Optimal Rate-based Scheduling on Multiprocessors ⋆

Anand Srinivasan and James H. Anderson

Department of Computer Science, University of North Carolina
Chapel Hill, NC 27599-3175 USA
December 2004

Abstract

The PD^2 Pfair/ERfair scheduling algorithm is the most efficient known algorithm for optimally scheduling periodic tasks on multiprocessors. In this paper, we prove that PD^2 is also optimal for scheduling "rate-based" tasks whose processing steps may be highly jittered. The rate-based task model we consider generalizes the widely-studied sporadic task model.

Key words: Fairness, multiprocessors, optimality, Pfair, real time, scheduling

1 Introduction

In the real-time scheduling literature, the periodic [15] and sporadic [16] task models have received the most attention. In the periodic model, each task is invoked repeatedly, with consecutive invocations, or *jobs*, being spaced apart by a fixed amount; in the sporadic model, a lower bound on the time between invocations is assumed. In practice, however, event occurrences often are neither periodic nor sporadic. For example, in an application that services packets arriving over a network, packet arrivals may be highly jittered. Rate-based scheduling schemes are more seamlessly able to cope with jitter. In such schemes, there is no restriction on a task's instantaneous rate of execution, but an average rate is assumed. If a task's instantaneous rate exceeds its average rate, then it is dealt with by using simple mechanisms such as postponing

⋆ Work supported by NSF grants CCR 9972211, CCR 9988327, ITR 0082866, CCR 0204312, and CCR 0309825. Some of the results in this paper were presented in preliminary form at the 34th ACM Symposium on Theory of Computing [18].

deadlines. In this paper, we investigate rate-based scheduling on multiprocessors. The starting point for our work is recent research on Pfair and ERfair scheduling algorithms, which are known to be optimal for scheduling periodic tasks on multiprocessors [3,5–7].

Pfair scheduling and variants. Under *Pfair scheduling*, each task is required to execute at a uniform rate, while respecting a fixed allocation quantum. Uniform rates are ensured by requiring the allocation error for each task to be always less than one quantum, where "error" is determined by comparing to an ideal fluid system. Due to this requirement, each task is effectively subdivided into quantum-length *subtasks* that must execute within *windows* of approximately equal lengths: if a subtask of a task T executes outside of its window, then T's error bounds are exceeded. The length and alignment of a task's Pfair windows are determined by its *weight*, which is defined as the ratio of its per-job execution cost and period. An example Pfair window layout for a task of weight 8/11 is given in Figure 1, which is considered in detail later.

Under Pfair scheduling, if some subtask of a task T executes "early" within its window, then T is ineligible for execution until the beginning of its next window. This means that Pfair scheduling algorithms are necessarily not work conserving when used to schedule periodic tasks. A scheduling algorithm is *work conserving* if no processor ever idles unnecessarily. More precisely, if there are M processors, and k uncompleted jobs at time t, then $min(k, M)$ processors should be busy at time t. Work-conserving algorithms are of interest because their use often results in lower job response times, especially in lightly-loaded systems. In addition, non-work-conserving algorithms often entail higher runtime overheads. (Extra bookkeeping must be done to keep track of when a job is and is not eligible.) In [3], we introduced a work-conserving variant of Pfair scheduling called *Early-release fair* (*ERfair*) scheduling. Under ERfair scheduling, subtasks may be released "early," *i.e.*, such a subtask may become eligible for execution before its Pfair window. This is illustrated in Figure 2.

In [4,18], we proposed a further extension of the Pfair task model called the *intra-sporadic* (IS) model. The sporadic model generalizes the periodic model by allowing *jobs* to be released "late," *i.e.*, the separation between consecutive job releases of a task is allowed to be more than the task's period. The IS model generalizes this by allowing *subtasks* to be released late, as illustrated in Figure 3. Early-release behavior is also allowed. As explained later, the IS notion of a rate is quite similar to that found in the recently-proposed *uniprocessor* rate-based execution model [14]. In [4], we presented an algorithm that optimally schedules IS tasks on two processors. However, we left open the problem of optimally scheduling IS tasks on systems of more than two processors.

