computer Science, The University of North Carolina, Chapel Hill, NC

Abstract

The earliest-pseudo-deadline-first (EPDF) Pfair algorithm is more efficient than other known Pfair scheduling algorithms, but is not optimal on more than two processors. In earlier work, Srinivasan and Anderson established a sufficient per-task utilization restriction for ensuring a tardiness of at most one quantum under EPDF. They also conjectured that a tardiness bound of one quantum applies to systems that are not restricted in any way. In this paper, we present counterexamples that show that this conjecture is false. We also present sufficient utilization restrictions that are more liberal than theirs.

1 Introduction

Pfair scheduling, originally introduced by Baruah *et al.* [4], is the only known way of optimally scheduling recurrent real-time tasks on multiprocessors. Under Pfair scheduling, each task must execute at an approximately uniform rate, while respecting a fixed-size allocation quantum. A task's execution rate is defined by its *weight* (or *utilization*). Uniform rates are ensured by subdividing each task T into quantum-length *subtasks* that are subject to intermediate deadlines. To avoid deadline misses, ties among subtasks with the same deadline must be broken carefully. In fact, tie-breaking rules are crucial when devising optimal Pfair scheduling algorithms.

As discussed by Srinivasan and Anderson [8], overheads associated with tie-breaking rules may be unnecessary or unacceptable for many soft real-time systems. A soft real-time task differs from a hard real-time task in that its deadlines may sometimes be missed. If a job (*i.e.*, task instance) or a subtask with a deadline at time d completes executing at time t, then it is said to have a *tardiness* of $\max(0, t - d)$.

Systems with quality-of-service requirements, such as multimedia applications, are examples where tie-breaking rules may be unnecessary. Here, fair resource allocation is necessary to provide service guarantees, but occasional deadline misses often result in tolerable performance degradation. Hence, an extreme notion of fairness that precludes all deadline misses is usually not required.

In dynamic systems that permit tasks to join or leave, the overhead introduced by tie-breaking rules may be unacceptable. In such a system, spare processing capacity may become available. To make use of this capacity, task weights must be changed on-the-fly. It is possible to reweight each task so that its next subtask deadline is preserved [8]. If no tie-breaking information is maintained, such an approach entails very little overhead. However, weight changes can cause tie-breaking information to change, so if tie-breaking rules are used, reweighting may necessitate a $\Omega(N \log N)$ cost for N tasks, due to the need to re-sort the scheduler's priority queue. This cost may be prohibitive if load changes are frequent.

The observations above motivated Srinivasan and Anderson to consider the viability of scheduling soft real-time applications using the simpler *earliest-pseudo-deadline-first* (EPDF) Pfair algorithm, which uses no tie-breaking rules. They succeeded in showing that EPDF can guarantee a tardiness bound of one quantum for every subtask, provided a certain condition holds. This condition, which is described in detail later, can be ensured by limiting each task's weight to at most 1/2, and can be generalized to apply to tardiness bounds other than one. Unfortunately, Srinivasan and Anderson left open the question of whether such conditions are necessary to guarantee small constant tardiness.

In this paper, we provide counterexamples that show that, in general, restrictions on individual task utilizations are *necessary* to guarantee constant tardiness bounds. In addition, we show that, in general, a more liberal per-task weight restriction of 2/3 (66.7%) is sufficient to ensure a tardiness of one quantum, and that for a somewhat special case, which is described in Sec. 3, this restriction can be relaxed to 11/15 (73.3%). We also present generalizations of these conditions that can be applied to other tardiness bounds.

The rest of the paper is organized as follows. In Sec. 2, needed definitions are given. In Sec. 3, the results above are proved. Sec. 4 concludes.

*Work supported by NSF grants CCR 9988327, ITR 0082866, CCR 0204312, and CCR 0309825.

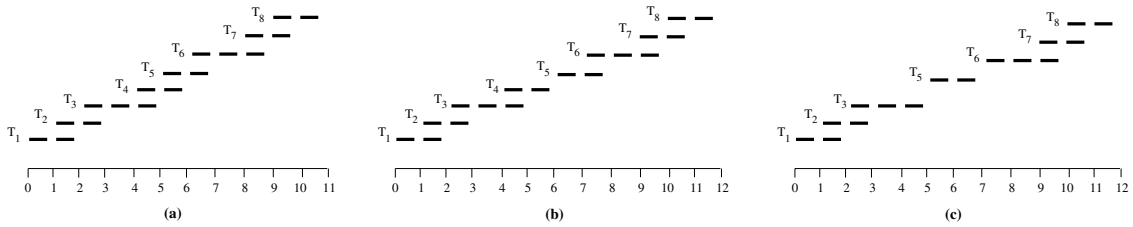

Figure 1. **(a)** Windows of the first job of a periodic task T with weight $8/11$. This job consists of subtasks $T_1, \ldots, T_8$, each of which must be scheduled within its window, or else a lag-bound violation will result. (This pattern repeats for every job.) **(b)** The Pfair windows of an IS task. Subtask T_5 becomes eligible one time unit late. **(c)** The Pfair windows of a GIS task. Subtask T_4 is absent and T_6 is one time unit late.

2 Pfair Scheduling

In this section, Pfair scheduling is defined and some prior results summarized [1, 2, 3, 4, 7, 8]. To begin with, we limit attention to periodic tasks that begin execution at time 0. Such a task T has an integer *period* $T.p$, an integer *execution cost* $T.e$, and a *weight* $wt(T) = T.e/T.p$, where $0 < wt(T) < 1$. A task is *light* if its weight is less than $1/2$, and *heavy* otherwise.

