# Fair Scheduling of Dynamic Task Systems on Multiprocessors ⋆

Anand Srinivsan and James H. Anderson

*Department of Computer Science, University of North Carolina*
*Chapel Hill, NC 27599-3175 USA*

**Abstract**

In dynamic real-time task systems, tasks that are subject to deadlines are allowed to join and leave the system. In previous work, Stoica *et al.* and Baruah *et al.* presented conditions under which such joins and leaves may occur in fair-scheduled uniprocessor systems without causing missed deadlines. In this paper, we extend their work by considering fair-scheduled multiprocessors. We show that their conditions are sufficient on $M$ processors, under any deadline-based Pfair scheduling algorithm, if the utilization of every subset of $M - 1$ tasks is at most one. Further, for the general case in which task utilizations are not restricted in this way, we derive sufficient join/leave conditions for the PD$^2$ Pfair algorithm. We also show that, in general, these conditions cannot be improved upon without causing missed deadlines.

*Key words:* Dynamic task systems, Pfairness, multiprocessor, real-time scheduling

## 1 Introduction

In many real-time systems, the set of runnable tasks may change dynamically. For example, in an embedded system, different modes of operation may need to be supported; a mode change may require adding new tasks and deleting existing tasks. Another example is a desktop system that supports real-time applications such as multimedia and collaborative-support systems, which may be initiated at arbitrary times.

⋆ Work supported by NSF grants CCR 9972211, CCR 9988327, ITR 0082866, and CCR 0204312.

The distinguishing characteristic of *dynamic* task systems such as these is that tasks are allowed to *join* and *leave* the system. If such joins and leaves are unrestricted, then the system may become overloaded, and deadlines may be missed. Thus, joins and leaves must be performed only under conditions that ensure that deadline guarantees are not compromised. A suitable join condition usually can be obtained from the feasibility test associated with the scheduling algorithm being used. A leave condition is somewhat trickier. In particular, if an "over-allocated" task is allowed to leave, then it might re-join immediately, and thus effectively execute at a higher-than-prescribed rate.

In this paper, we consider the problem of scheduling such task systems on multiprocessors. This problem has been studied earlier in the context of uniprocessor static-priority [9,13] and fair-allocation schemes [5,12]. Our focus is fair scheduling because it is the only known way of optimally scheduling recurrent real-time tasks on multiprocessors [1,3,4,11]. In addition, practical interest in multiprocessor fair scheduling algorithms is growing. For example, Ensim Corp., an Internet service provider, has deployed such algorithms in its product line [8]. The need to support dynamic tasks is fundamental in this setting.

In fair scheduling disciplines, tasks are required to make progress at steady rates. Steady allocation rates are ensured by scheduling in a manner that closely tracks an ideal, fluid allocation. The *lag* of a task measures the difference between its ideal and actual allocations. In fair scheduling schemes, lags are required to remain bounded. (Such a bound in turn implies a bound on the timeliness of real-time tasks.) If a task's lag is positive, then it has been under-allocated; if negative, then it has been over-allocated. In the uniprocessor join/leave conditions presented previously [5,12], a task is allowed to leave iff it is not over-allocated, and join iff the total utilization after it joins is at most one.

Extending the above-mentioned work to multiprocessors is not straightforward; in fact, Baruah *et al.* explicitly noted the multiprocessor case as an open problem [5]. In recent work, dynamic multiprocessor systems were considered by Chandra *et al.* [6,7]. However, their work was entirely experimental in nature, with no formal analysis of the algorithms considered. In this paper, we present join/leave conditions for which such analysis is provided. Before presenting a more detailed overview of the contributions of this paper, we briefly describe some of the fair scheduling concepts used in this paper.

**Pfair scheduling.** The *periodic* task model provides the simplest notion of a recurrent real-time task. In this model, successive job releases (*i.e.*, invocations) by the same task are spaced apart by a fixed interval, called the task's *period*. Periodic tasks can be optimally scheduled on multiprocessors using *Pfair* scheduling algorithms [1,3,4]. Pfairness requires the lag of each task to be bounded between $-1$ and 1, which is a stronger requirement than

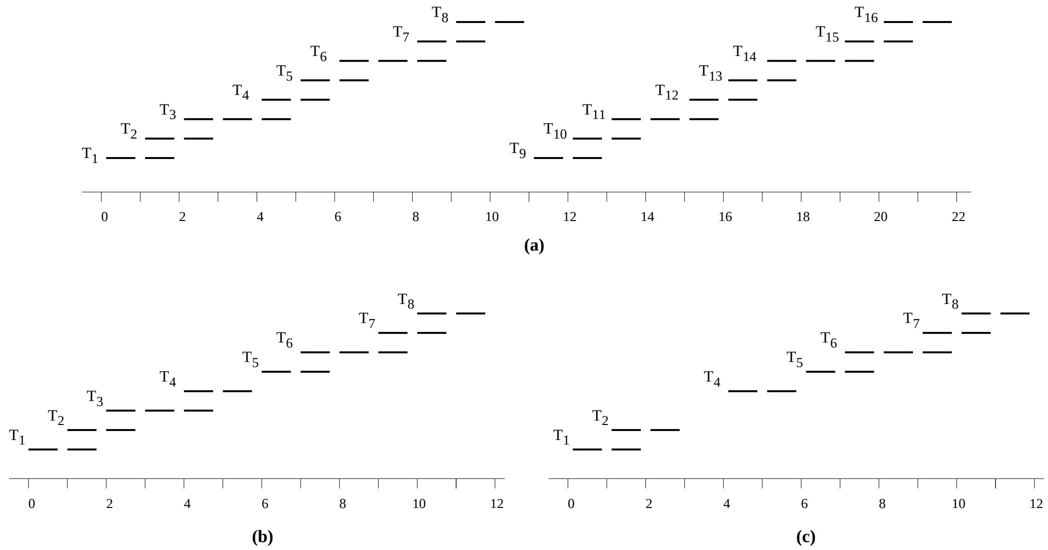


Fig. 1. **(a)** Pfair windows of the first two jobs (or sixteen subtasks) of a periodic task $T$ of weight $8/11$. Each of these subtasks must be scheduled during its window, or a lag-bound violation will result. **(b)** The first eight Pfair windows of an IS task. Subtask $T_5$ becomes eligible one time unit late. **(c)** The Pfair windows of a GIS task. Subtask $T_3$ is absent and $T_5$ is one time unit late. (Because $T_3$ is absent, this is not an IS task.)

periodicity. As we shall see, these lag bounds have the effect of breaking each task into quantum-length *subtasks* that must be scheduled within *windows* of approximately equal lengths. The length and alignment of a task's windows are determined by its weight. The *weight* or *utilization* of a task is the ratio of its per-job execution cost and period; a task's weight determines the processor share it requires. Fig. 1(a) shows the subtasks and windows for the first two jobs of a periodic task $T$ with an execution requirement of 8 and a period of 11 (*i.e.*, of weight $8/11$).

In the *sporadic* model, the periodic notion of recurrence is relaxed by specifying a minimum (rather than exact) spacing between consecutive job releases of the same task. In recent work [2,11], we extended the sporadic model to obtain the *intra-sporadic* (IS) and *generalized intra-sporadic* (GIS) models. The sporadic model allows *jobs* to be released "late"; the IS model allows *subtasks* to be released late, as illustrated in Fig. 1(b). The GIS model is obtained from the IS model by allowing subtasks to be absent. Fig. 1(c) shows an example.

In [11], we showed that the PD$^2$ Pfair algorithm optimally schedules static GIS task systems on multiprocessors. In [2], we proved that the (simpler) earliest-pseudo-deadline-first (EPDF) algorithm is optimal for scheduling static IS task systems on two processors. (PD$^2$ and EPDF are described in Sec. 2.2.)

**Contributions.** In this paper, we extend our earlier work, as well as prior work on uniprocessor fairness, in several significant ways. First, we show that the previously-presented uniprocessor join/leave conditions [5,12] are insufficient for avoiding deadline misses when tasks are scheduled using any of a class of algorithms that includes all known (dynamic-priority) Pfair scheduling algorithms. Second, we show that these uniprocessor conditions are sufficient when using any deadline-based algorithm, if the total weight of any subset of $M-1$ tasks is at most one at all times. This result extends our earlier result on the optimality of EPDF for two-processor systems [2]. Third, we derive sufficient conditions (that are tight) for the general case in which task weights are not restricted as above, and PD$^2$ is used for scheduling.

**Overview.** The rest of the paper is organized as follows. In Sec. 2, needed definitions are given. In Sec. 3, our join/leave conditions are stated. Results pertaining to the EPDF and PD$^2$ algorithms are then presented in Secs. 4 and 5, respectively. We conclude in Sec. 6.

## 2 Preliminaries

In the following subsections, relevant concepts and terms are defined. We begin with Pfair scheduling.

### *2.1 Pfair scheduling*

In defining notions relevant to Pfair scheduling, we limit attention (for now) to periodic tasks; we assume that each such task releases its first job at time 0. A periodic task $T$ with an integer *period* $T.p$ and an integer per-job *execution cost* $T.e$ has a *weight* $wt(T) = T.e/T.p$, where $0 < wt(T) \leq 1$. Such a task $T$ is *light* if $wt(T) < 1/2$, and *heavy* otherwise.

Under Pfair scheduling, processor time is allocated in discrete time units, called *quanta*; the time interval $[t, t+1)$, where $t$ is a nonnegative integer, is called *slot* $t$. (Hence, time $t$ refers to the beginning of slot $t$.) In each slot, each processor can be allocated to at most one task. 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 $\tau$ is a set of tasks and $\mathcal{N}$ is the set of nonnegative integers. $S(T, t) = 1$ iff $T$ is scheduled in slot $t$. Thus, in any $M$-processor schedule, $\sum_{T \in \tau} S(T, t) \leq M$ for all $t$.

The notion of a Pfair schedule is defined by comparing such a schedule to a fluid processor-sharing schedule that allocates $wt(T)$ processor time to task $T$ in each slot. Deviation from the fluid schedule is formally captured by the concept of *lag*. The *lag of task $T$ at time $t$* $lag(T,t)$ is defined as $wt(T)\cdot t - \sum_{u=0}^{t-1} S(T,u)$. A schedule is *Pfair* iff

$$(\forall T, t :: -1 < lag(T,t) < 1). \tag{1}$$

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

The lag bounds above have the effect of breaking each task $T$ into an infinite sequence of *unit-time subtasks*. We denote the $i^{th}$ subtask of task $T$ as $T_i$, where $i \geq 1$. As in [3], we associate a *pseudo-release* $r(T_i)$ and *pseudo-deadline* $d(T_i)$ with each subtask $T_i$, as follows. (For brevity, we often drop the prefix "pseudo-.")

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

$T_i$ must be scheduled in the interval $w(T_i) = [r(T_i), d(T_i))$, termed its *window*, or (1) will be violated. Note that $r(T_{i+1})$ is either $d(T_i) - 1$ or $d(T_i)$. Thus, consecutive windows of the same task either overlap by one slot or are disjoint (see Fig. 1(a)). The *length* of $T_i$'s window, denoted $|w(T_i)|$, is $d(T_i) - r(T_i)$. As an example, consider subtask $T_2$ in Fig. 1(a). Here, we have $r(T_2) = 1$, $d(T_2) = 3$, and $|w(T_2)| = 2$. Therefore, $T_2$ must be scheduled in either slot 1 or 2. (If $T_1$ is scheduled in slot 1, then $T_2$ must be scheduled in slot 2.)

### *2.2 Scheduling Algorithms*

In earlier work [1,2], we proved that the earliest-pseudo-deadline-first (EPDF) Pfair algorithm is optimal on at most two processors, but not on more than two processors. As its name suggests, EPDF gives higher priority to subtasks with earlier deadlines. A tie between subtasks with equal deadlines is broken arbitrarily.