Contributions of this paper. In this paper, we close this problem by showing that the PD^2 Pfair algorithm [3,5] correctly schedules any feasible IS task system on M processors. Because the IS model is a generalization of the sporadic model, our work also shows that PD^2 is optimal for scheduling sporadic tasks on multiprocessors. As periodic task systems represent a "worst-case" scenario in the spectrum of IS (or sporadic) task systems, one may think that the optimality of PD^2 follows as a simple corollary from previous work. However, as explained in detail later, previously-presented proofs for Pfair and ERfair scheduling algorithms do not easily extend beyond the periodic task model. In this paper, we provide a new approach for dealing with Pfair- or ERfair-scheduled systems and use it to show that IS tasks can be optimally scheduled on multiprocessors. Since the presentation of this work as a conference paper [18], this approach has been used as a basis for proving a number of other results about fair-scheduled multiprocessor systems [2,8,11,13,12,19,20]. In addition to presenting a fundamentally new proof approach, this paper breaks new ground by being the first to show that sporadic or IS tasks can be optimally scheduled on systems of more than two processors.

In the rest of this paper, we present needed definitions (Section 2), describe the PD^2 algorithm (Section 3), prove that PD^2 optimally schedules IS tasks (Section 4), and then conclude (Section 5). A few technical results are proved in an appendix.

2 Definitions

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

2.1 Pfair and ERfair 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

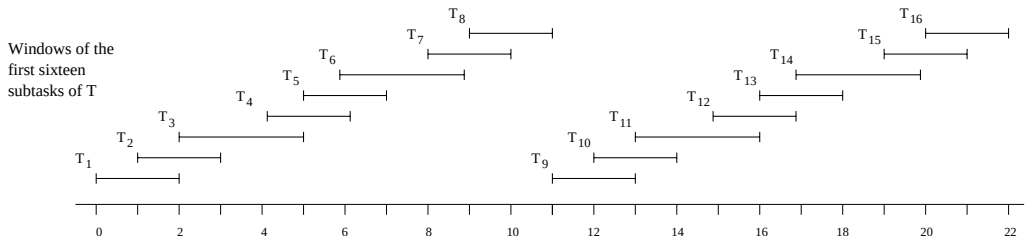

Fig. 1. The Pfair windows of the first two jobs (or sixteen subtasks) of a task T with weight 8/11 in a Pfair-scheduled system. During each job of T, each of the eight units of computation must be allocated processor time during its window, or else a lag-bound violation will result.

is allowed but each task must execute sequentially). The sequence of allocation decisions over time defines a *schedule* S. Formally, $S\colon \tau \times \mathcal{N} \mapsto \{0,1\}$, where τ 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, $\sum_{T\in\tau} S(T,t) \leq M$ holds 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*, denoted $lag(T,t)$, 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 ith subtask of task T as T_i, where $i \geq 1$. As in [6], 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-.")

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

$$d(T_i) = \left\lceil \frac{i}{wt(T)} \right\rceil \tag{3}$$

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 Figure 1). 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 Figure 1. Here, we have $r(T_2) = 1$,

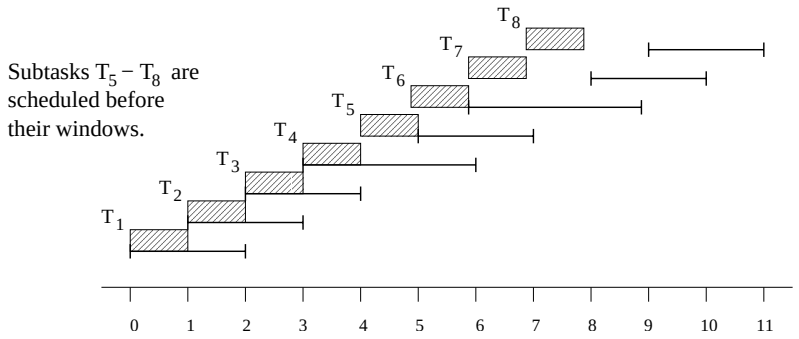

Fig. 2. The Pfair windows of the first job of a task T with weight 8/11 are shown. The schedule shown is ERfair, but not Pfair.

$d(T_2) = 3$, 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.)

The notion of ERfair scheduling [3] is obtained by simply dropping the -1 constraint in (1). With this change, a subtask can become eligible before its Pfair window. This is illustrated in Figure 2. Note that any Pfair schedule is ERfair, but not necessarily vice versa. It is easy to show that, in any Pfair or ERfair schedule, all job deadlines are met [3].

2.2 The Intra-sporadic Task Model

As noted earlier, the sporadic model generalizes the periodic model by allowing jobs to be released late. The IS model generalizes this notion further by allowing *subtasks* to be released late, as illustrated in Figure 3. More specifically, the separation between subtask releases $r(T_i)$ 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 separation if T were periodic (refer to Equation (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. Each subtask of an IS task has an *offset* that gives the amount by which its window has been right-shifted. The offset of subtask T_i is denoted $\theta(T_i)$. By (2) and (3), 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.