Pfair algorithms allocate processor time in discrete quanta; the time interval $[t, t+1)$ is called *slot* t. (Hence, time t refers to the beginning of slot t.) A task may be allocated time on different processors, but not in the same slot (*i.e.*, interprocessor migration is allowed but parallelism is not). The sequence of allocation decisions over time defines a *schedule* S. Formally, $S : \tau \times \mathcal{N} \mapsto \{0, 1\}$, where τ is a task set. $S(T, t) = 1$ iff T is scheduled in slot t. On M processors, $\sum_{T \in \tau} S(T, t) \leq M$ holds for all t.

Lags and subtasks. The notion of a Pfair schedule is defined by comparing such a schedule to an ideal fluid schedule, which allocates $wt(T)$ processor time to task T in each slot. Deviation from the fluid schedule is formally captured by the concept of *lag*. Formally, the *lag of task T at time t* is defined by[1] $lag(T, t) = wt(T) \cdot t - \sum_{u=0}^{t-1} S(T, u)$. A schedule is defined to be *Pfair* iff

$$(\forall T, t :: -1 < lag(T, t) < 1). \quad (1)$$

Informally, the allocation error associated with each task must always be less than one quantum.

These lag bounds have the effect of breaking each task T into an infinite sequence of quantum-length *subtasks*, $T_1, T_2, \ldots$. Each subtask has a *pseudo-release* $r(T_i)$ and a *pseudo-deadline* $d(T_i)$, where

$$r(T_i) = \left\lfloor \frac{i-1}{wt(T)} \right\rfloor \quad \wedge \quad d(T_i) = \left\lceil \frac{i}{wt(T)} \right\rceil. \quad (2)$$

(For brevity, we often omit the prefix "pseudo-.") To satisfy (1), T_i must be scheduled in the interval $w(T_i) = [r(T_i), d(T_i))$, termed its *window*. The *length* of T_i's window, denoted $|w(T_i)|$, is $d(T_i) - r(T_i)$. As an example, consider subtask T_1 in Fig. 1(a). Here, we have $r(T_1) = 0$, $d(T_1) = 2$, and $|w(T_1)| = 2$. Hence, T_1 must be scheduled at either time 0 or time 1.

Note that $r(T_{i+1})$ is either $d(T_i) - 1$ or $d(T_i)$. Thus, consecutive windows either overlap by one slot, or are disjoint. The "b-bit," denoted by $b(T_i)$, distinguishes between these possibilities. Formally, $b(T_i) = \left\lceil \frac{i}{wt(T)} \right\rceil - \left\lfloor \frac{i}{wt(T)} \right\rfloor$. For example, in Fig. 1(a), $b(T_i) = 1$ for $1 \leq i \leq 7$ and $b(T_8) = 0$.

It can be shown that all windows of a heavy task are of length two or three. For such tasks, the "group deadline" is used to mark the end of a sequence of windows of length two. Consider a sequence $T_i, \ldots, T_j$ of subtasks of a heavy task T such that $b(T_k) = 1$, $|w(T_{k+1})| = 2$ for all $i \leq k < j$. Then, scheduling T_i in its last slot forces the other subtasks in this sequence to be scheduled in their last slots. For example, in Fig. 1(a), scheduling T_3 in slot 4 forces T_4 and T_5 to be scheduled in slots 5 and 6, respectively. A group deadline corresponds to a time by which any such "cascade" of scheduling decisions must end. Formally, it is a time t such that either $(t = d(T_i) \wedge b(T_i) = 0)$ or $(t + 1 = d(T_i) \wedge |w(T_i)| = 3)$ for some subtask T_i. For example, the task in Fig. 1(a) has group deadlines at times 4, 8, and 11.

We let $D(T_i)$ denote the group deadline of subtask T_i. If T is heavy, then $D(T_i) = (\min u : u \geq d(T_i) \wedge u$ is a group deadline of $T)$. For example, in Fig. 1(a), $D(T_1) = 4$ and $D(T_6) = 11$. If T is light, then $D(T_i) = 0$.

Task models. In this paper, we consider the *intra-sporadic* (IS) and the *generalized-intra-sporadic* (GIS) task models [2, 7], which provide a general notion of recurrent execution that subsume that found in the well-studied periodic and sporadic task models. The *sporadic* model generalizes the periodic model by allowing jobs to be released "late"; the IS model allows subtasks to be released late, as illustrated in Fig. 1(b). More specifically, the separation between $r(T_i)$ and $r(T_{i+1})$ is allowed to be more than $\lfloor i/wt(T) \rfloor - \lfloor (i-1)/wt(T) \rfloor$, which would be the sepa-

[1] For conciseness, we leave the schedule implicit and use $lag(T, t)$ instead of $lag(T, t, S)$.