At present, three Pfair scheduling algorithms are known to be optimal on an arbitrary number of processors: PF [3], PD [4], and PD$^2$ [1]. These algorithms prioritize subtasks on an EPDF basis, but differ in the choice of tie-breaking rules. PD$^2$, which is the most efficient of the three, uses two tie-break parameters. (Scheduling decisions under PD$^2$ can be implemented in $O(M \log N)$ time, where $M$ is the number of processors and $N$ is the number of tasks.)

The first PD$^2$ tie-break is a bit, denoted by $b(T_i)$. As mentioned earlier, consecutive windows of a task are either disjoint or overlap by one slot. $b(T_i)$

distinguishes between these two possibilities.

$$b(T_i) = \left\lceil \frac{i}{wt(T)} \right\rceil - \left\lfloor \frac{i}{wt(T)} \right\rfloor \tag{3}$$

For example, in Fig. 1(a), $b(T_i) = 1$ for $1 \leq i \leq 7$ and $b(T_8) = 0$. PD$^2$ favors a subtask with a $b$-bit of 1 over one with a $b$-bit of 0. Informally, it is better to execute $T_i$ "early" if its window overlaps that of $T_{i+1}$, because this potentially leaves more slots available to $T_{i+1}$.

The second PD$^2$ tie-break, the *group deadline*, is needed in systems containing tasks with windows of length two. A task $T$ has such windows iff $1/2 \leq wt(T) < 1$. Consider a sequence $T_i, \ldots, T_j$ of subtasks of such a task $T$ such that $b(T_k) = 1 \ \wedge \ |w(T_{k+1})| = 2$ for all $i \leq k < j$. 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. The group deadline of a subtask $T_i$, denoted $D(T_i)$, is the earliest time by which such a "cascade" must end. Formally, it is the earliest time $t$, where $t \geq d(T_i)$, such that either $(t = d(T_k) \ \wedge \ b(T_k) = 0)$ or $(t + 1 = d(T_k) \ \wedge \ |w(T_k)| = 3)$ for some subtask $T_k$. For example, in Fig. 1(a), $D(T_3) = d(T_6) - 1 = 8$ and $D(T_7) = d(T_8) = 11$. PD$^2$ favors subtasks with later group deadlines because scheduling them later can lead to longer cascades, which places more constraints on the future schedule.

We can now describe the PD$^2$ priority definition. If subtasks $T_i$ and $U_j$ are both eligible at time $t$, then PD$^2$ prioritizes $T_i$ over $U_j$ at $t$ if $(d(T_i) < d(U_j))$, or $(d(T_i) = d(U_j) \ \wedge \ b(T_i) = 1 \ \wedge \ b(U_j) = 0)$, or $(d(T_i) = d(U_j) \ \wedge \ b(T_i) = b(U_j) = 1 \ \wedge \ D(T_i) > D(U_j))$. (Refer to [1] for a more detailed explanation.)

### *2.3 Generalized Intra-sporadic Tasks*

Having described the concept of Pfair scheduling, we now describe the intra-sporadic (IS) and the generalized intra-sporadic (GIS) task models.

The IS model generalizes the sporadic model by allowing separation between consecutive subtasks of a task. 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 is the separation if $T$ were periodic (refer to (2)). Thus, an IS task is obtained by allowing a task's windows to be right-shifted from where they would appear if the task were periodic. Fig. 1(b) illustrates this.

Each subtask of an IS task has an *offset* that gives the amount by which its window has been right-shifted. Let $\theta(T_i)$ denote the offset of subtask $T_i$. Then, by (2), we have the following.

$$r(T_i) = \theta(T_i) + \left\lfloor \frac{i-1}{wt(T)} \right\rfloor \tag{4}$$

$$d(T_i) = \theta(T_i) + \left\lceil \frac{i}{wt(T)} \right\rceil \tag{5}$$

These 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, the offsets satisfy the following property.

$$k \geq i \Rightarrow \theta(T_k) \geq \theta(T_i) \tag{6}$$

Each subtask $T_i$ has an additional parameter $e(T_i)$ that corresponds to the first time slot in which $T_i$ is eligible to be scheduled. It is assumed that $e(T_i) \leq r(T_i)$ and $e(T_i) \leq e(T_{i+1})$ for all $i \geq 1$. Allowing $e(T_i)$ to be less than $r(T_i)$ is equivalent to allowing "early" subtask releases as in ERfair scheduling [1]. The interval $[r(T_i), d(T_i))$ is called the *PF-window* of $T_i$, while the interval $[e(T_i), d(T_i))$ is called the *IS-window* of $T_i$.

The GIS model generalizes the IS model by allowing subtasks to be absent. Thus, the subtasks of a GIS task are a subset of the subtasks of an IS task. Fig. 1(c) shows an example. The formulae for subtask release times and deadlines of a GIS task are the same as for an IS task. The $b$-bit and group deadline for a subtask are defined as before assuming that all future subtask releases of that task are as early as possible. Thus, $D(T_i) = \theta(T_i) + D_p(T_i)$, where $D_p(T_i)$ is $T_i$'s group deadline if $T$ were periodic.

Because the GIS model generalizes the other task models above, it is the notion of recurrence considered hereafter. We now present some definitions and properties about GIS task systems.

**Terminology.** An *instance* of a task system is obtained by specifying a unique assignment of release times and eligibility times for each subtask, subject to (6). Note that the deadline of a subtask is automatically determined once its release time is fixed (refer to (4) and (5)). 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 $T_k$ (*e.g.*, $T_4$ is $T_2$'s successor in Fig. 1(c)).

**Feasibility.** In [2,11], we showed that a GIS task system $\tau$ is feasible on $M$ processors iff

$$\sum_{T \in \tau} wt(T) \leq M. \tag{7}$$

In fact, the proof of this shows that a schedule exists in which each subtask is scheduled in its PF-window. In [11], we also proved that $\text{PD}^2$ correctly schedules any static GIS task system that satisfies (7).

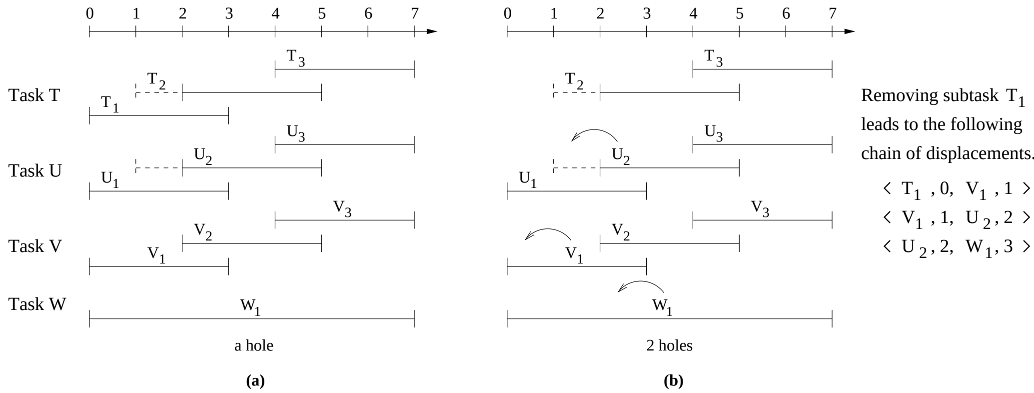


Fig. 2. A schedule for three tasks of weight 3/7 and one task of weight 1/7 on two processors. Solid lines depict PF-windows; dashed lines are used to show the extent to which an IS-window extends before a corresponding PF-window. Note that only subtasks $T_2$ and $U_2$ are eligible before their PF-windows. The schedule in inset (a) is depicted by showing which subtasks are scheduled in each slot (*e.g.*, $W_1$ is scheduled in slot 3). Inset (b) illustrates the displacements caused by the removal of $T_1$ from the schedule shown in inset (a).

**Displacements.** By definition, the removal of a subtask from one instance of a GIS task system results in another valid instance. Let $X^{(i)}$ denote a subtask of any task in a GIS task system $\tau$. Let $S$ denote any schedule of $\tau$ obtained by an EPDF-based algorithm. Assume that removing $X^{(1)}$ scheduled at slot $t_1$ in $S$ causes $X^{(2)}$ to shift from slot $t_2$ to $t_1$, where $t_1 \neq t_2$, which in turn may cause other shifts. We call this shift a *displacement* and represent it by a four-tuple $\langle X^{(1)}, t_1, X^{(2)}, t_2 \rangle$. A displacement $\langle X^{(1)}, t_1, X^{(2)}, t_2 \rangle$ is *valid* iff $e(X^{(2)}) \leq t_1$. Because there can be a cascade of shifts, we may have a *chain* of displacements, as illustrated in Fig. 2.

Removing a subtask may also lead to slots in which some processors are idle. In a schedule $S$, if $k$ processors are idle in slot $t$, then we say that there are $k$ *holes* in $S$ in slot $t$. Note that holes may exist because of late subtask releases, even if total utilization is $M$.

The lemmas below concern displacements and holes. The first two were proved earlier for PD$^2$ [11] but apply to all algorithms that prioritize subtasks on an EPDF basis. Lemma 1 states that a subtask removal can only cause left-shifts, as in Fig. 2(b). Lemma 2 indicates when a left-shift into a slot with a hole can occur. Lemma 3 shows that shifts across a hole cannot occur. Here, $\tau$ is an instance of a GIS task system and $S$ denotes a schedule for $\tau$ obtained by an EPDF-based algorithm. Throughout this paper, we assume that ties among subtasks are resolved consistently, *i.e.*, if $\tau'$ is obtained from $\tau$ by a subtask removal, then the relative priorities of two subtasks in $\tau'$ are the same as in $\tau$.

**Lemma 1** *Let $X^{(1)}$ be a subtask that is removed from $\tau$, and let the resulting chain of displacements in $S$ be $C = \Delta_1, \Delta_2, \ldots, \Delta_k$, where $\Delta_i = \langle X^{(i)}, t_i,$*

$X^{(i+1)}, t_{i+1}\rangle$. *Then* $t_{i+1} > t_i$ *for all* $i \in \{1, \ldots, k\}$.

**Lemma 2** *Let* $\Delta = \langle X^{(1)}, t_1, X^{(2)}, t_2\rangle$ *be a valid displacement in* $S$. *If* $t_1 < t_2$ *and there is a hole in slot* $t_1$ *in* $S$, *then* $X^{(2)}$ *is the successor of* $X^{(1)}$.

**Lemma 3** *Let* $\Delta = \langle X^{(1)}, t_1, X^{(2)}, t_2\rangle$ *be a valid displacement in* $S$. *If* $t_1 < t_2$ *and there is a hole in slot* $t'$ *such that* $t_1 \le t' < t_2$ *in* $S$, *then* $t' = t_1$ *and* $X^{(2)}$ *is the successor of* $X^{(1)}$.

**Proof (of Lemma 3):** Since $\Delta$ is valid, $e(X^{(2)}) \le t_1$. If $t_1 < t'$, then $e(X^{(2)}) < t'$, implying that $X^{(2)}$ is not scheduled in slot $t_2 > t'$, as assumed, since there is a hole in $t'$. Thus, $t_1 = t'$; by Lemma 2, $X^{(2)}$ is the successor of $X^{(1)}$. $\square$

**Flows and lags in GIS task systems.** The lag of a GIS task is defined in the same way as it is defined for periodic tasks. Let $ideal(T, t)$ denote the share that $T$ receives in a fluid schedule in $[0, t)$. Then,

$$lag(T, t) = ideal(T, t) - \sum_{u=0}^{t-1} S(T, u). \tag{8}$$

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

$$f(T_i, u) = \begin{cases} \left(\left\lfloor \frac{i-1}{wt(T)} \right\rfloor + 1\right) \cdot wt(T) - (i-1), & \text{if } u = r(T_i) \\ i - \left(\left\lceil \frac{i}{wt(T)} \right\rceil - 1\right) \cdot wt(T), & \text{if } u = d(T_i) - 1 \\ wt(T), & \text{if } r(T_i) < u < d(T_i) - 1 \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

(Note that $f(T_i, u)$ is 0 if $u$ does not lie in $T_i$'s window.) Fig. 3 shows the values of $f$ for different subtasks of a task of weight $5/16$. $flow(T, u)$ is simply defined as $flow(T, u) = \sum_i f(T_i, u)$. Observe that $flow(T, u)$ usually equals $wt(T)$, but in certain slots, it may be less than $wt(T)$, so that each subtask of $T$ has a unit share. Using (9), we can obtain the following flow properties. (These are proved in [11].)