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

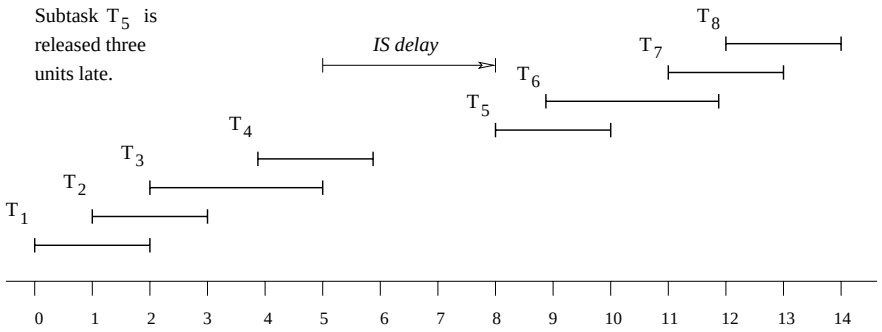

Fig. 3. The PF-windows of the first eight subtasks of an IS task T with weight 8/11. Subtask T_5 is released three time units late causing all later subtask releases to be delayed by three time units.

Because $\left\lfloor \frac{i}{wt(T)} \right\rfloor \geq \left\lceil \frac{i}{wt(T)} \right\rceil - 1$, by (4), $r(T_{i+1}) \geq \theta(T_{i+1}) + \left\lceil \frac{i}{wt(T)} \right\rceil - 1$. Hence, by (5) and (6), it follows that

$$r(T_{i+1}) \geq d(T_i) - 1. \tag{7}$$

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. 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. (This is not shown in Figure 3.) We refer to the interval $[r(T_i), d(T_i))$ as the *PF-window* of T_i and the interval $[e(T_i), d(T_i))$ as its *IS-window*. Since $e(T_i) \leq r(T_i)$, a subtask's PF-window is contained within its IS-window. Inequality (7) implies that PF-windows of consecutive subtasks of a task overlap by at most one slot. Henceforth, whenever the term "window" is used without qualification, it is meant to refer to a task's "PF-window."

The validity of a schedule for an IS task system is given by the definition below.

Definition 1 *A* valid *schedule for an IS task system is one that satisfies the following properties:* **(i)** *each subtask is scheduled in its IS-window,* **(ii)** *two subtasks of the same task are not scheduled in the same slot, and* **(iii)** *the number of subtasks scheduled in any slot is at most the number of processors.* □

Note that the notion of a job is secondary to the notion of a subtask in IS task systems. For systems in which subtasks are grouped into jobs that are released in sequence, the definition of e would preclude a subtask from becoming eligible before the beginning of its job. Using the definitions above, it is easy to show that sporadic and periodic tasks are special cases of IS tasks. In particular, a sporadic task T is an IS task in which only the first subtask of each job may be released late, *i.e.*, if T_i and T_{i+1} are part of the same job, then $\theta(T_i) = \theta(T_{i+1})$. A periodic task T is an IS task such that only the very first subtask of each