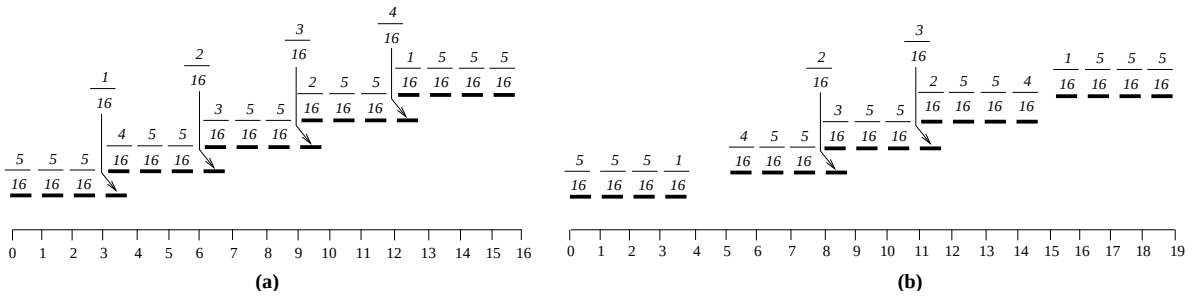

Figure 2. Fluid schedule for the first five subtasks ($T_1, \ldots, T_5$) of a task T of weight $5/16$. The share of each subtask in each slot of its PF-window is shown. In **(a)**, no subtask is released late; in **(b)**, T_2 and T_5 are released late. Note that $share(T, 3)$ is either $5/16$ or $1/16$ depending on when subtask T_2 is released.

ration if T were periodic. Thus, an IS task is obtained by allowing a task's windows to be shifted right from where they would appear if the task were periodic.

Let $\theta(T_i)$ denote the offset of subtask T_i, *i.e.*, the amount by which $w(T_i)$ has been shifted right. Then, by (2), we have the following.

$$r(T_i) = \theta(T_i) + \left\lfloor \frac{i-1}{wt(T)} \right\rfloor \ \wedge \ d(T_i) = \theta(T_i) + \left\lceil \frac{i}{wt(T)} \right\rceil \quad (3)$$

The offsets are constrained so that the separation between any pair of subtask releases is at least the separation between those releases if the task were periodic. Formally, $k > i \Rightarrow \theta(T_k) \geq \theta(T_i)$.

Each subtask T_i has an additional parameter $e(T_i)$ that specifies the first time slot in which it is eligible to be scheduled. We require $e(T_i) \leq r(T_i)$ and $e(T_i) \leq e(T_{i+1})$ for all $i \geq 1$. Additionally, no subtask can become eligible before its predecessor completes execution. The interval $[r(T_i), d(T_i))$ is called the *PF-window* of T_i and the interval $[e(T_i), d(T_i))$ is called the *IS-window* of T_i. A schedule for an IS system is *valid* iff each subtask is scheduled in its IS-window.

b-bits for IS tasks are defined in the same way as for periodic tasks. $r(T_i)$ is defined as follows.

$$r(T_i) = \begin{cases} e(T_i), & \text{if } i = 1 \\ max(e(T_i), d(T_{i-1}) - b(T_{i-1})), & \text{if } i \geq 2 \end{cases} \quad (4)$$

T_i's deadline $d(T_i)$ is defined to be $r(T_i) + |w(T_i)|$. PF-window lengths are defined as before. Thus, by (2), we have $d(T_i) = r(T_i) + \left\lceil \frac{i}{wt(T)} \right\rceil - \left\lfloor \frac{i-1}{wt(T)} \right\rfloor$.

Generalized intra-sporadic (GIS) task systems. A GIS task system is obtained by removing subtasks from a corresponding IS task system, and thus, is a more general model than the IS model. Specifically, in a GIS task system, a task T, after releasing subtask T_i, may release subtask T_k, where $k > i + 1$, instead of T_{i+1}, with the following restriction: $r(T_k) - r(T_i)$ is at least $\left\lfloor \frac{k-1}{wt(T)} \right\rfloor - \left\lfloor \frac{i-1}{wt(T)} \right\rfloor$. For the special case where T_k is the first subtask released by T, $r(T_k)$ must be at least $\left\lfloor \frac{k-1}{wt(T)} \right\rfloor$. Fig. 1(c) shows an example. If a task T, after executing subtask T_i, releases subtask T_k, then T_k is called the *successor* of T_i and T_i is called the *predecessor* of T_k.

As shown in [2], an IS or GIS task system τ is feasible on M processors iff

$$\sum_{T \in \tau} wt(T) \leq M. \quad (5)$$

Algorithm EPDF. The earliest-pseudo-deadline-first (EPDF) Pfair scheduling algorithm, considered in this paper, is optimal on one or two processors, but not on more than two processors [3]. At each time t, EPDF schedules at most M eligible subtasks with the highest priority. As its name suggests, higher priority is given to subtasks with earlier deadlines; a tie between subtasks with equal deadlines is broken arbitrarily.

Shares and lags in IS and GIS task systems. $lag(T, t)$ is defined for IS and GIS tasks as before [7]. Let $ideal(T, t)$ denote the processor share that T receives in an ideal fluid (processor-sharing) schedule in $[0, t)$. Then,

$$lag(T, t) = ideal(T, t) - \sum_{u=0}^{t-1} S(T, u). \quad (6)$$

Towards defining $ideal(T, t)$, we define $share(T, u)$, which is the share assigned to task T in slot u. $share(T, u)$ is defined in terms of a function $f(T_i, t)$ that indicates the share assigned to subtask T_i in slot t. $f(T_i, t)$ is defined as follows.

$$\begin{cases} (\left\lfloor \frac{i-1}{wt(T)} \right\rfloor + 1) \times wt(T) - (i-1), & \text{if } t = r(T_i) \\ i - (\left\lceil \frac{i}{wt(T)} \right\rceil - 1) \times wt(T), & \text{if } t = d(T_i) - 1 \\ wt(T), & \text{if } t \in (r(T_i), d(T_i){-}1) \\ 0, & \text{otherwise} \end{cases} \quad (7)$$