**(F1)** For all time slots $t$, $flow(T, t) \le wt(T)$.

**(F2)** Let $T_i$ be a subtask of a GIS task and let $T_k$ be its successor. If $b(T_i) = 1$ and $r(T_k) \ge d(T_i)$, then $flow(T, d(T_i) - 1) + flow(T, d(T_i)) \le wt(T)$.

For example, in Fig. 3(b), $flow(T, 3) + flow(T, 4) = 1/16 < 5/16$ and $flow(T, 14) + flow(T, 15) = 5/16$.

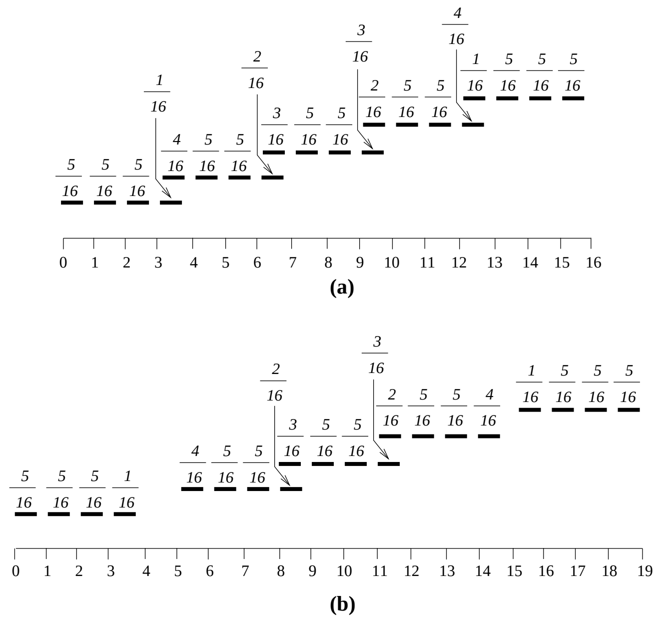


Fig. 3. Fluid schedule for a task $T$ of weight 5/16. The share of each subtask in the slots of its window is shown. In **(a)**, no subtask is released late; in **(b)**, $T_2$ and $T_5$ are released late. Note that $flow(T, 3)$ is either 5/16 or 1/16 depending on when subtask $T_2$ is released.

Given the above flow values, $ideal(T, t)$ is defined as $\sum_{u=0}^{t-1} \mathit{flow}(T, u)$. Hence, by (8), we obtain that $lag(T, t+1) = \sum_{u=0}^{t} (\mathit{flow}(T, u) - S(T, u)) = lag(T, t) + \mathit{flow}(T, t) - S(T, t)$. Similarly, the total lag for a schedule $S$ and task system $\tau$ at time $t + 1$, denoted by $LAG(\tau, t + 1)$, is defined as follows.

$$LAG(\tau, t + 1) = LAG(\tau, t) + \sum_{T \in \tau} (\mathit{flow}(T, t) - S(T, t)). \tag{10}$$

($LAG(\tau, 0)$ is defined to be 0.) The lemma below is used in our proofs.

**Lemma 4** *If $LAG(\tau, t) < LAG(\tau, t + 1)$, then there is a hole in slot $t$.*

**Proof:** Suppose there is no hole in slot $t$. Then, $\sum_{T \in \tau} S(T, t) = M$. On the other hand, by (F1) and (7), $\sum_{T \in \tau} flow(T, t) \le M$. Therefore, by (10), $LAG$ cannot increase from $t$ to $t + 1$. $\square$

## 3 Dynamic Task Systems

Prior work in the real-time-sytems literature has focused mostly on static systems, in which the set of tasks does not change with time. However, systems exist in which the set of tasks may change frequently. One example of such a system is a virtual-reality application in which the user moves within a virtual environment. As the user moves and the virtual scene changes, the time required to render the scene may vary substantially. If a single task is responsible for rendering, then its weight may change frequently. Task *reweighting* can be modeled as a leave-and-join problem, in which a task with the old weight leaves and a task with the new weight joins.

As shown in [2,11], a valid schedule can be obtained for any *static* GIS task system satisfying (7) by constructing a flow network with a real-valued flow based on the *flow* values defined in Sec. 2 and illustrated in Fig. 3. A corresponding integral flow exists because all edge capacities in the network are integers. This gives us a Pfair schedule. This argument can be easily extended to apply to any *dynamic* task system for which the total utilization of all tasks present at every instant is at most $M$. This proof produces an offline schedule in which *each subtask is scheduled in its PF-window*. (The schedule is offline because all subtask release times must be known beforehand.)

A condition for allowing tasks to join the system is an immediate consequence of this feasibility test, *i.e.*, admit a task if the total utilization is at most $M$ after its admission. The important question left is: *when should a task be allowed to leave the system*? (Here, we are referring to the time when we can reclaim the utilization of the task. The task may actually be allowed to leave the system earlier.) As shown in [5,12], if an over-allocated task is allowed to leave, then it can re-join immediately and effectively execute at a rate higher than its specified rate causing other tasks to miss their deadlines. Hence, we only allow non-over-allocated tasks (*i.e.*, tasks with non-negative lags) to leave the system, as stated in (C1) below.

**(C1)** *Join condition*: A task $T$ can join at time $t$ iff the total utilization after joining is at most $M$. If $T$ joins at time $t$, then $\theta(T_1)$ is set to $t$.
*Leave condition*: A task $T$ can leave at time $t$ iff $t \geq d(T_i)$, where $T_i$ is the last-released subtask of $T$.

The condition $t \geq d(T_i)$ implies that $lag(T, t) = 0$. To see why, note that since $T_i$ is the last-released subtask of $T$, $T$ is neither under-allocated nor over-allocated at time $d(T_i)$. Thus, only tasks with zero lag are allowed to leave the system. It is easy to extend (C1) to allow a task with positive lag to leave. This is because such a task is under-allocated, and hence its last-released subtask has not yet been scheduled. Intuitively, not scheduling a subtask is equivalent to removing it, and by Lemma 1, the removal of a subtask cannot lead to a missed deadline. Thus, we can allow task $T$ to leave the system if (C1) is satisfied by the last-*scheduled* subtask of $T$. However, for simplicity, we assume (C1) as stated above.

(C1) is a direct extension of the uniprocessor conditions presented by Baruah

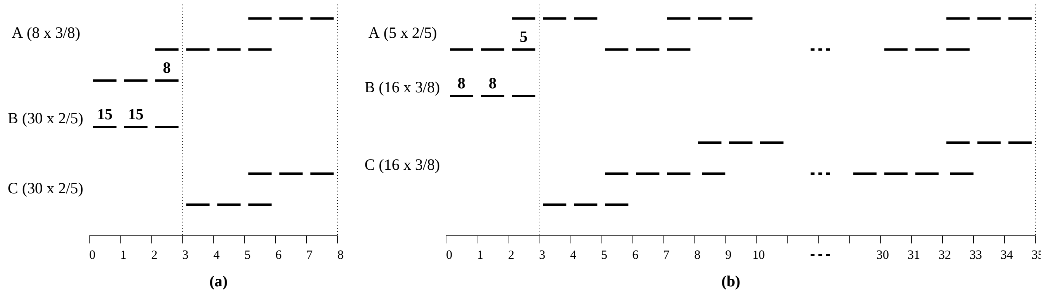


Fig. 4. Counterexamples demonstrating insufficiency of (C1). An integer value $n$ in slot $t$ of some window means that $n$ of the subtasks that must execute within that window are scheduled in slot $t$. No integer value means that no such subtask is scheduled in slot $t$. The vertical lines depict intervals with excess demand. **(a)** Theorem 1. Case 1: Tasks of weight $2/5$ are favored at times 0 and 1. **(b)** Theorem 1. Case 2: Tasks of weight $3/8$ are favored at times 0 and 1.

*et al.* [5] and Stoica *et al.* [12]. However, as shown below, it is not sufficient on multiprocessors. The theorem below applies to any "weight-consistent" Pfair scheduling algorithm. An algorithm is *weight-consistent* if, given two tasks $T$ and $U$ of equal weight with eligible subtasks $T_i$ and $U_i$, respectively, where $r(T_i) = r(U_i)$ (and hence, $d(T_i) = d(U_i)$), $T_i$ has priority over a third subtask $V_k$ iff $U_i$ does. All known Pfair scheduling algorithms are weight-consistent.

**Theorem 1** *No weight-consistent scheduler can guarantee all deadlines on multiprocessors under* (C1)*.*

**Proof:** Consider a class of task systems consisting of two sets of tasks $X$ and $Y$ of weights $w_1 = 2/5$ and $w_2 = 3/8$, respectively. Let $X_f$ ($Y_f$) denote the set of first subtasks of tasks in $X$ ($Y$). We construct a task system depending on the task weight favored by the scheduler. We say that $X_f$ *is favored* (analogously for $Y_f$) if, whenever subtasks in $X_f$ and $Y_f$ are released at the same time, those in $X_f$ are favored.

**Case 1: $X_f$ is favored.** Consider a dynamic task system consisting of the following types of tasks to be scheduled on 15 processors. (In each of our counterexamples, no subtask is eligible before its PF-window.)

**Type A:** 8 tasks of weight $w_2$ that join at time 0.
**Type B:** 30 tasks of weight $w_1$ that join at time 0 and leave at time 3; each releases one subtask.
**Type C:** 30 tasks of weight $w_1$ that join at time 3.

Because $30w_1 + 8w_2 = 15$, this task system is feasible, and the join condition for type-C tasks in (C1) is satisfied. Note that $d(T_1) = \left\lceil \frac{5}{2} \right\rceil = 3$ for every type-B task $T$; hence, the leave condition in (C1) is also satisfied.

Since subtasks in $X_f$ are favored, type-B tasks are favored over type-A tasks at times 0 and 1. Hence, the schedule for $[0, 3)$ will be as shown in Fig. 4(a). Consider the interval $[3, 8)$. Each type-A task has two subtasks remaining for execution, which implies that the type-A tasks need 16 quanta. Similarly, each type-C task also has two subtasks, which implies that the type-C tasks need 60 quanta. However, the total number of quanta in $[3, 8)$ is $15 \cdot (8 - 3) = 75$. Thus, one subtask will miss its deadline at or before time 8.

**Case 2: $Y_f$ is favored.** Consider a dynamic task system consisting of the following types of tasks to be scheduled on 8 processors.

**Type A:** 5 tasks of weight $w_1$ that join at time 0.
**Type B:** 16 tasks of weight $w_2$ that join at time 0 and leave at time 3; each releases one subtask.
**Type C:** 16 tasks of weight $w_2$ that join at time 3.

As in Case 1, we can show that (C1) is satisfied at time 3. Since subtasks in $Y_f$ are favored, type-B tasks are favored over type-A tasks at times 0 and 1. Hence, the schedule for $[0, 3)$ will be as shown in Fig. 4(b). Consider the interval $[3, 35)$. The number of subtasks of each type-A task that need to be executed in $[3, 35)$ is $1 + (35 - 5) \cdot 2/5 = 13$. Similarly, the number of subtasks of each type-C task is $(35 - 3) \cdot 3/8 = 12$. The total is $5 \cdot 13 + 16 \cdot 12 = 257$, whereas the number of quanta in $[3, 35)$ is $(35 - 3) \cdot 8 = 256$. Thus, one subtask will miss its deadline at or before time 35. □

Theorem 1 can be "circumvented" if it can be known at the time a subtask is *released* whether it is the final subtask of its task. For example, in Fig. 4(a), if we knew that the first subtask $T_1$ of each type-B task is its last, then we could have given $T_1$ an effective $b$-bit of zero. Hence, PD$^2$ would have scheduled it with a lower priority than any type-A task. However, in general, such knowledge may not be available to the scheduler.