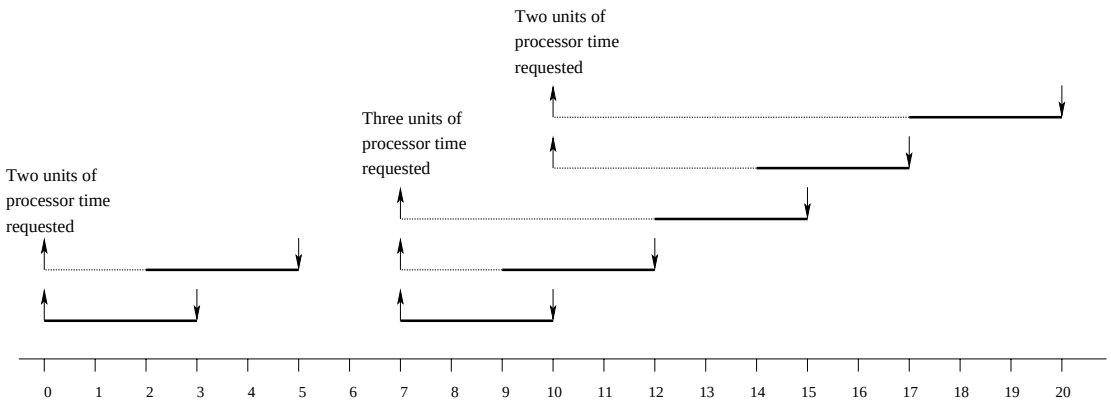

Fig. 4. The up arrows corresponds to subtask eligibility times and down arrows correspond to subtask deadlines. The dotted lines are used to illustrate IS-windows and the bold lines are used to illustrate the PF-windows. A server with weight 2/5 is shown. It receives requests of two units of processor time at times 0 and 10, and a request of three units of processor time at time 7.

task may be released late, *i.e.*, $\theta(T_i) = \theta(T_1)$ for all $i \geq 1$. (In Section 2.1, we assumed $\theta(T_1) = 0$.) Note that, by defining the function e appropriately, we can obtain eligibility intervals (*i.e.*, IS-windows) like those in either a Pfair or ERfair system. In fact, we can define eligibility intervals (*i.e.*, IS-windows) that are longer than in a Pfair system but shorter than in an ERfair system.

In [4], we proved that an IS task system τ has a valid schedule on M processors (*i.e.*, is *feasible*) iff

$$\sum_{T \in \tau} \frac{T.e}{T.p} \leq M. \tag{8}$$

The feasibility proof actually shows that a valid schedule exists in which each subtask is scheduled in its *PF-window*. (This fact will be of importance when we consider lags in IS task systems later in Section 4.2.)

Usefulness of the IS task model. Figure 4 illustrates an example server task that reserves a processor share of 2/5 (given by its weight) and receives client requests requiring two or three units of processor time. As seen in the figure, the IS model allows this functionality to be modeled easily, so that the server's request size is decoupled from its service rate.

The IS model also allows the instantaneous rate of subtask releases to differ greatly from the corresponding task's average rate (given by its weight). Hence, it is more suitable than the periodic model for several applications in networking. Examples include web servers that provide quality-of-service guarantees, packet scheduling in networks, and the scheduling of packet-processing activities in routers [21]. Due to network congestion and other factors, packets may arrive late or in bursts. The IS model treats these possibilities as first-class

concepts and handles them more seamlessly. In particular, a late packet arrival corresponds to an IS delay. On the other hand, if a packet arrives early (as part of a bursty sequence), then its eligibility time will be less than its Pfair release time. Note that its Pfair release time determines its deadline. Thus, in effect, an early packet arrival is handled by postponing its deadline to where it would have been had the packet arrived on time.

Relation to the RBE task model. In the uniprocessor *rate-based execution* (RBE) model [14], each task is characterized by four parameters: (x, y, d, c). A task is *expected* to release x jobs every y time units; each job has an execution cost of c and a relative deadline of d. In the IS model, a task with parameters (e, p) is *expected* to release e subtasks every p time units; each subtask has an execution cost of one and a relative deadline of approximately p/e. An RBE task may release more than x jobs every y time units, but the deadlines of jobs released early are postponed in a way that ensures the system is still feasible. Deadlines of early IS subtasks are similarly postponed using (4) and (5).

3 Algorithm PD^2

PD^2 prioritizes subtasks by their deadlines. Any ties are broken using two tie-break parameters, the "b-bit" (or "successor bit"), and the "group deadline." These parameters are defined next.

The successor bit. The *successor bit* for a subtask T_i is defined as follows.

$$b(T_i) = \left\lceil \frac{i}{wt(T)} \right\rceil - \left\lfloor \frac{i}{wt(T)} \right\rfloor \tag{9}$$

Thus, $b(T_i)$ is either 0 or 1. In a periodic task system, $b(T_i)$ denotes the number of slots by which T_{i+1}'s window overlaps T_i's window (see (2) and (3)). For example, in Figure 1(a), $b(T_i) = 1$ for $1 \leq i \leq 7$ and $b(T_8) = 0$.