Fig. 2 shows the values of f for different subtasks of a task of weight 5/16. Given f, $share(T, u)$ can be defined as $share(T, u) = \sum_i f(T_i, u)$. As shown in Fig. 2, $share(T, u)$ usually equals $wt(T)$, but in certain slots, it

may be less than $wt(T)$. We can now define $ideal(T,t)$ as $\sum_{u=0}^{t-1} share(T,u)$. Hence, from (6),

$$\begin{aligned} lag(T,t+1) &= \sum_{u=0}^{t} (share(T,u) - S(T,u)) \\ &= lag(T,t) + share(T,t) - S(T,t). \end{aligned} \quad (8)$$

Similarly, the total lag for a schedule S and task system τ at time $t+1$, denoted $LAG(\tau, t+1)$, is as follows. ($LAG(\tau,0)$ is defined to be 0.)

$$LAG(\tau,t+1) = LAG(\tau,t) + \sum_{T \in \tau} (share(T,t) - S(T,t)). \quad (9)$$

The following lemma gives two properties concerning the f values of subtasks as defined by (7).

Lemma 1 [**6**] *Let f be as defined by (7). Then, the following hold.*

(**a**) *If $b(T_{i-1}) = 1$ and subtask T_i exists, then $f(T_{i-1}, d(T_{i-1})) + f(T_i, r(T_i)) = wt(T)$.*

(**b**) $(\forall T_i, t :: f(T_i,t) \geq \frac{1}{T.p})$.

3 Tardiness Bounds for EPDF

In this section, we present results concerning tardiness bounds that can be guaranteed under EPDF. As mentioned earlier, if subtask T_i completes execution at time t, then its tardiness is given by $\max(0, t - d(T_i))$. The *tardiness of a task system* is defined as the maximum tardiness among all of its subtasks in any schedule.

It is easy to show that subtask deadlines can be missed under EPDF. In [8], it was conjectured that EPDF always ensures a tardiness of at most one. We now show that this conjecture is false.

Theorem 1 *Tardiness under EPDF can exceed three quanta. In particular, if EPDF is used to schedule task system τ_i $(1 \leq i \leq 3)$ in Table 1, then a tardiness of $i+1$ quanta is possible.*

Proof: Fig. 3 shows a schedule for τ_1, in which a subtask has a tardiness of two at time 50. The schedules for τ_2 and τ_3 are too lengthy to be depicted; we verified them using two EPDF simulators. □

The sufficient condition for a tardiness of one as given by Srinivasan and Anderson requires that the sum of the weights of the $M-1$ heaviest tasks be less than $\frac{M+1}{2}$. This can be ensured if the weight of each task is restricted to be at most $1/2$. We next show that, in general, a weight restriction of $2/3$ (66.7%) per task is sufficient to guarantee a tardiness of one, and that for the special case where a subtask does not become eligible before its release time, this restriction can be improved to $11/15$ (73.3%). These restrictions are stated below.

Table 1. Counterexamples to show that tardiness under EPDF can exceed three.

	Task Set		Util. (M)	Tardiness (in quanta)
	# of tasks	weight		
τ_1	4	1/2	10	2 at 50
	3	3/4		
	6	23/24		
τ_2	4	1/2	19	3 at 963
	3	3/4		
	5	23/24		
	10	239/240		
τ_3	4	1/2	80	4 at 43,204
	3	3/4		
	3	23/24		
	1	31/32		
	4	119/120		
	4	239/240		
	6	479/480		
	8	959/960		
	15	1199/1200		
	15	2399/2400		
	20	4799/4800		

(C) The weight of each task is at most $2/3$.
(D) The weight of each task is at most $11/15$, and for every subtask T_i, $e(T_i) = r(T_i)$.

In this paper, we prove the following theorem, which states that (C) is sufficient for EPDF to guarantee a tardiness of at most one.

Theorem 2 *EPDF ensures a tardiness of at most one quantum for feasible GIS task systems that satisfy (C).*

The proof of the theorem stated next, which establishes the sufficiency of (D), can be found in [5]. (It is the same as that for Theorem 2, except for one case.)

Theorem 3 *EPDF ensures a tardiness of at most one quantum for feasible GIS task systems that satisfy (D).*

Before proving Theorem 2, we reproduce some helpful definitions and lemmas from [7] and [8].

In a schedule S, if k processors are idle at time slot t, then we say that there are k *holes* in S at slot t. The following lemma relates an increase in total lag to the presence of holes.

Lemma 2 [**7**] *If $LAG(\tau,t) < LAG(\tau,t+1)$, then there is a hole in slot t in S.*

We prove Theorem 2 in a manner similar to that used in [8]. If (C) is not sufficient, then t_d and τ defined as follows both exist.