The examples in Fig. 4 show that allowing a light task $T$ to leave at $d(T_i)$ when $b(T_i) = 1$ can lead to missed deadlines. We now derive a similar, but stronger, condition for heavy tasks, when PD$^2$ is used for scheduling.

**Theorem 2** *If a heavy task $T$ is allowed to leave before $D(T_i)$, where $T_i$ is the last-released subtask of $T$, then there exist task systems that miss a deadline under* PD$^2$.

**Proof:** Consider the following dynamic task system to be scheduled on 35 processors, where $2 \le t \le 4$.

**Type A:** 9 tasks of weight 7/9 that join at time 0.
**Type B:** 35 tasks of weight 4/5 that join at time 0 and leave at time $t$; each releases one subtask.

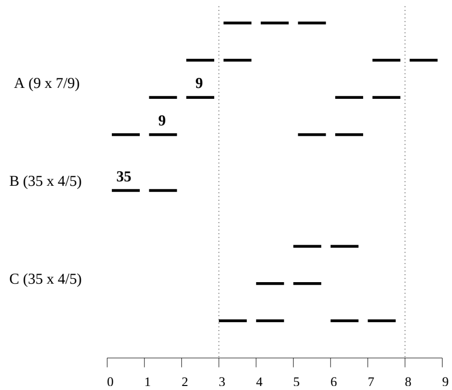


Fig. 5. Theorem 2. (The notation used in this figure is similar to that in Fig. 4.)

**Type C:** 35 tasks of weight 4/5 that join at time $t$.

All type-A and type-B tasks have the same PD$^2$ priority at time 0, because each has a deadline at time 2, a $b$-bit of 1, and a group deadline at time 5. Hence, the type-$B$ tasks may be given higher priority. Assuming this, Fig. 5 depicts the schedule for the case of $t = 3$. (This theorem applies to PF and PD as well, because both favor the type-$B$ tasks at time 0.)

Consider the interval $[t, t+5)$. Each type-A and type-C task has four subtasks with deadlines in $[t, t+5)$ (see Fig. 5). Thus, $9 \cdot 4 + 35 \cdot 4 = 35 \cdot 5 + 1$ subtasks must be executed in $[t, t+5)$. Since only $35 \cdot 5$ quanta are available in $[t, t+5)$, one subtask will miss its deadline. $\square$

Although (C1) is not sufficient in general, in Sec. 4, we show that it is sufficient for a restricted class of task systems, even under EPDF. Theorems 1 and 2 suggest the following new conditions, which are sufficient when used with PD$^2$ (proved in Sec. 5). Theorems 1 and 2 also show that these conditions are tight.

**(C2)** *Join condition*: A task $T$ can join at time $t$ iff the total utilization after joining is at most $M$. If $T$ joins at time $t$, then $\theta(T_1)$ is set to $t$.
*Leave condition*: Let $T_i$ denote the last-released subtask of $T$. If $T$ is light, then $T$ can leave at time $t$ iff either $t = d(T_i) \ \wedge \ b(T_i) = 0$ or $t > d(T_i)$ holds. If $T$ is heavy, then $T$ can leave at time $t$ iff $t \geq D(T_i)$.

As before with (C1), when a task $T$ leaves the system at time $t$, (C2) implies that $lag(T, t) = 0$. We can also allow $T$ to leave with positive lag, provided its last-scheduled subtask satisfies the leave condition in (C2).

Observe that (C2) guarantees that periodic and sporadic tasks can always leave the system at period boundaries. To see why, note that if $T_i$ is the last subtask of $T$'s final job, then, because consecutive task periods do not overlap, $b(T_i) = 0$. If $T$ is heavy, then this implies that $D(T_i) = d(T_i)$. Thus, $T$ can leave at time $d(T_i)$, which also corresponds to the deadline of $T$'s last job.

## 4 Sufficiency of (C1) for Restricted Systems

In this section, we show that task systems can be correctly scheduled using EPDF on $M$ processors provided (C1) and (M1) below hold.

**(M1)** At any time, the sum of the weights of the $M-1$ heaviest tasks is at most 1.

We use the phrase "C1M1 task system" to refer to task systems satisfying both (C1) and (M1). Assume to the contrary that there exists a C1M1 task system $\tau$ that misses a deadline under EPDF. Let $S$ denote its EPDF schedule. Let $T_i$ be the subtask (in some given schedule) with the earliest deadline among all subtasks that miss a deadline, and let $t_d = d(T_i)$. Thus, all subtasks with deadlines less than $t_d$ meet their deadlines.

Note that, under EPDF, any subtask with deadline after $t_d$ is scheduled at a slot prior to $t_d$ only if no subtask with a deadline at most $t_d$ is eligible at that slot. Thus, the scheduling of $T_i$ is not affected by subtasks with deadlines greater than $t_d$. Hence, we can assume that no task in $\tau$ releases any subtask with a deadline greater than $t_d$. In other words,

$$\text{for every subtask } U_j \in \tau, d(U_j) \leq t_d. \tag{11}$$

Using this, we obtain the following bound on $LAG(\tau, t_d)$.

**Lemma 5** $LAG(\tau, t_d) \geq 1$.

**Proof:** By (10), we have

$$LAG(\tau, t_d) = \sum_{t=0}^{t_d-1} \sum_{T \in \tau} \mathit{flow}(T, t) - \sum_{t=0}^{t_d-1} \sum_{T \in \tau} S(T, t).$$

The first term on the right-hand side of the above equation is the total share in the ideal schedule in $[0, t_d)$, which equals the total number of subtasks in $\tau$. (Follows from (11).) The second term corresponds to the number of subtasks scheduled by EPDF in $[0, t_d)$. Since $T_i$ misses its deadline at $t_d$, the difference between these two terms is at least one. $\Box$

Because $LAG(\tau, 0) = 0$, by Lemma 5, there exists a time $t < t_d$ such that $LAG(\tau, t) < 1$ and $LAG(\tau, t+1) \geq 1$. We now prove some properties about task lags at time $t+1$; using these properties and (M1), we later derive a contradiction concerning the existence of time $t$.

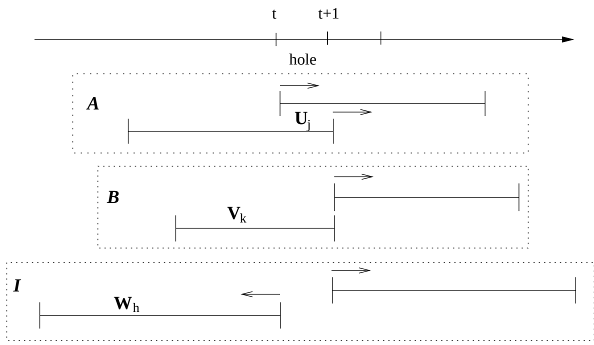


Fig. 6. Sets $A$, $B$, and $I$. The PF-windows of a sample task of each set are shown. The PF-windows are denoted by line segments. An arrow over a release (respectively, deadline) indicates that the release (respectively, deadline) could be anywhere in the direction of the arrow.

By Lemma 4, there is at least one hole in slot $t$ (*i.e.*, in $[t, t+1)$). In other words, the number of tasks scheduled in slot $t$ is at most $M-1$. Let $A$ denote the set of tasks scheduled in slot $t$. Then, we have

$$|A| \leq M - 1. \tag{12}$$

Let $B$ denote the set of tasks not in $A$ that are "active" at $t$. A task $U$ is *active* at time $t$ if it has a subtask $U_j$ such that $e(U_j) \leq t < d(U_j)$. (A task may be inactive either because it has already left the system or because of a late subtask release.) Consider any task $U \in B$ and let $U_j$ be such that $e(U_j) \leq t < d(U_j)$. Because there is a hole in slot $t$ and no subtask of $U$ is scheduled at time $t$, and because $e(U_j) \leq t < d(U_j)$, $U_j$ must be scheduled before time $t$.

Let $I$ denote the set of the remaining tasks that are not active at time $t$. Fig. 6 shows how the tasks in $A$, $B$, and $I$ are scheduled. We now estimate the *lag* values for the tasks in each of $A$, $B$, and $I$ at time $t+1$.

**Lemma 6** *For $W \in I$, $lag(W, t+1) = 0$.*

**Proof:** Consider any subtask $W_h$ of task $W$. We consider two cases depending on whether $e(W_h) \geq t+1$ holds. If $e(W_h) \geq t+1$, then $r(W_h) \geq t+1$. Therefore, by (9), $f(W_h, u) = 0$ for all slots $u \leq t < r(W_h)$. Hence, the share of $W_h$ in the ideal schedule in $[0, t+1)$ is zero. Also, in the EPDF schedule, $W_h$ is scheduled at or after $t+1$. On the other hand, if $e(W_h) \leq t$, then by the definition of $I$, $d(W_h) \leq t < t_d$. Since such a subtask meets its deadline, $W_h$ is scheduled in $[0, t)$. Hence, the share received by $W_h$ in $[0, t+1)$ is one in both the ideal and EPDF schedules. Thus, under both cases, for each subtask of $W$, the share over $[0, t+1)$ is the same in both the ideal and EPDF schedules. Therefore, $lag(W, t+1) = 0$. □

**Lemma 7** *For $V \in B$, $lag(V, t+1) \leq 0$.*

**Proof:** Consider any subtask $V_k$ of task $V$. Again, as in the proof of Lemma 6, we consider two cases. If $r(V_k) \geq t+1$, then by (9), the share of $V_k$ in $[0, t+1)$ in the ideal schedule is zero. On the other hand, if $r(V_k) \leq t$, then, as discussed earlier, $V_k$ is scheduled before $t$ because of the hole in slot $t$. Thus, the share of $V_k$ in $[0, t+1)$ is one in the EPDF schedule, and at most one in the ideal schedule. ($d(V_k)$ may be greater than $t+1$, in which case a portion of $V_k$'s share in the ideal schedule is allocated after $t+1$.) Thus, the share over $[0, t+1)$ of any subtask of $V$ in the EPDF schedule is at least that in the ideal schedule. Hence, $lag(V, t+1) \leq 0$. □

**Lemma 8** *For $U \in A$, $lag(U, t+1) < wt(U)$.*

**Proof:** Let $U_j$ be the subtask of $U$ scheduled at time $t$. Since $t < t_d$, $U_j$ meets its deadline. Therefore, $d(U_j) \geq t+1$. By (5), it follows that $\theta(U_j) + \lceil j/wt(U) \rceil \geq t+1$. By (6), $\theta(U_{j+1}) + \lceil j/wt(U) \rceil \geq t+1$, which implies that $\theta(U_{j+1}) + \lfloor j/wt(U) \rfloor \geq t$. Thus, by (4), $r(U_{j+1}) \geq t$. It follows that $r(U_k) \geq t$, where $U_k$ is $U_j$'s successor (if it exists).

If $r(U_k) \geq t+1$, then the share in the ideal schedule for any subtask after $U_j$ is zero. Thus, the total share of $U$ over $[0, t+1)$ in the EPDF schedule is at least that in the the ideal schedule, *i.e.*, $lag(U, t+1) \leq 0$.