The group deadline. It can be shown that all windows of a heavy task are of length two or three [17] (see Figure 1). Consider a sequence $T_i, \ldots, T_j$ of subtasks of a heavy task T (without any late releases) such that $|w(T_k)| = 2$ for all $i < k \leq j$, $b(T_k) = 1$ for all $i \leq k < j$, and either $b(T_j) = 0$ or $|w(T_{j+1})| = 3$ (*e.g.*, T_1, T_2 or T_3, T_4, T_5 or T_6, T_7, T_8 in Figure 1). If any of $T_i, \ldots, T_j$ is scheduled in the last slot of its window, then each subsequent subtask in this sequence must be scheduled in its last slot. In effect, $T_i, \ldots, T_j$ must be considered as a single schedulable entity subject to a "group" deadline. Formally, we define the *group deadline* for the subtasks $T_i, \ldots, T_j$ to be $d(T_j)$ if $b(T_j) = 0$, and $d(T_j) + 1$ if $|w(T_{j+1})| = 3$. Intuitively, if we imagine a job of T in which each subtask is scheduled in the first slot of its window, then

the remaining empty slots exactly correspond to the group deadlines of T. For example, in Figure 1, T has group deadlines at slots 3, 7, and 10.

We let $D(T_i)$ denote the group deadline of subtask T_i. Formally, if T is heavy, then $D(T_i) = (\textbf{min}\ u :: u \geq d(T_i)$ and u is a group deadline of $T)$. For example, in Figure 1, $D(T_1) = 3$ and $D(T_6) = 10$. If T is light, then $D(T_i) = 0$. For an IS task, the group deadline is defined in the same way, assuming that all the future subtasks are released as early as possible. In an IS task system, the group deadline of a heavy task can be calculated using the following formula [17].

$$D(T_i) = \theta(T_i) + \left\lceil \frac{\left\lceil \left\lceil \frac{i}{wt(T)} \right\rceil \times (1 - wt(T)) \right\rceil}{1 - wt(T)} \right\rceil \tag{10}$$

Having explained the notion of a group deadline, we can now state the PD^2 priority definition.

PD^2 Priority Definition: Subtask T_i's priority at slot t is defined to be $(d(T_i), b(T_i), D(T_i))$, if it is eligible at t. Priorities are ordered using the following relation.

$$\begin{aligned}(d', b', D') \preceq (d, b, D) \equiv\ & [d < d'] \vee [(d = d') \wedge (b > b')] \\ & \vee\ [(d = d') \wedge (b = b') \wedge (D \geq D')]\end{aligned}$$

If T_i and U_j are both eligible at t, then T_i's priority is at least U_j's at t if $(d(U_j), b(U_j), D(U_j)) \preceq (d(T_i), b(T_i), D(T_i))$. □

According to the definition above, T_i has higher priority than U_j if it has an earlier deadline. If T_i and U_j have equal deadlines, but $b(T_i) = 1$ and $b(U_j) = 0$, then the tie is broken in favor of T_i. This is because the window of T_i may overlap with that of its successor, and hence *not* scheduling it may reduce the number of slots available for its successor by one, constraining the future schedule. If T_i and U_j have equal deadlines and b-bits, then their group deadlines are inspected to break the tie. If one is heavy and the other light, then the tie is broken in favor of the heavy task (by the definition of the group deadline). If both are heavy and their group deadlines differ, then the tie is broken in favor of the one with the later group deadline. Note that the subtask with the later group deadline can force a longer cascade of scheduling decisions in the future. Thus, choosing to schedule such a subtask early places fewer constraints on the future schedule. Any ties not resolved by PD^2 can be broken arbitrarily.

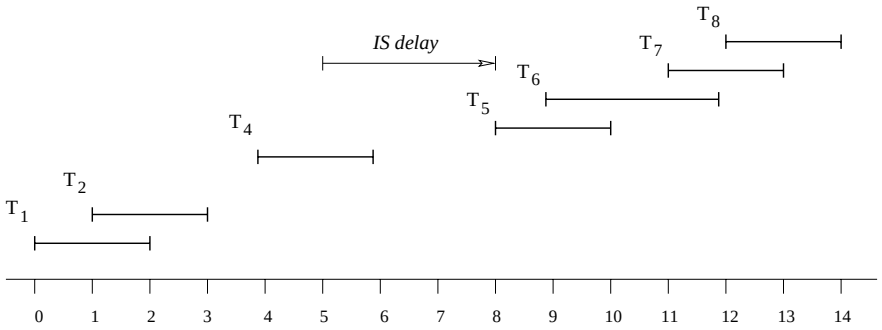

Fig. 5. The PF-windows of the first eight subtasks of a GIS task T with weight 8/11. Subtask T_3 is missing and T_5 is released three time units late. (Because T_3 is missing, this is not an IS task.)