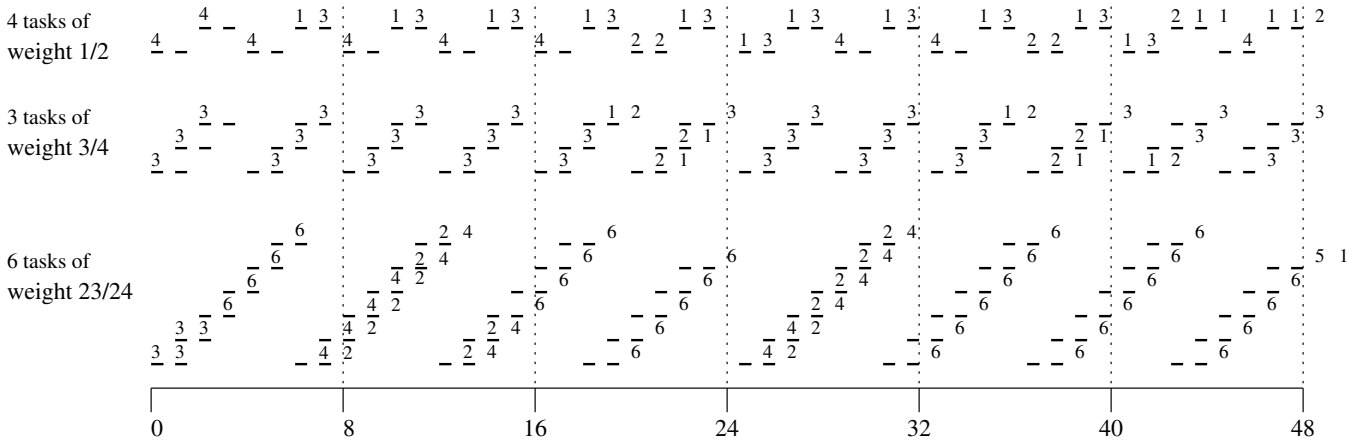

Figure 3. Counterexample to prove that tardiness under EPDF can exceed one quantum. 13 periodic tasks with total utilization 10 are scheduled on 10 processors using EPDF. In the schedule, tasks of the same weight are shown together as a group. Each column corresponds to a time slot. The Pfair window of each subtask is shown as a sequence of dashes that are aligned. An integer value n in slot t means that n tasks in the corresponding group have a subtask scheduled at t. Subtasks that miss their deadlines are shown scheduled after their windows. Ties are broken in favor of tasks with lower weights. In this schedule, 11 subtasks miss their deadlines at time 48. Hence, at least one subtask has a tardiness of two quanta.

Definition 1: t_d is the earliest deadline of a subtask with a tardiness of two under EPDF in any task system satisfying (C), *i.e.*, there exists some task system with a subtask with a deadline at t_d and a tardiness of two, and there does not exist any other task system with a subtask with a deadline prior to t_d and a tardiness of two.

Definition 2: τ is a feasible task system satisfying (C) with the following properties.

$(\mathbf{T1})$ t_d is the earliest deadline of a subtask in τ with a tardiness of two under EPDF.

(T2) No feasible task system satisfying (C) and (T1) releases fewer subtasks in $[0, t_d)$ than τ.

(T3) No feasible task system satisfying (C), (T1), and (T2) has a larger rank than τ at t_d, where rank is defined as follows.

The *rank* of a system τ at t is the sum of the eligibility times of all subtasks with deadlines at most t.

By (T1) and (T2), exactly one subtask in τ has a tardiness of two: if several such subtasks exist, then all but one can be removed and the remaining subtask will still have a tardiness of two, contradicting (T2). Additionally, the following assertions follow from the above properties and definitions.

(A1) $(\exists T_i \in \tau : d(T_i) = t_d \wedge\ tardiness(T_i) = 2)$
(A2) $(\forall T_i \in \tau : d(T_i) < t_d \Rightarrow tardiness(T_i) \leq 1)$

In the rest of this paper, we use S to denote an EPDF schedule for τ on M processors, in which a subtask with a deadline at t_d has a tardiness of two. The following lemma summarizes some properties of τ and S. It is proved in [7], [8], and [5].

Lemma 3 *The following properties hold for τ and S, where T_i is any subtask in τ.*

(**a**) *For all T_i, $d(T_i) \leq t_d$.*

(**b**) $LAG(\tau, t_d) \geq M + 1$.

(**c**) *There are no holes in slot $t_d - 1$.*

(**d**) $LAG(\tau, t_d - 1) \geq M + 1$.

(**e**) *There exists a time $u \in [0, t_d - 2]$ such that $LAG(\tau, u) < M + 1$ and $LAG(\tau, u + 1) \geq M + 1$.*

By Lemma 3(e), there exists a time slot $u < t_d - 1$ across which LAG increases to at least $M+1$. By Lemma 2, there is at least one hole in u. Thus, there exists a time slot t_h with $h \geq 1$ holes satisfying the following.

(A3) $0 \leq t_h < t_d - 1 \wedge\ LAG(\tau, t_h + 1) \geq M + 1 \wedge\ (\forall u : u \in [0, t_h] :: LAG(\tau, u) < M + 1)$.

In other words, t_h is the earliest time slot across which LAG increases to $M + 1$. In what follows, we derive an upper bound on the lags of all tasks in τ at $t_h + 1$ and prove that if (C) is satisfied, then their sum is strictly less than $M + 1$, contradicting the existence of t_h.

3.1 Categorization of Subtasks

In this subsection, we show how to categorize subtasks and bound their lags based on those categories.