On the other hand, if $r(U_k) = t$, then we have $r(U_k) \leq d(U_j) - 1$. By (4)–(6), this can only happen if $k = j+1$, $\theta(U_j) = \theta(U_{j+1})$, and $d(U_j) = t+1$. Hence, by (4) and (5), $\lfloor j/wt(U) \rfloor = \lceil j/wt(U) \rceil - 1$, which implies that $\lfloor j/wt(U) \rfloor < j/wt(U)$. Note that since $U_j$ is scheduled in $[0, t+1)$ in the EPDF schedule, any excess share of $U$ in the ideal schedule in $[0, t+1)$ is due to $f(U_{j+1}, t)$. Therefore, we have $lag(U, t+1) \leq f(U_{j+1}, r(U_{j+1}))$. By (9), $f(U_{j+1}, r(U_{j+1})) = (\lfloor j/wt(U) \rfloor + 1) \cdot wt(U) - j$. Hence, $lag(U, t+1) \leq (\lfloor j/wt(U) \rfloor + 1) \cdot wt(U) - j < (j/wt(U) + 1) \cdot wt(U) - j$. Thus, $lag(U, t+1) < wt(U)$. □

Because $LAG(\tau, t+1) = \sum_{U \in A \cup B \cup I} lag(U, t+1)$, by Lemmas 6–8, $LAG(\tau, t+1) < \sum_{U \in A} wt(U)$. By (12), $|A| \leq M-1$. Therefore, by (M1), $LAG(\tau, t+1) < 1$, contradicting our assumption about $t$. Thus, we have the following theorem.

**Theorem 3** EPDF *correctly schedules every* C1M1 *task system on $M$ processors.*

Since M1 is always true if $M \leq 2$, Theorem 3 generalizes our earlier result [2] that EPDF is optimal on one or two processors.

Ensuring (M1) involves identifying the $M-1$ heaviest tasks and summing their weights. A more efficient (and more restrictive) way to enforce (M1) is to require each individual task weight to be at most $1/(M-1)$.

**Corollary 1** EPDF *correctly schedules any dynamic GIS task system satisfying* (C1) *on* $M$ $(>1)$ *processors if each task's weight is at most* $\frac{1}{M-1}$.

Condition (M1) can be improved by more accurately bounding $lag(U, t+1)$ for $U \in A$. Let $U.f = \frac{U.e - gcd(U.e, U.p)}{U.p}$. Using (9), it can be shown that if $d(U_j) = r(U_{j+1}) + 1$, then $U.f$ is the maximum share subtask $U_{j+1}$ can have in slot $r(U_{j+1})$. Thus, we can improve Lemma 8 to show that $lag(U, j+1) \leq U.f$ for $U \in A$. Performing the same analysis as above, we obtain a contradiction if $\sum_{U \in A} U.f < 1$. Thus, EPDF produces a correct schedule if, at all times, $\sum_{U \in H} U.f < 1$ for all sets $H$ of at most $M-1$ tasks.

## 5 Sufficiency of (C2) for PD$^2$

We now show that PD$^2$ correctly schedules any dynamic task system for which (C2) holds. The proof strategy used here is similar to that used in [11]. Suppose that PD$^2$ misses a deadline for some task system that satisfies (C1). Then there exists a time $t_d$ and a task system $\tau$ as given in Definitions 1 and 2 below.

**Definition 1** $t_d$ is the earliest time at which any task system instance misses a deadline under PD$^2$. □

**Definition 2** $\tau$ is an instance of a task system with the following properties.

- **(T1)** $\tau$ misses a deadline under PD$^2$ at $t_d$.
- **(T2)** No task system instance satisfying (T1) releases fewer subtasks in $[0, t_d)$ than $\tau$.
- **(T3)** No task system instance satisfying (T1) and (T2) has a larger rank than $\tau$, where the *rank* of an instance is the sum of the eligibility times of all subtasks with deadlines at most $t_d$. □

Note that (T1)–(T3) are being applied *in sequence*; *e.g.*, $\tau$'s rank is maximal only among those task system instances satisfying (T1) *and* (T2).

By (T1), (T2), and Def. 1, exactly one subtask in $\tau$ misses its deadline: if several subtasks miss their deadlines, all but one can be removed and the remaining subtask will still miss its deadline, contradicting (T2). We now prove several properties about $\tau$ and $S$, the PD$^2$ schedule for $\tau$.

**Lemma 9** *The following properties hold for* $\tau$ *and* $S$.

- **(a)** *Let* $t$ *be the time at which* $T_i$ *is scheduled. Then,* $e(T_i) \geq min(r(T_i), t)$.

**(b)** *Let $t$ be as in* (a)*. If either $d(T_i) > t+1$ or $d(T_i) = t+1 \ \wedge \ b(T_i) = 0$, then $T_i$'s successor is not eligible before $t+1$.*
**(c)** *For all $T_i$, $d(T_i) \leq t_d$.*
**(d)** *There are no holes in slot $t_d - 1$.*
**(e)** $LAG(\tau, t_d) = 1$.
**(f)** $LAG(\tau, t_d - 1) \geq 1$.

**Proof of (a):** Suppose that $e(T_i) < min(r(T_i), t)$. Consider the task system instance $\tau'$ obtained from $\tau$ by changing $e(T_i)$ to $min(r(T_i), t)$. Note that $e(T_i)$ is still at most $r(T_i)$ and $\tau'$'s rank is larger than $\tau$'s. It is easy to show that the relative priorities of the subtasks do not change for any slot $u \in \{0, \ldots, t_d - 1\}$, and hence, $\tau'$ and $\tau$ have identical PD$^2$ schedules. Thus, $\tau'$ misses a deadline at $t_d$, contradicting (T3).

**Proof of (b):** Let subtask $T_k$ be $T_i$'s successor. By (3) and (6), $r(T_k) \geq d(T_i) - b(T_i)$. (Recall that consecutive PF-windows overlap by at most one slot.) If $d(T_i) > t+1$ or $d(T_i) = t+1 \ \wedge \ b(T_i) = 0$, then $r(T_k) \geq t+1$. Since $T_i$ is scheduled in slot $t$, $T_k$ is scheduled at or after $t+1$. Therefore, by (a), $e(T_k) \geq t+1$.

**Proof of (c):** Suppose $\tau$ contains a subtask $U_j$ with a deadline greater than $t_d$. $U_j$ can be removed without affecting the scheduling of higher-priority subtasks with earlier deadlines. Thus, if $U_j$ is removed, then a deadline is still missed at $t_d$. This contradicts (T2).

**Proof of (d):** If there were a hole in slot $t_d - 1$, then the subtask that misses its deadline at $t_d$ would have been scheduled there, a contradiction. (Note that its predecessor meets its deadline at or before time $t_d - 1$ and hence is not scheduled in slot $t_d - 1$.)

**Proof of (e):** By (10), we have

$$LAG(\tau, t_d) = \sum_{t=0}^{t_d - 1} \sum_{T \in \tau} \mathit{flow}(T, t) - \sum_{t=0}^{t_d - 1} \sum_{T \in \tau} S(T, t).$$

The first term on the right-hand side of the above equation is the total share in $[0, t_d)$, which equals the total number of subtasks in $\tau$. The second term corresponds to the number of subtasks scheduled by EPDF in $[0, t_d)$. Since exactly one subtask misses its deadline, the difference between these two terms is 1, *i.e.*, $LAG(\tau, t_d) = 1$.

**Proof of (f):** By (d), there are no holes in slot $t_d - 1$. Hence, by Lemma 4, $LAG(\tau, t_d - 1) \geq LAG(\tau, t_d)$. Therefore, by (e), $LAG(\tau, t_d - 1) \geq 1$. □

Because $LAG(\tau, 0) = 0$, by part (f) of Lemma 9, there exists a time $t$ such that

$$0 \leq t < t_d - 1 \ \wedge \ LAG(\tau, t) < 1 \ \wedge \ LAG(\tau, t+1) \geq 1. \tag{13}$$

Without loss of generality, let $t$ be the latest such time, *i.e.*, for all $u$ such that $t < u \leq t_d - 1$, $LAG(\tau, u) \geq 1$. We now show that such a $t$ cannot exist, thus contradicting our starting assumption that $t_d$ and $\tau$ exist.

By (13), $LAG(\tau, t) < LAG(\tau, t+1)$. Hence, by Lemma 4, *there is at least one hole in slot* $t$. Define sets $A$, $B$, and $I$ as in Sec. 4 (refer to Fig. 6). We begin by proving certain properties about $B$.

**Lemma 10** *$B$ is non-empty.*

**Proof:** Let the number of the holes in slot $t$ be $h$. Then, $\sum_{T \in \tau} S(T, t) = |A| = M - h$. By (10), $LAG(\tau, t+1) = LAG(\tau, t) + \sum_{T \in \tau} (\mathit{flow}(T, t) - S(T, t))$. Thus, because $LAG(\tau, t+1) > LAG(\tau, t)$, we have $\sum_{T \in \tau} \mathit{flow}(T, t) > M - h$. Since for every $V \in I$, either $d(V_k) < t$ or $r(V_k) > t$, by (9), $\mathit{flow}(V, t) = 0$. It follows that $\sum_{T \in A \cup B} \mathit{flow}(T, t) > M - h$. Therefore, by (F1), $\sum_{T \in A \cup B} wt(T) > M - h$. Because $|A| = M - h$ and $wt(T) \leq 1$ for any task $T$, $\sum_{T \in A} wt(T) \leq M - h$. Thus, $\sum_{T \in B} wt(T) > 0$. Hence, $B$ is not empty. □

In the proof of each of Lemmas 11–13 below, we show that if the required condition is not satisfied, then a subtask can be removed without causing the missed deadline at $t_d$ to be met. Thus, we obtain a contradiction of (T2). (Lemmas 13, and 15 below are proved in the appendix. They generalize properties proved in our earlier work on *static* GIS task systems [10,11].)

**Lemma 11** *Let $U$ be any task in $B$. Let $U_j$ be the subtask with the largest index such that $e(U_j) \leq t < d(U_j)$. Then, $d(U_j) = t + 1 \ \wedge \ b(U_j) = 1$.*

**Proof:** As shown in Sec. 4, $U_j$ must be scheduled before $t$. By (13), $t < t_d$. Hence, $U_j$ does not miss its deadline and $d(U_j) \geq t + 1$. Suppose that the following holds.

$$d(U_j) > t + 1 \text{ or } d(U_j) = t + 1 \ \wedge \ b(U_j) = 0 \tag{14}$$

We now show that $U_j$ can be removed and a deadline will still be missed at $t_d$, contradicting (T2). Let the chain of displacements caused by removing $U_j$ be $\Delta_1, \Delta_2, \ldots, \Delta_k$, where $\Delta_i = \langle X^{(i)}, t_i, \ X^{(i+1)}, t_{i+1} \rangle$ and $X^{(1)} = U_j$. By Lemma 1, $t_{i+1} > t_i$ for $1 \leq i \leq k$.

Note that at slot $t_i$, the priority of $X^{(i)}$ is at least that of $X^{(i+1)}$ because $X^{(i)}$ was chosen over $X^{(i+1)}$ in $S$. Thus, because $X^{(1)} = U_j$, by (14), for each subtask

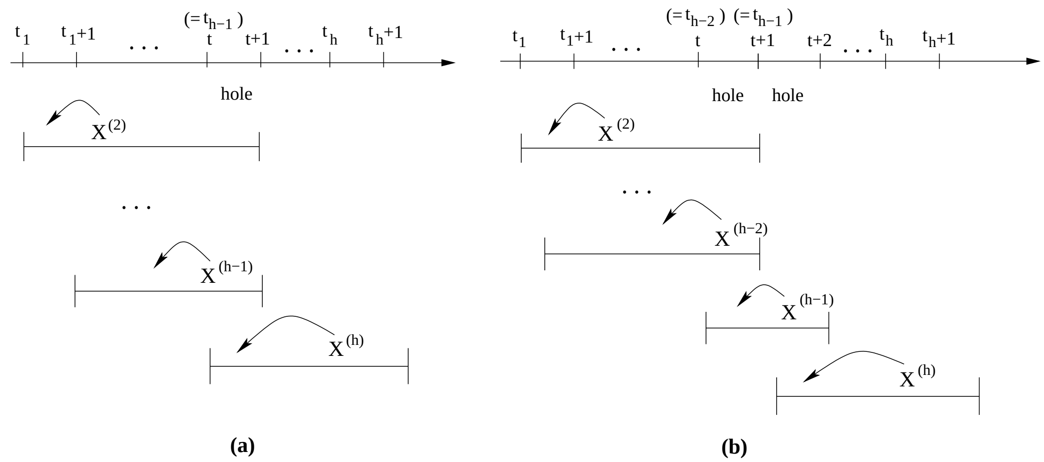


Fig. 7. **(a)** Lemma 11. IS-windows are denoted by line segments. $X^{(h)}$ must be the successor of $X^{(h-1)}$ because there is a hole in slot $t$. **(b)** Lemma 12. If there is a hole in both slots $t$ and $t+1$, then $X^{(h-2)}$ and $X^{(h-1)}$ must be scheduled at $t$ and $t+1$ in $S$, respectively. Also, $X^{(h)}$ must be the successor of $X^{(h-1)}$, which in turn, must be the successor of $X^{(h-2)}$.