4 Proof of Optimality of PD^2

In our proof, we consider task systems obtained by removing subtasks from an IS task system. Note that such a task system may no longer be an IS task system (see Figure 5). To circumvent this problem, we define a more general model called the *generalized* IS (GIS) task model, and show that PD^2 can optimally schedule task systems that belong to this model. 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 $\left\lfloor \frac{k-1}{wt(T)} \right\rfloor - \left\lfloor \frac{i-1}{wt(T)} \right\rfloor$. In other words, $r(T_k)$ (and hence, $d(T_k)$) is not smaller than what it would have been if $T_{i+1}, T_{i+2}, \ldots, T_{k-1}$ were present and released as early as possible. For the special case where T_k is the first subtask released by T, $r(T_k)$ is at least $\left\lfloor \frac{k-1}{wt(T)} \right\rfloor$.

Thus, the GIS model generalizes the IS model by allowing subtasks to be absent. It follows that for every GIS task system τ, there exists an IS task system τ' such that τ can be obtained by simply removing certain subtasks in τ'. Hence, if there exists a schedule for τ' in which no deadline is missed, then that schedule can be easily modified (by removing subtasks) to obtain a schedule for τ. Therefore, Expression (8) is a feasibility condition for GIS task systems as well.

Note that subtask indices for a GIS task are assigned to reflect the missing subtasks. For example, task T in Figure 5 releases subtask T_4 after releasing T_2; T_3 is missing and $\theta(T_4) = 0$. Hence, the formulae for subtask release times and deadlines of a GIS task are as in (4) and (5). Further, the formulae for the b-bit and group deadlines are also as defined in (9) and (10). This implies that the PD^2 priority definition of a subtask of a GIS task is the same as for the corresponding IS task.

Terminology. An *instance* of a task system is obtained by specifying a unique

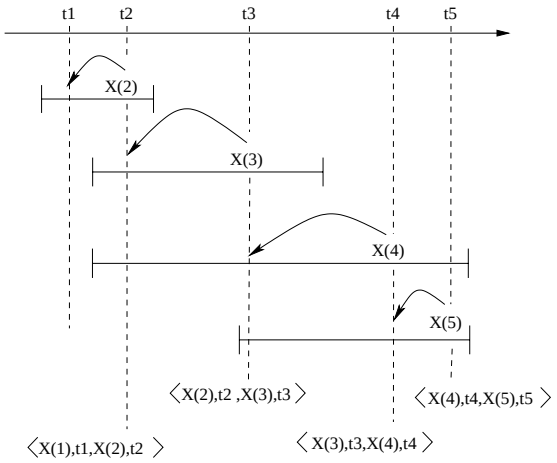

Fig. 6. A chain of four displacements, caused by removing $X^{(1)}$, which was scheduled in slot t_1.

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 Figure 5). The following property is used in our proofs.

Claim 1 *If subtask T_k is the successor of subtask T_i, then $r(T_k) \geq d(T_i) - 1$.*

Proof. Note that $\left\lceil \frac{i}{wt(T)} \right\rceil \leq \left\lfloor \frac{i}{wt(T)} \right\rfloor + 1$. Because $k \geq i + 1$, $\left\lfloor \frac{k-1}{wt(T)} \right\rfloor \geq \left\lfloor \frac{i}{wt(T)} \right\rfloor$. Therefore, $\left\lfloor \frac{k-1}{wt(T)} \right\rfloor \geq \left\lceil \frac{i}{wt(T)} \right\rceil - 1$. By (6), $\theta(T_k) \geq \theta(T_i)$. Therefore, $\theta(T_k) + \left\lfloor \frac{k-1}{wt(T)} \right\rfloor \geq \theta(T_i) + \left\lceil \frac{i}{wt(T)} \right\rceil - 1$. By (4) and (5), this implies that $r(T_k) \geq d(T_i) - 1$. □

4.1 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 τ. Let S denote a schedule of τ obtained by any scheduling algorithm (such as PD^2) that schedules on an earliest-pseudo-deadline-first (EPDF) basis. Assume that removing $X^{(1)}$, scheduled in 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 Figure 6.

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

The lemmas below concern displacements and holes. Lemma 1 states that a subtask removal can only cause left-shifts, as in Figure 6. 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, τ is an instance of a GIS task system and S denotes a schedule for τ obtained by a greedy EPDF-based scheduling algorithm. Throughout this section, we assume that ties among subtasks are resolved consistently, *i.e.*, if τ' is obtained from τ by a subtask removal, then the relative priorities of two subtasks in τ' are the same as in τ.