***k*-dependent subtasks.** Subtasks of heavy tasks can be divided into "groups" based on their group deadlines in a straightforward manner: place all subtasks with identical group deadlines in the same *group* and identify the group using the smallest index of any subtask in that group. For

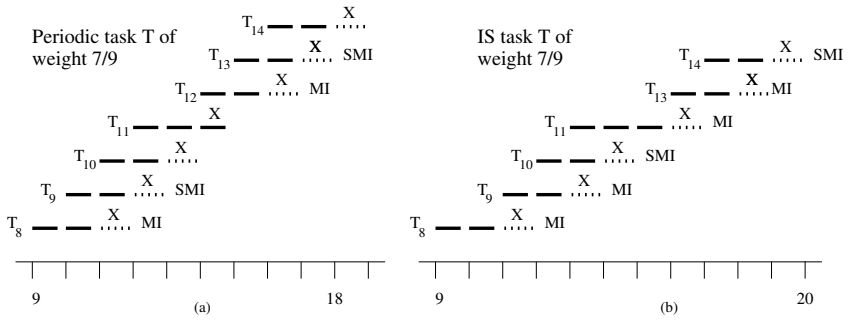

Figure 4. Possible schedules for the second job of **(a)** a periodic and **(b)** an IS task of weight 7/9 under EPDF. Subtasks are scheduled in the slots marked by an X. Solid (dotted) lines indicate slots that lie within (outside) the window of a subtask. A subtask scheduled in a dotted slot misses its deadline. In (a), T_8 and T_{12} are MIs, T_9 and T_{13} are SMIs, and the remaining subtasks fall within neither category. T_{10} and T_{14} have a tardiness of 1, and T_{11} has a tardiness of 0. In (b), T_8, T_9, T_{11}, and T_{13} are MIs, and T_{10} and T_{14} are SMIs. Note that T_8 and T_9 (T_{11} and T_{13}) belong to the same group G_8 (G_{11}). Thus, if there are IS separations, there may be more than one MI in a group.

example, in Fig. 1, $G_1 = \{T_1, T_2\}$, $G_3 = \{T_3, T_4, T_5\}$, and $G_6 = \{T_6, T_7, T_8\}$. If there are no IS or GIS separations among the subtasks of a group, then a deadline miss by one for a subtask T_i will necessarily result in a deadline miss by at least one for the remaining subtasks in T_i's group. Hence, a subtask T_j is dependent on all prior subtasks in its group for not missing its deadline. We say that T_j is k-*dependent*, where $k \geq 0$, if T is heavy and T_j is the $(k+1)^{st}$ subtask in its group (assuming all subtasks are present). If a task T is light, then we simply define all of its subtasks to be 0-dependent.

Miss initiators. We call a subtask missing its deadline at t by *one* a *miss initiator* (MI) for its group if no subtask of the same task is scheduled at $t-1$. Thus, a subtask is an MI if it misses its deadline and is either the first subtask in its group to do so or is separated from its predecessor by an IS or GIS separation. Such a subtask is termed a miss initiator because in the absence of future separations, it causes all subsequent subtasks in its group to miss their deadlines as well. $T_k \in G_i$ is an MI if $tardiness(T_k) = 1 \wedge S(T_k, t) = 1$, and $S(T_j, t-1) = 0$, for all $j < k$. Several examples of MIs are shown in Fig. 4

Successors of miss initiators. The immediate successor T_{i+1} of a miss-initiator subtask T_i is called a *successor of a miss initiator* (SMI) if $tardiness(T_{i+1}) = tardiness(T_i) = 1$ and $S(T_{i+1}, t) = 1 \Rightarrow S(T_i, t-1) = 1$. Fig. 4 shows several examples. Note that for T_{i+1} to be an SMI, its predecessor in S must be T_i, rather than some lower-indexed subtask of T.

The following lemma, proved in [5], bounds the lag of a task at time t, based on the k-dependency of its last-scheduled subtask.

Lemma 4 [**5**] *Let T_i be a k-dependent subtask of a GIS task T for $k \geq 0$, and let $d(T_i) < t_d$. Then $lag(T, d(T_i)+1) < (k+2) \cdot wt(T) - k$.*

The share that a GIS task receives in the ideal system may be zero during certain time slots, if subtasks are absent or are released late. We distinguish between tasks with and without subtasks at time t using the following definition of an *active* task.

Definition 3:[**7**] A task U is *active* at time t if it has a subtask U_j such that $e(U_j) \leq t < d(U_j)$.

Earlier, we showed how subtasks can be categorized. The following is a classification of *tasks* as given by Srinivasan and Anderson [7, 8].

A**:** Set of all tasks that are active and scheduled at t_h.

B**:** Set of all tasks that are active, but not scheduled at t_h.

I**:** Set of all tasks that are inactive at t_h.

A, B, and I form a partition of τ, *i.e.*,

$$A \cup B \cup I = \tau \text{ and } A \cap B = B \cap I = I \cap A = \emptyset. \quad (10)$$

We further classify tasks in A, based on the tardiness of their subtasks scheduled at t_h, as follows.

A_0**:** Includes T in A iff its subtask scheduled at t_h has zero tardiness.

A_1**:** Includes T in A iff its subtask scheduled at t_h has a tardiness of one.

A_1 is further partitioned into A_1^0, A_1^1, and A_1^2.

A_1^0**:** Includes T in A_1 iff its subtask scheduled at t_h is an MI.

A_1^1**:** Includes T in A_1 iff its subtask scheduled at t_h is an SMI.

A_1^2**:** Includes T in A_1 iff its subtask scheduled at t_h is neither an MI nor an SMI.