$X^{(i)}, 1 \leq i \leq k+1$, either $d(X^{(i)}) > t+1$ or $d(X^{(i)}) = t+1 \ \wedge \ b(X^{(i)}) = 0$. Therefore, by part (b) of Lemma 9, the following property holds.

**(E)** The eligibility time of the successor of $X^{(i)}$ (if it exists in $\tau$) is at least $t+1$ for all $i \in [1, k+1]$.

We now show that the displacements do not extend beyond slot $t$. Assume, to the contrary, that $t_{k+1} > t$. Consider $h \in \{2, \ldots, k+1\}$ such that $t_h > t$ and $t_{h-1} \leq t$, as depicted in Fig. 7(a). Such an $h$ exists because $t_1 < t < t_{k+1}$. Because there is a hole in slot $t$ and $t_{h-1} \leq t < t_h$, by Lemma 3, $t_{h-1} = t$ and $X^{(h)}$ must be $X^{(h-1)}$'s successor. Therefore, by (E), $e(X^{(h)}) \geq t+1$. This implies that $\Delta_{h-1}$ is not valid.

Thus, the displacements do not extend beyond slot $t$, implying that no subtask scheduled after $t$ is left-shifted. Hence, a deadline is still missed at time $t_d$, contradicting (T2). Hence, $d(U_j) = t+1 \ \wedge \ b(U_j) = 1$. □

**Lemma 12** *There is no hole in slot $t+1$ if $B$ has at least one light task.*

**Proof:** By (13), $t < t_d - 1$, and therefore, $t+1 \leq t_d - 1$. Suppose that there is a hole in slot $t+1$. By part (d) of Lemma 9, $t+1 < t_d - 1$, *i.e.*,

$$t + 2 \leq t_d - 1. \tag{15}$$

Let $U$ be a light task in $B$ and let $U_j$ be the subtask of $U$ with the largest index such that $e(U_j) \leq t < d(U_j)$. Our approach is the same as in the proof of Lemma 11. Let the chain of displacements caused by removing $U_j$ be

$\Delta_1, \Delta_2, \ldots, \Delta_k$, where $\Delta_i = \langle X^{(i)}, t_i, X^{(i+1)}, t_{i+1}\rangle$ and $X^{(1)} = U_j$. By Lemma 1, we have $t_{i+1} > t_i$ for all $i \in [1, k]$. Also, the priority of $X^{(i)}$ is at least that of $X^{(i+1)}$ at $t_i$, because $X^{(i)}$ was chosen over $X^{(i+1)}$ in $S$. Because $U$ is light and $d(U_j) = t + 1 \ \wedge \ b(U_j) = 1$ (by Lemma 11), this implies the following.

**(P)** For all $i \in [1, k+1]$, either **(i)** $d(X^{(i)}) > t+1$ or **(ii)** $d(X^{(i)}) = t+1$ and $X^{(i)}$ is the subtask of a light task.

Suppose the chain of displacements extends beyond $t+1$, *i.e.*, $t_{k+1} > t+1$. Consider $h \in \{1, \ldots, k+1\}$ such that that $t_h > t+1$ and $t_{h-1} \le t+1$. Because there is a hole in slot $t+1$ and $t_{h-1} \le t+1 < t_h$, by Lemma 3, $t_{h-1} = t+1$ and $X^{(h)}$ is the successor of $X^{(h-1)}$. Similarly, because there is a hole in slot $t$, $t_{h-2} = t$ and $X^{(h-1)}$ is the successor of $X^{(h-2)}$. This is illustrated in Fig. 7(b).

By (P), either $d(X^{(h-2)}) > t+1$ or $d(X^{(h-2)}) = t+1$ and $X^{(h-2)}$ is the subtask of a light task. In either case, $d(X^{(h-1)}) > t+2$. To see why, note that if $d(X^{(h-2)}) > t+1$, then because $X^{(h-1)}$ is the successor of $X^{(h-2)}$, by (5), $d(X^{(h-1)}) > t+2$. On the other hand, if $d(X^{(h-2)}) = t+1$ and $X^{(h-2)}$ is the subtask of a light task, then, by (L), $d(X^{(h-1)}) > t+2$.

Now, because $X^{(h-1)}$ is scheduled at $t+1$, by part (b) of Lemma 9, the successor of $X^{(h-1)}$ is not eligible before $t+2$, *i.e.*, $e(X^{(h)}) \ge t+2$. This implies that the displacement $\Delta_{h-1}$ is not valid. Thus, the chain of displacements cannot extend beyond time $t+2$. Hence, because $t+2 \le t_d - 1$ (by (15)), removing $U_j$ cannot cause a missed deadline at $t_d$ to be met. This contradicts (T2). Hence, there is no hole in slot $t+1$. □

Lemma 13 below is needed in systems consisting solely of heavy tasks and is the counterpart of Lemma 12 for such systems.

**Lemma 13** *Let $U$ be a heavy task in $B$ and let $U_j$ be the subtask of $U$ with the largest index such that $e(U_j) \le t < d(U_j)$. Then, there exists a slot in $[d(U_j), \min(D(U_j), t_d))$ with no holes.*

**Lemma 14** *If $B$ has at least one light task, then $LAG(\tau, t+2) < 1$.*

**Proof:** Let the number of holes in slot $t$ be $h$. We now derive some properties about the *flow* values in slots $t$ and $t+1$.

By definition, only tasks in $A \cup B$ are active at time $t$. Thus, $\sum_{T \in \tau} flow(T, t) = \sum_{T \in A \cup B} flow(T, t)$. Since $wt(T) \le 1$ for any $T$, we have $\sum_{T \in A} wt(T) \le |A|$. Thus, by (F1), $\sum_{T \in A} flow(T, t) \le |A|$. Now, because there are $h$ holes in slot $t$, $M - h$ tasks are scheduled at $t$, *i.e.*, $|A| = M - h$. Thus, $\sum_{T \in A} flow(T, t) \le$

$M - h$ and

$$\sum_{T \in \tau} \mathit{flow}(T, t) \leq M - h + \sum_{T \in B} \mathit{flow}(T, t). \tag{16}$$

Consider $U \in B$. Let $U_j$ be the subtask of $U$ with the largest index such that $e(U_j) \leq t < d(U_j)$. Let $C$ denote the set of such subtasks for all tasks in $B$. Then, by Lemma 11,

$$\text{for all } U_j \in C,\ d(U_j) = t + 1 \ \wedge\ b(U_j) = 1. \tag{17}$$

If $U$ is heavy, then this would imply that $D(U_j) > t + 1$. (By the definition of a group deadline, for any subtask $T_i$ of a heavy task $T$, $D(T_i) = d(T_i)$ holds iff $b(T_i) = 0$.) Thus, the leave condition in (C2) is not satisfied at time $t + 1$, and hence no task in $B$ leaves at time $t + 1$.

Let $A'$ ($I'$) denote the tasks in $A$ ($I$) that are active at time $t + 1$. Then, the set of active tasks at time $t + 1$ is $A' \cup I' \cup B$. Thus, by the join condition in (C2),

$$\sum_{T \in A' \cup I' \cup B} wt(T) \leq M. \tag{18}$$

Also, $\sum_{T \in \tau} \mathit{flow}(T, t + 1) = \sum_{T \in A' \cup I' \cup B} \mathit{flow}(T, t + 1)$. By (F1), this implies that $\sum_{T \in \tau} \mathit{flow}(T, t + 1) \leq \sum_{T \in A' \cup I'} wt(T) + \sum_{T \in B} \mathit{flow}(T, t + 1)$. Thus, by (16),

$$\begin{aligned} \sum_{T \in \tau} (\mathit{flow}(T, t) + \mathit{flow}(T, t + 1)) \leq M - h + & \sum_{T \in A' \cup I'} wt(T) \\ & + \sum_{T \in B} (\mathit{flow}(T, t) + \mathit{flow}(T, t + 1)) \end{aligned} \tag{19}$$

Consider $U_j \in C$ (hence, $U \in B$). Let $U_k$ denote the successor of $U_j$. Since $U_j$ is the subtask with the largest index such that $e(U_j) \leq t < d(U_j)$, we have $e(U_k) \geq t + 1$. Hence, $r(U_k) \geq t + 1$. By (17), we have $d(U_j) = t + 1$. Therefore, by (F2), $\mathit{flow}(U, t) + \mathit{flow}(U, t + 1) \leq wt(U)$ for each $U \in B$. By (19), this implies that $\sum_{T \in \tau} (\mathit{flow}(T, t) + \mathit{flow}(T, t + 1)) \leq M - h + \sum_{T \in A' \cup I' \cup B} wt(T)$. Thus, from (18), it follows that

$$\sum_{T \in \tau} (\mathit{flow}(T, t) + \mathit{flow}(T, t + 1)) \leq M - h + M. \tag{20}$$

By the statement of the lemma, $B$ contains at least one light task. Therefore, by Lemma 12, there is no hole in slot $t + 1$. Since there are $h$ holes in slot $t$, we have $\sum_{T\in\tau}(S(T,t)+S(T,t+1)) = M-h+M$.

Hence, by (20), $\sum_{T\in\tau}(\mathit{flow}(T,t)+\mathit{flow}(T,t+1)) \leq \sum_{T\in\tau}(S(T,t)+S(T,t+1))$. Using this relation in the identity (obtained from (10)), $LAG(\tau,t+2) = LAG(\tau,t)+\sum_{T\in\tau}(\mathit{flow}(T,t)+\mathit{flow}(T,t+1)) - \sum_{T\in\tau}(S(T,t)+S(T,t+1))$, and the fact that $LAG(\tau,t)<1$, we obtain $LAG(\tau,t+2)<1$. □

The following lemma generalizes Lemma 14 by allowing $B$ to consist solely of heavy tasks. It is proved in the appendix.

**Lemma 15** *There exists $v \in \{t+2,\ldots,t_d\}$ such that $LAG(\tau,v)<1$.*

Recall our assumption that $t$ is the latest time such that $LAG(\tau,t)<1$ and $LAG(\tau,t+1)\geq 1$. Because $t \leq t_d-2$ (by (13)), we have $t+2\leq t_d$. By Lemma 15, $LAG(\tau,v)\leq 0$ for some $v\in\{t+2,\ldots,t_d\}$. By parts (e) and (f) of Lemma 9, $v\leq t_d-2$. Because $LAG(\tau,t_d)\geq 1$, this contradicts the maximality of $t$. Therefore, $t_d$ and $\tau$ as defined cannot exist. Thus, we have the following.

**Theorem 4** PD$^2$ *correctly schedules any dynamic GIS task system satisfying* (C2).

## 6 Conclusions

In this paper, we have addressed the problem of scheduling dynamic GIS task systems on multiprocessors. We have shown that if the sum of the weights of the $M-1$ heaviest tasks is at most 1 and EPDF is used, then the uniprocessor join/leave conditions presented previously [5,12] are sufficient to avoid deadline misses on $M$ processors. This result applies to any EPDF-based algorithm, and hence to PD$^2$ as well. We have also provided join/leave conditions for the general case in which weights are not restricted in this way and tasks are scheduled using PD$^2$. We have further shown that, in general, it is not possible to improve upon these conditions.

## References


[1] J. Anderson and A. Srinivasan. Mixed Pfair/ERfair scheduling of asynchronous periodic tasks. *Journal of Computer and System Sciences*. To appear.

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

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

[4] S. Baruah, J. Gehrke, and C.G. Plaxton. Fast scheduling of periodic tasks on multiple resources. In *Proceedings of the 9th International Parallel Processing Symposium*, pages 280–288, April 1995.

[5] S. Baruah, J. Gehrke, C.G. Plaxton, I. Stoica, H. Abdel-Wahab, and K. Jeffay. Fair on-line scheduling of a dynamic set of tasks on a single resource. *Information Processing Letters*, 26(1):43–51, Jan. 1998.

[6] A. Chandra, M. Adler, P. Goyal, and P. Shenoy. Surplus fair scheduling: A proportional-share CPU scheduling algorithm for symmetric multiprocessors. In *Proceedings of the 4th ACM Symposium on Operating System Design and Implementation*, pages 45–58, Oct. 2000.

[7] A. Chandra, M. Adler, and P. Shenoy. Deadline fair scheduling: Bridging the theory and practice of proportionate-fair scheduling in multiprocessor servers. In *Proceedings of the 7th IEEE Real-time Technology and Applications Symposium*, pages 3–14, May 2001.

[8] S. Keshav. Private communication, 2001.

[9] L. Sha, R. Rajkumar, J. Lehoczky, and K. Ramamritham. Mode change protocols for priority-driven pre-emptive scheduling. *Real-time Systems*, 1(3):244–264, 1989.

[10] A. Srinivasan and J. Anderson. Optimal rate-based scheduling on multiprocessors. *Submitted to Real-time Systems*.

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

[12] I. Stoica, H. Abdel-Wahab, K. Jeffay, S. Baruah, J. Gehrke, and C.G. Plaxton. A proportional share resource allocation algorithm for real-time, time-shared systems. In *Proceedings of the 17th IEEE Real-time Systems Symposium*, pages 288–299, Dec. 1996.

[13] K. W. Tindell, A. Burns, and A. J. Wellings. Mode changes in priority pre-emptively scheduled systems. In *Proceedings of the IEEE Real-time Systems Symposium*, pages 100–109, Dec. 1992.

## Appendix: Proofs of Lemma 13 and Lemma 15

In this appendix, we present the proofs of Lemma 13 and 15 omitted earlier. In all the lemmas, the time $t$ corresponds to that defined in Equation (13).

**Lemma 13** *Let $U$ be a heavy task in $B$ and let $U_j$ be the subtask of $U$ with the largest index such that $e(U_j) \le t < d(U_j)$. Then, there exists a slot in $[d(U_j), \min(D(U_j), t_d))$ with no holes.*

**Proof:** By Lemma 11, $d(U_j) = t + 1 \wedge b(U_j) = 1$. By (13), $t < t_d - 1$. Therefore $d(U_j) \le t_d - 1$. If $min(D(U_j), t_d) = t_d$, then by part (f) of Lemma 9, slot $t_d - 1$ satisfies the stated requirement. In the rest of the proof, assume that $D(U_j) < t_d$. Let $v = D(U_j)$. Since $b(U_j) = 1$, by the definition of $D$, $D(U_j) > d(U_j)$, *i.e.*,

$$t + 1 < v. \tag{21}$$

Suppose that the following property holds.

**(H)** There is a hole in slot $u$ for all $u \in \{t, \ldots, v - 1\}$.

Given (H), we show that removing $U_j$ does not cause the missed deadline to be met, contradicting (T2). Let $\Delta_1, \Delta_2, \ldots, \Delta_k$ be the chain of displacements caused by removing $U_j$, where $\Delta_i = \langle X^{(i)}, t_i, X^{(i+1)}, t_{i+1}\rangle$, $X^{(1)} = U_j$, and $t_1$ is the slot in which $U_j$ is scheduled. By Lemma 1, $t_{i+1} > t_i$ for all $i \in [1, k - 1]$. Also, the priority of $X^{(i)}$ is at least that of $X^{(i+1)}$ at $t_i$, because $X^{(i)}$ was chosen over $X^{(i+1)}$ at $t_i$ in $S$. Thus, by Lemma 11, for all $i \in [2, k + 1]$, one of the following holds:

**(a)** $d(X^{(i)}) > t + 1$,
**(b)** $d(X^{(i)}) = t + 1 \ \wedge \ b(X^{(i)}) = 0$, or
**(c)** $d(X^{(i)}) = t + 1 \ \wedge \ b(X^{(i)}) = 1 \ \wedge \ D(X^{(i)}) \le v$.

We now show that the displacements do not extend beyond slot $v - 1$ (which implies that $U_j$ can be removed without causing the missed deadline to be met). Suppose, to the contrary, they do extend beyond slot $v - 1$, *i.e.*, $t_{k+1} > v - 1$.

Let $t_g$ be the largest $t_i$ such that $t_i < t$ and let $t_h$ be the smallest $t_i$ such that $t_i > v - 1$. (Note that such a $t_g$ exists because $t_1 < t$.) Then, by (H), there are holes in all slots in $[t_{g+1}, t_{h-1}]$. Thus, by Lemma 3,

$$\forall i \in [g+1, h-1],\ X^{(i+1)} \text{ is the successor of } X^{(i)}. \tag{22}$$

Also, $t_{i+1} = t_i + 1$ for all $i \in [g + 1, h - 2]$.

$$t_{g+1} = t \ \wedge \ t_{h-1} = v - 1 \tag{23}$$
$$\forall i \in \{g + 1, \ldots, h - 1\},\ t_i = t + i - (g + 1) \tag{24}$$

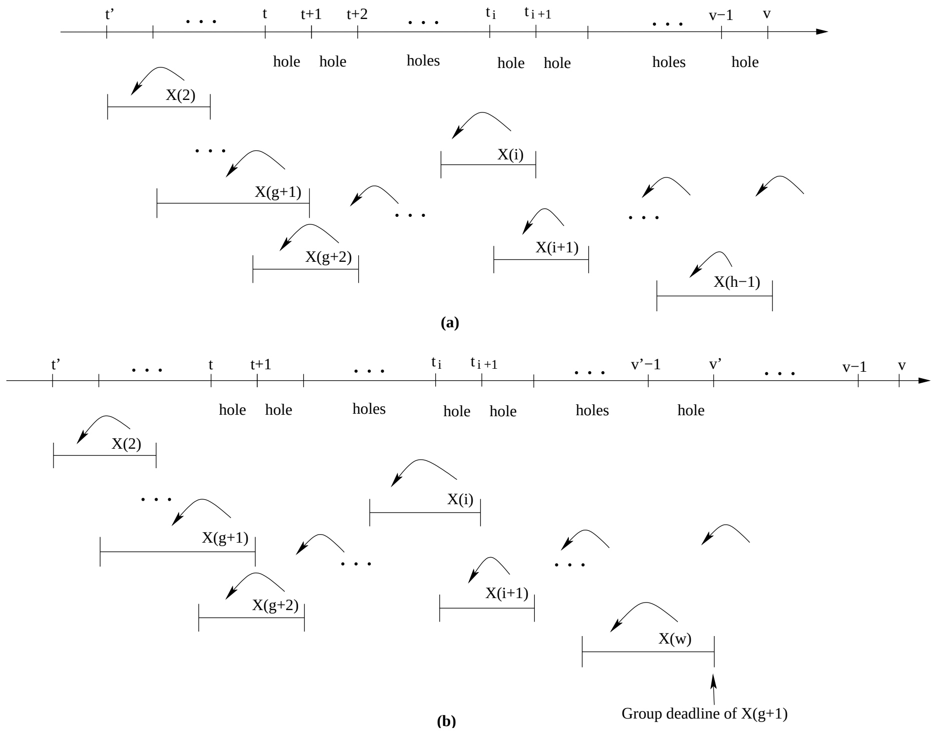


Fig. 8. Lemma 13. **(a)** There are holes in all slots in $[t, v)$. $X^{(i)}$ scheduled at $t_i$ displaces $X^{(i-1)}$ scheduled at $t_{i-1}$. By (24), the $t_i$'s are consecutive and satisfy $t_i = t + i - (g + 1)$. Further, $X^{(h-1)}$ is the subtask scheduled in slot $v - 1$. **(b)** Case 2. $D(X^{(g+1)}) = v'$. Hence, either $d(X^{(w)}) = v' \ \wedge\ b(X^{(w)}) = 0$ (as depicted) or $d(X^{(w)}) > v'$.

Earlier, we showed that one of (a)–(c) holds for all $i \in [2, k + 1]$. If either $d(X^{(g+1)}) > t + 1$ or $d(X^{(g+1)}) = t + 1 \ \wedge\ b(X^{(g+1)}) = 0$, then since $X^{(g+1)}$ is scheduled at $t$, by Lemma 9, part (b), $e(X^{(g+2)}) \geq t + 1$ (recall that, by (22), $X^{(g+2)}$ is the successor of $X^{(g+1)}$). In other words, the displacement $\Delta_g$ is not valid. Therefore,

$$d(X^{(g+1)}) = t + 1 \ \wedge\ b(X^{(g+1)}) = 1 \ \wedge\ D(X^{(g+1)}) \leq v. \tag{25}$$

We now consider two cases. In each, we show that the displacements do not extend beyond $v - 1$, as desired.

**Case 1: $X^{(g+1)}$ is the subtask of a light task.** By (21), $t + 1 \leq v - 1$ and hence, by (H), there is a hole in both $t$ and $t + 1$. Also, by (23) and (24), we have $v - 1 = t + (h - 1) - (g + 1) = t + h - g - 2$. Because $t < v - 1$ (by (21)), we have $h > g + 2$, *i.e.*,

$$h \geq g + 3.$$

Because $X^{(g+1)}$ is the subtask of a light task, the reasoning used in the proof of Lemma 12 applies. Thus, the displacement $\Delta_{g+2}$ is not valid. Hence, the displacements do not extend beyond slot $t+1$ (and hence, slot $v-1$).

**Case 2: $X^{(g+1)}$ is the subtask of a heavy task.** Let $v' = D(X^{(g+1)})$. By (25), $v' \le v$. We now show that the displacements cannot extend beyond slot $v'-1$ (and hence, slot $v-1$). By (24), $X^{(i)}$ is scheduled in slot $t+i-(g+1)$ in $S$ for all $i \in \{g+1, \ldots, h-1\}$. By (22), all $X^{(i)}$ where $g+1 \le i \le h$ are subtasks of the same heavy task. We now show that the displacement $\Delta_{v'-1-t+(g+1)} (= \Delta_{v'-t+g})$ is not valid. Let $w = v'-t+g$.

By (24), $t_w = v'-1$. Because $X^{(i)}$ is scheduled at $t_i$, the subtask scheduled at $v'-1$ is $X^{(w)}$. Since $X^{(i+1)}$ is the successor of $X^{(i)}$, by (5), $d(X^{(i)}) > d(X^{(i-1)})$ for all $i \in [g+2, w]$. Because $d(X^{(g+1)}) = t+1$ (by (25)),

$$\forall i \in [g+1, w],\ d(X^{(i)}) \ge t+i-g. \tag{26}$$

In particular, $d(X^{(w)}) \ge v'$.

We now show that if $d(X^{(w)}) = v'$, then $b(X^{(w)}) = 0$. In this case, because $d(X^{(w-1)}) < d(X^{(w)})$, we have $d(X^{(w-1)}) < v'$. By (26), $d(X^{(w-1)}) \ge v'-1$. Therefore, $d(X^{(w-1)}) = v'-1$. Similarly, by induction, $d(X^{(i)}) = u+i-g)$ for all $i \in [g+1, w]$. (Refer to Fig. 8(b).) Because $D(X^{(g+1)}) = v'$, by the definition of $D$, $b(X^{(v'-u+g+1)}) = 0$. (In this case, the group deadline corresponds to the last slot of a window of length two.)