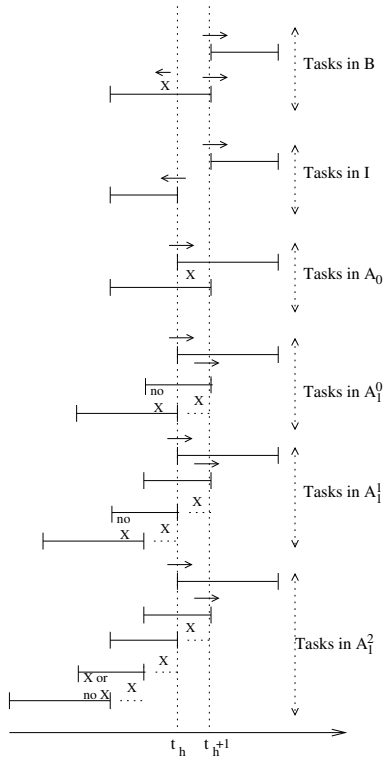

Figure 5. Task classification for Lemmas 6 – 11. The PF-windows of a sample task in each set are shown. An arrow over release (deadline) indicates that the release (deadline) could be anywhere in the direction of the arrow. An (no) X in a slot indicates that a subtask is (not) scheduled in that slot.

From the above, we have

$$A_0 \cup A_1 = A \text{ and } A_0^1 \cup A_1^1 \cup A_1^2 = A_1. \tag{11}$$

This classification of tasks is illustrated in Fig. 5. The cardinalities of the subsets of A are denoted as follows.

$$a_0 = |A_0|;\ a_1^0 = |A_1^0|;\ a_1^1 = |A_1^1|;\ a_1^2 = |A_1^2|. \tag{12}$$

The next lemma, proved in [5], shows that $a_0 \geq 1$.

Lemma 5 [**5**] *There exists a subtask W_l scheduled at t_h with $d(W_l) = t_h + 1$.*

The next six lemmas give bounds on the lags of tasks in A, B, and I at $t_h + 1$.

Lemma 6 [8] *For $T \in I, lag(I, t_h + 1) = 0$.*

Lemma 7 [8] *For $T \in B, lag(B, t_h + 1) \leq 0$.*

Lemma 8 *For $T \in A_0, lag(T, t_h + 1) < wt(T)$.*

Proof: Let T_i be the subtask of T scheduled at t_h. As shown in Fig. 5, the ideal system can be ahead of the actual system in executing T only by the amount of flow in T_{i+1}'s first slot. By parts (a) and (b) of Lemma 1, this flow is less than $wt(T)$. □

Lemma 9 *For $T \in A_1^0, lag(T, t_h + 1) < 2 \cdot wt(T)$.*

Proof: If $T \in A_1^0$, then the subtask T_i of T scheduled at t_h is an MI , and $d(T_i) = t_h$. If T_i is k-dependent, then by Lemma 4, $lag(T, t_h + 1)$ is less than $((k+2) \cdot wt(T) - k)$, which is at most $2 \cdot wt(T)$, for all $k \geq 0$. □

The next two lemmas follow similarly.

Lemma 10 *For $T \in A_1^1, lag(T, t_h + 1) < 3 \cdot wt(T) - 1$.*

Lemma 11 *For $T \in A_1^2, lag(T, t_h + 1) < 4 \cdot wt(T) - 2$.*

Finally, we show that $LAG(\tau, t_h + 1) < M + 1$ in each of the following cases.

Case A: $A_1 = \emptyset$.

Case B: $A_1^0 \neq \emptyset$.

Case C: $A_1^0 = \emptyset$ and $A_1^1 \neq \emptyset$.

Case D: $A_1^0 = A_1^1 = \emptyset$.

For each case above, $LAG(\tau, t_h + 1)$ can be expressed as follows. From (10), (11), and Lemmas 6 and 7, we have $LAG(\tau, t_h + 1) \leq \sum_{T \in A_0} lag(T, t_h + 1) + \sum_{T \in A_1^0} lag(T, t_h + 1) + \sum_{T \in A_1^1} lag(T, t_h + 1) + \sum_{T \in A_1^2} lag(T, t_h + 1)$, which by Lemmas 8– 11 implies that $LAG(\tau, t_h+1) < \sum_{T \in A_0} wt(T) + \sum_{T \in A_1^0} 2 \cdot wt(T) + \sum_{T \in A_1^1} (3 \cdot wt(T) - 1) + \sum_{T \in A_1^2} (4 \cdot wt(t) - 2)$. Letting wt denote the weight of the heaviest task, by (12), $LAG(\tau, t_h + 1)$ can be bounded as

$$\begin{aligned} LAG(\tau, t_h + 1) < a_0 \cdot wt + a_1^0 \cdot 2 \cdot wt \\ + a_1^1 \cdot (3 \cdot wt - 1) + a_1^2 \cdot (4 \cdot wt - 2). \end{aligned} \tag{13}$$

The total number of processors, M, expressed in terms of the number of subtasks in each subset of A scheduled at t_h, and the number of holes in t_h, is as follows.

$$M = a_0 + a_1^0 + a_1^1 + a_1^2 + h \tag{14}$$

Case A: $A_1 = \emptyset$**.** Case A is dealt with as follows.

Lemma 12 *If $A_1 = \emptyset$, then $LAG(\tau, t_h + 1) < M - 1$.*

Proof: If $A_1 = \emptyset$, then $a_1^0 = a_1^1 = a_1^2 = 0$. Therefore, by (13), $LAG(\tau, t_h + 1) < a_0 \cdot wt$, and by (14), $a_0 = M - h$. Hence, because $wt < 1$, $LAG(\tau, t_h + 1) < M - h$, which, because $h > 0$, implies that $LAG(\tau, t_h + 1) < M - 1$. □

Case B: $A_1^0 \neq \emptyset$**.** The following lemma, proved in [5], shows that if an MI is scheduled at t_h, then the total lag at $t_h + 1$ is less than $M + 1$.