Thus, either $d(X^{(w)}) > v'$ or $d(X^{(w)}) = v' \wedge b(X^{(w)}) = 0$. Since $X^{(w)}$ is scheduled at $v'-1$, by Lemma 9, part (b), the eligibility time of the successor of $X^{(w)}$ is at least $v'$. Hence, $\Delta_w$ is not valid. Thus, the displacements do not extend beyond slot $v'-1$. $\square$

The following claims are used in proving Lemma 15.

**Claim 1** *If $U_j$ is scheduled in slot $u$, where $0 \le u < t_d$ and $u \le d(U_j)$, and if there is a hole in slot $u$, then $d(U_j) = u+1$.*

**Proof:** Because $u < t_d$, by Def. 1, no deadline is missed in $[0, u+1)$. Because $U_j$ is scheduled in slot u, *i.e.*, $[u, u+1)$, we have $d(U_j) \ge u+1$. Suppose that $d(U_j) > u+1$. Then, by part (b) of Lemma 9, the successor of $U_j$ (if it exists) is not eligible before $u+1$. Hence, by Lemma 2, we can remove $U_j$ and no displacements will result, *i.e.*, a deadline is still missed at $t_d$, contradicting (T2). Therefore, $d(U_j) = u+1$. $\square$

**Claim 2** *Suppose there is a hole in slot $u \in \{0, \ldots, t_d-1\}$. Let $U_j$ be a subtask scheduled at $t' \le u$. If the eligibility time of the successor of $U_j$ is at least $u+1$, then $d(U_j) \le u+1$.*

**Proof:** If $t' = u$, then by Claim 1, $d(U_j) = u + 1$. On the other hand, if $t' < u$ and $d(U_j) \geq u$, then by Lemma 11, $d(U_j) = u + 1$. $\Box$

We use the following property about flows (proved in [11]) in the proof of Lemma 15.

**(F3)** Let $T_i$ be a subtask of a heavy task $T$ such that $b(T_i) = 1$ and let $T_k$ be the successor of $T_i$. If $u \in \{d(T_i), \ldots, D(T_i) - 1\}$ and $u \leq r(T_k)$, then $\mathit{flow}(T, d(T_i)) + \mathit{flow}(T, u) \leq wt(T)$.

**Lemma 15** *There exists $v \in \{t+2, \ldots, t_d\}$ such that $LAG(\tau, v) < 1$.*

**Proof:** Because $LAG(\tau, t) < 1$ and $LAG(\tau, t+1) \geq 1$ (by (13)),

$$LAG(\tau, t) < LAG(\tau, t+1). \tag{27}$$

Thus, by Lemma 4, we have the following property.

**(H)** There is at least one hole in slot $t$.

Let $A, B$, and $I$ be as defined in the proof of Lemma 14. If any task in $B$ is light, then by Lemma 14, $LAG(\tau, t+2) \leq 0$, which establishes our proof obligation. We henceforth assume all tasks in $B$ are heavy.

Let $U$ be any task in $B$. Let $U_j$ be the subtask with the largest index such that $e(U_j) \leq t < d(U_j)$. Let $C$ denote the set of such subtasks of all tasks in $B$. Then, by Lemma 11,

$$\forall\ U_j \in C,\ d(U_j) = t + 1\ \wedge\ b(U_j) = 1. \tag{28}$$

Let $L_i$ be the lowest-priority subtask in $C$. Then,

$$\forall U_j \in C,\ d(U_j) = t + 1\ \wedge\ b(U_j) = 1\ \wedge\ D(U_j) \geq D(L_i). \tag{29}$$

By Lemma 13, there is a slot in $[t, \min(D(L_i), t_d))$ with no hole. Let $u$ be as follows.

**(U)** $u$ is the earliest slot in $[t, \min(D(L_i), t_d))$ with no hole.

Fig. 9 depicts this situation. By (U) and (H),

$$u \geq t + 1, \tag{30}$$

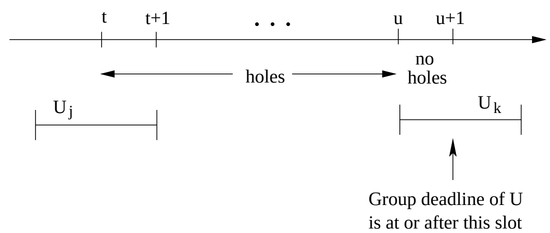


Fig. 9. Lemma 15. $U_j \in C$ and $U_k$ is the successor of $U_j$. There is a hole in each slot in $[t, u)$ and there is no hole in slot $u$. The earliest time at which $U_k$'s PF-window starts is $u$, *i.e.*, $r(U_k) \geq u$.

and there are holes in all slots in $\{t, \ldots, u-1\}$. We now establish the following property about tasks in $B$.

**Claim 3** *All tasks in $B$ are inactive over the interval $[t+1, u)$.*

**Proof:** If the interval $[t+1, u)$ is empty, then the claim is vacuously true, so assume it is nonempty. Let $V$ be any task in $B$. We first show that no subtask of $V$ is scheduled in $[t, u)$.

Note that because $V \in B$, no subtask of $V$ is scheduled in slot $t$. Let $V_i$ be the earliest subtask of $V$ scheduled in $[t+1, u)$ and let $v$ be the slot in which it is scheduled. Because there is hole in slot $v$, by Claim 1, $d(V_i) = v + 1$. By (4) and (5), this implies that $r(V_i) \leq v$. If $r(V_i) < v$, then $e(V_i) < v$. Thus, because there are holes in all slots in $\{t, \ldots, v-1\}$, it should have been scheduled earlier. Therefore, $r(V_i) = v$, which implies that $wt(V) = 1$. However, this contradicts the fact that some subtask of $V$ has a $b$-bit of 1 (by (28)). Hence, no subtask of any task in $B$ is scheduled in $[t, u)$ (see Fig. 9). Moreover, because there are holes in all slots in $[t, u)$, the earliest slot after $t$ at which a subtask of a task in $B$ is eligible to be scheduled is $u$. By (28), this implies that all the tasks in $B$ are inactive in $[t+1, u-1]$. $\square$

For any $U_j \in C$, by (29) and (U), $D(U_j) > u$. Therefore, by (C2), task $U$ cannot leave before time $u+1$. Thus, no task in $B$ can leave before time $u+1$.

Let $U_j$ be any subtask in $C$, and let $U_k$ be the successor of $U_j$. By Claim 3, $r(U_k) \geq u$. Furthermore, by (28)–(30) and (U), $d(U_j) = t + 1 \leq u < D(U_j)$. Hence, by (F3), $\mathit{flow}(U, t) + \mathit{flow}(U, u) \leq wt(U)$. Because this argument applies to all tasks in $B$, we have

$$\forall U \in B,\ \mathit{flow}(U, t) + \mathit{flow}(U, u) \leq wt(U). \qquad (31)$$

We now show that $LAG$ is non-increasing over $[t+1, u)$.

**Claim 4** *$LAG(\tau, v+1) \leq LAG(\tau, v)$ for all $v \in \{t+1, \ldots u-1\}$,*

**Proof:** If $\{t+1, \ldots, u\}$ is empty, then the claim is vacuously true, so assume it is nonempty. Suppose for some $v \in \{t+1, \ldots, u-1\}$, $LAG(\tau, v+1) >$

$LAG(\tau, v)$. Then, by Lemma 10, there exists a task that is active at $v$ but not scheduled at $v$. Let $V$ be one such task and let $V_k$ be the subtask with the largest index such that

$$e(V_k) \leq v < d(V_k). \tag{32}$$

Because no subtask of $V$ is scheduled at $v$ and because there is a hole at $v$, $V_k$ is scheduled before $v$. By (U), there is a hole at $v-1$; moreover, because $t+1 \leq v \leq u-1$, we have $v-1 \in \{t, \ldots, u-2\} \subseteq \{0, \ldots, t_d - 1\}$. Hence, by Claim 2, we have $d(V_k) \leq v$, which contradicts (32). Therefore, $LAG(\tau, v+1) \leq LAG(\tau, v)$ for all $v \in [t+1, u-1]$. □

We now show that $LAG(\tau, u+1) \leq 0$, which establishes our proof obligation.

For each $v \in \{t, \ldots, u\}$, let $H_v$ denote the number of holes in slot $v$. Then, $M - H_v$ tasks are scheduled in slot $v$. Also, let $I_v$ ($A_v$) denote the tasks in $I$ ($A$) that are active at $v$.

By (10) and Claim 4, $\sum_{T \in \tau} \mathit{flow}(T, v) \leq \sum_{T \in \tau} S(T, v)$. Therefore,

$$\forall\ v \in \{t+1, \ldots, u-1\},\ \sum_{T \in \tau} \mathit{flow}(T, v) \leq M - H_v. \tag{33}$$

By the join condition in (C2) and by (7), we have $\sum_{T \in B \cup I_u \cup A_u} wt(T) \leq M$. Hence, by (31) and (F1), it follows that $\sum_{T \in B}(\mathit{flow}(T, t) + \mathit{flow}(T, u)) + \sum_{T \in I_u \cup A_u} \mathit{flow}(T, u) \leq M$. Thus,

$$\sum_{T \in B} \mathit{flow}(T, t) + \sum_{T \in B \cup I_u \cup A_u} \mathit{flow}(T, u) \leq M. \tag{34}$$

Because the tasks in $A (= A_t)$ are the ones scheduled in slot $t$, the number of tasks in set $A_t$ is $M - H_t$. Hence, by (F1) and because the weight of each task is at most one,

$$\sum_{T \in A_t} \mathit{flow}(T, t) \leq \sum_{T \in A_t} wt(T) \leq M - H_t. \tag{35}$$

We are now ready to show that $LAG(\tau, u+1) \leq 0$. Because $S(T, v) = M - H_v$, by (10), $LAG(\tau, u+1) - LAG(\tau, t) = R$, where $R = \sum_{v=t}^{u} \left(\sum_{T \in \tau} \mathit{flow}(T, v)\right) - \sum_{v=t}^{u} (M - H_v)$. Because, by (U), there are no holes in slot $u$, $H_u = 0$. Therefore,

$$R = \sum_{v=t}^{u} \left( \sum_{T \in \tau} \mathit{flow}(T, v) \right) - \sum_{v=t}^{u-1} (M - H_v) - M. \tag{36}$$

The right-hand side of (36) can be rewritten as follows.

$$
\begin{aligned}
&\sum_{T\in\tau}(\mathit{flow}(T,t)+\mathit{flow}(T,u))-(M-H_t)-M \\
&+\sum_{v=t+1}^{u-1}\left(\sum_{T\in\tau}\mathit{flow}(T,v)-(M-H_v)\right)
\end{aligned}
$$

Rearranging terms, and using $\sum_{T\in I}\mathit{flow}(T,t)=0$ (which follows by the definition of $I$), we get

$$
\begin{aligned}
&\sum_{T\in B}\mathit{flow}(T,t)+\sum_{T\in B\cup I_u\cup A_u}\mathit{flow}(T,u)-M \\
&+\sum_{T\in A_t}\mathit{flow}(T,t)-(M-H_t) \\
&+\sum_{v=t+1}^{u-1}\left(\sum_{T\in\tau}\mathit{flow}(T,v)-(M-H_v)\right).
\end{aligned}
$$

By (33)–(35), the above value is non-positive. Hence, by (36), $LAG(\tau,u+1)-LAG(\tau,t)\leq 0$. Because $LAG(\tau,t)<1$, this implies that $LAG(\tau,u+1)<1$.

By (U) and (30), $t+1\leq u\leq min(D(U_j),t_d)-1$. Hence, $t+2\leq u+1\leq t_d$. Thus, there exists a $v\in\{t+2,\ldots,t_d\}$ such that $LAG(\tau,v)<1$. $\square$