Lemma 13 [**5**] *If $a_1^0 > 0$, then $LAG(\tau, t_h + 1) < M + 1$.*

Case C: $A_1^0 = \emptyset$**;** $A_1^1 \neq \emptyset$**.** The following lemma establishes the sufficiency of (C) for this case.

Lemma 14 *If* $A_1^0 = \emptyset$ *and* $A_1^1 \neq \emptyset$*, then* $LAG(\tau, t_h + 1) < M + 1$.

Proof: The proof is by contradiction. Assume to the contrary that $LAG(\tau, t_h+1) \geq M+1$. From (13) and $A_1^0 = \emptyset$, we have $LAG(\tau, t_h + 1) < a_0 \cdot wt + a_1^1 \cdot (3wt - 1) + a_1^2 \cdot (4wt - 2)$. Because $wt < 1$, we have $4wt - 2 < 3wt - 1$. Therefore, $LAG(\tau, t_h+1) < a_0 \cdot wt+(a_1^1+a_1^2)\cdot(3wt-1)$, which by (14) yields $LAG(\tau, t_h + 1) < a_0 \cdot wt + (M - h - a_0) \cdot (3wt - 1)$. If $LAG(\tau, t_h + 1) \geq M + 1$, then $a_0 \cdot wt+(M-h-a_0)\cdot(3wt-1) > M+1$, which implies that $wt > f = \frac{2M-h-a_0+1}{3M-3h-2a_0}$. By Lemma 5, $a_0 \geq 1$. It can be shown that f is minimized when $h = a_0 = 1$. Because $a_1^1 > 0$, this implies that $M \geq 3$, by (14). For $h = a_0 = 1$, $f = \frac{2M-1}{3M-5} > \frac{2}{3}$, for all $M \geq 2$. Thus, (C) is violated. □

Case D: $A_1^0 = A_1^1 = \emptyset$**.**

Lemma 15 *If* $A_1^0 = A_1^1 = \emptyset$*, then* $LAG(\tau, t_h + 1) < M + 1$.

Proof: The proof is again by contradiction. From (13) and $A_1^0 = A_1^1 = \emptyset$, $LAG(\tau, t_h + 1) < a_0 \cdot wt + (4wt - 2) \cdot a_1^2$, which, by (14), equals $a_0 \cdot wt + (4wt - 2) \cdot (M - h - a_0)$. If $LAG(\tau, t_h + 1) \geq M + 1$, then $a_0 \cdot wt + (4wt - 2) \cdot (M - h - a_0) > M + 1$, which in turn implies that $wt > f = \frac{3M-2h-2a_0+1}{4M-4h-3a_0}$. M, h, and a_0 are constrained by $M \geq h + a_0$, and $M, h, a_0 > 0$. It can be shown that the value of f is minimized when $h = a_0 = 1$, for which $f = \frac{3M-3}{4M-7} > \frac{3}{4}$, for all $M > 1$. This violates (C) (and (D)). □

By Lemmas 12–15, if (C) is satisfied, then $LAG(\tau, t_h + 1) < M + 1$, which is a contradiction to (A3). Thus, Theorem 2 is proved.

3.2 Other Results

(C) and (D) can be generalized by giving per-task weight restrictions of $\frac{1+q}{2+q}$ and $\frac{7+4q}{11+4q}$, respectively, for ensuring a tardiness of q, where $q \geq 1$. The proof for each is the same as before, except for generalizations to allow subtasks with tardiness up to q to be scheduled in any slot.

It can be shown that a tardiness of two less than the largest difference between successive group deadlines of any task can be ensured, in the absence of any restrictions. A formal proof is omitted due to space constraints. However, note that the key to the proof we have presented is dealing with the impact of cascades of deadline misses in heavy tasks. Such cascades must end by the next group deadline, regardless of any restrictions.

4 Conclusion

We have presented counterexamples that show that tardiness under the EPDF Pfair algorithm can exceed a small constant number of quanta for task systems that are not restricted, thereby proving false the conjecture that EPDF ensures a tardiness of one quantum. We have also presented sufficient weight restrictions that are more liberal than those previously known.

References

[1] J. Anderson and A. Srinivasan. Early-release fair scheduling. In *Proc. of the 12th Euromicro Conference on Real-Time Systems*, pages 35–43, June 2000.

[2] J. Anderson and A. Srinivasan. Pfair scheduling: Beyond periodic task systems. In *Proc. of the 7th International Conference on Real-Time Computing Systems and Applications*, pages 297–306, Dec. 2000.

[3] J. Anderson and A. Srinivasan. Mixed Pfair/ERfair scheduling of asynchronous periodic tasks. *Journal of Computer and System Sciences*, 68(1):157–204, 2004.

[4] S. Baruah, N. Cohen, C.G. Plaxton, and D. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15:600–625, 1996.

[5] U. Devi and J. Anderson. Improved conditions for bounded tardiness under EPDF fair multiprocessor scheduling (full paper). Available at http://www.cs.unc.edu/~anderson/papers.html, Nov. 2003.

[6] A. Srinivasan. *Efficient and Flexible Fair Scheduling of Real-time Tasks on Multiprocessors*. PhD thesis, University of North Carolina at Chapel Hill, Dec. 2003.

[7] A. Srinivasan and J. Anderson. Optimal rate-based scheduling on multiprocessors. In *Proc. of the 34th ACM Symposium on Theory of Computing*, pages 189–198, May 2002.

[8] A. Srinivasan and J. Anderson. Efficient scheduling of soft real-time applications on multiprocessors. In *Proc. of the 15th Euromicro Conference on Real-time Systems*, pages 51–59, July 2003.