Lemma 1 *Let $X^{(1)}$ be a subtask that is removed from τ, 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 $i \in \{1, \ldots, k\}$.*

Proof. Let τ' be the task system instance obtained by removing $X^{(1)}$ from τ, and let S' be its PD^2 schedule. Note that the last displacement creates a hole at t_{k+1} in S'. Suppose $t_{i+1} \leq t_i$ for some $i \in \{1, \ldots, k\}$. Let

$$t_j = min\{t_i \mid t_{i+1} < t_i \ \wedge \ 1 \leq i \leq k\}.$$

(Informally, the leftmost right-shift occurs when $X^{(j+1)}$ scheduled at t_{j+1} shifts to t_j.) We consider two cases depending on whether j is equal to k.

If $j = k$, then the last displacement is as shown in Figure 7(a). Note that $X^{(k+1)}$ is eligible to be scheduled in slot t_{k+1} in S', because it is scheduled there in S and no subtask (in particular, its predecessor) scheduled before t_{k+1} is shifted to t_{k+1} (by the choice of j). Because there is a hole in slot t_{k+1} in S' and $t_{k+1} < t_k$, this contradicts the greedy behavior of the scheduling algorithm.

If $j < k$, then by our choice of j, $t_{j+1} < t_j$ and the displacements are as in Figure 7(b). By the minimality of t_j, $t_{j+2} > t_{j+1}$. Thus, at t_{j+1}, $X^{(j+1)}$ is chosen over $X^{(j+2)}$ in S. After the displacements, $X^{(j+1)}$ is scheduled at t_j and $X^{(j+2)}$ at $t_{j+1}(< t_j)$. This contradicts our assumption that ties are broken consistently in S and S'. Hence, $t_{i+1} > t_i$ 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 $X^{(1)}$.*

Proof. Because Δ is valid, $e(X^{(2)}) \leq t_1$. Since there is a hole in slot t_1 and $X^{(2)}$ is not scheduled there in S, $X^{(2)}$ is the successor of $X^{(1)}$. □

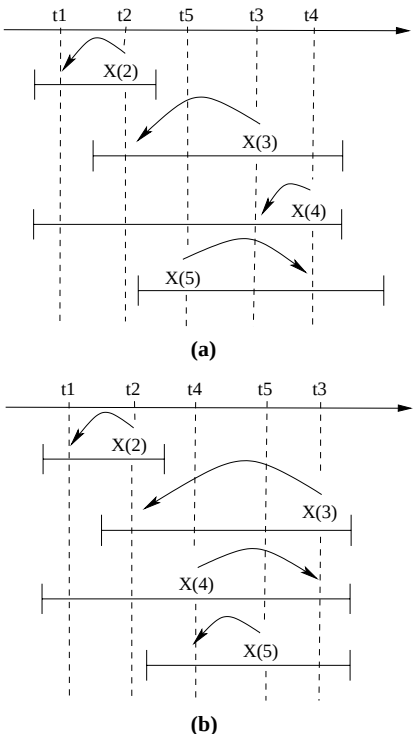

Fig. 7. Lemma 1. A chain of $k = 4$ displacements is shown. **(a)** The leftmost right shift occurs when $X^{(5)}$ shifts from t_5 to t_4, *i.e.*, $j = k$. **(b)** The leftmost right shift occurs when $X^{(4)}$ shifts from t_4 to t_3, *i.e.*, $j < k$ (here, $t_j = t_3, t_{j+1} = t_4$, and $t_{j+2} = t_5$).

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 $t_1 \leq t' < t_2$ in that schedule, then $t' = t_1$ and $X^{(2)}$ is the successor of $X^{(1)}$.*

Proof. Because Δ is valid, $e(X^{(2)}) \leq t_1$. 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 $X^{(1)}$. □

4.2 Flows and Lags in GIS Task Systems

The lag of an IS or GIS task at time t can be 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,

$$lag(T, t) = ideal(T, t) - \sum_{u=0}^{t-1} S(T, u). \tag{11}$$

For a periodic task that begins execution at time 0, $ideal(T, t) = (T.e/T.p)t$. To define $ideal(T, t)$ for an IS or GIS task, we consider the feasibility proof

given in [4]. There, a valid schedule is shown to exist by constructing a flow network with a certain real-valued flow. $ideal(T, t)$ is defined based on this flow:

$$ideal(T, t) = \sum_{u=0}^{t-1} flow(T, u). \tag{12}$$

Here, $flow(T, u)$ is the flow (or share) assigned to task T in slot u. We formally define $flow(T, u)$ below. For motivation, consider a task of weight 5/16. In any valid schedule, each subtask of this task receives a share of one unit processor time over its IS-window. In the ideal system, each subtask gets a share of 5/16 in each slot of its PF-window, except maybe the first and last slots of the window. This is illustrated in Figure 8. Inset (a) shows the shares assigned in each slot of the PF-window for a periodic task of weight 5/16, and inset (b) shows the shares in each slot for an IS task of weight 5/16 in which some subtasks are released late. Note that the shares for each subtask sum to one (*e.g.*, $5/16 + 5/16 + 5/16 + 1/16 = 1$ for the first subtask). Also, note that the share in each slot is at most 5/16, the weight of the task. For the periodic task, the share in each slot is *exactly* 5/16, whereas for the IS task, it may be less (see slot 3 in inset (b)). In the flow network, each subtask has flows corresponding to these shares.

Formally, $flow(T, u)$ is defined in terms of a function f, which indicates the share assigned to each subtask T_i in each slot u. The function f is defined as follows.

$$f(T_i, u) = \begin{cases} \left(\left\lfloor \frac{i-1}{wt(T)} \right\rfloor + 1\right) \cdot wt(T) - (i-1), & u = r(T_i) \\ i - \left(\left\lceil \frac{i}{wt(T)} \right\rceil - 1\right) \cdot wt(T), & u = d(T_i) - 1 \\ wt(T), & r(T_i)+1 \leq u \leq d(T_i)-2 \\ 0, & \text{otherwise.} \end{cases} \tag{13}$$

For example, consider the last slot of the second subtask in Figure 8(b) and also the first slot of the third subtask. $f(T_2, d(T_2)) = f(T_2, 8)$, which by (13) equals $2 - (\lceil \frac{2}{5/16} \rceil - 1) \cdot (5/16) = 2/16$, as shown in the figure. Similarly, $f(T_3, r(T_3)) = f(T_3, 8)$, which by (13) equals $(\lfloor \frac{2}{5/16} \rfloor + 1) \cdot (5/16) - (3-1) = 3/16$. Note that these two flows sum to 5/16, the weight of the task.

The function $flow(T, u)$ is defined as $flow(T, u) = \sum_i f(T_i, u)$. The following properties about flows are used in our proof. (We only prove (F1) here to give a flavor of the proof technique used. The other properties are proved in an appendix.)

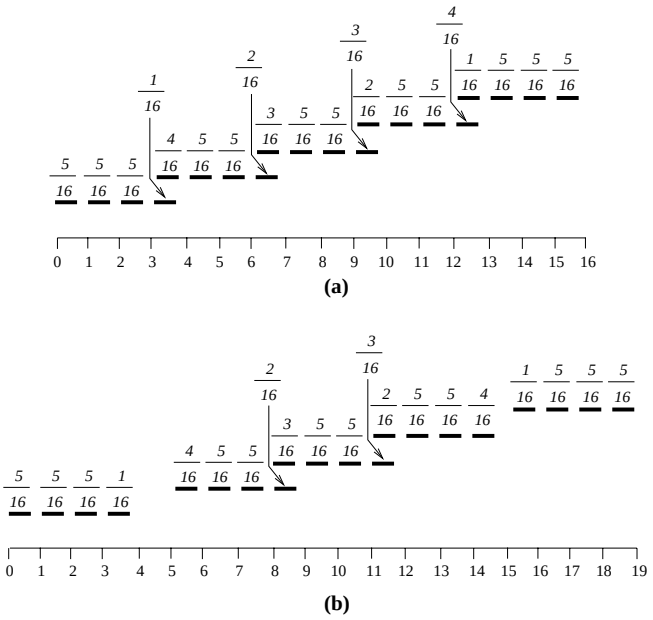

Fig. 8. Fluid schedule for a task T of weight 5/16. The share of each subtask in the slots of its window is shown. In **(a)**, no subtask is released late; in **(b)**, T_2 and T_5 are released late. Note that $flow(T, 3)$ is either 5/16 or 1/16 depending on when subtask T_2 is released.

(F1) For all time slots t, $\mathit{flow}(T, t) \leq wt(T)$.

Proof: We first show that $f(T_i, t) \leq wt(T)$. This follows directly from (13) if $t \notin \{r(T_i), d(T_i) - 1)\}$. If $t = r(T_i)$, then

$$\begin{aligned} f(T_i) &= \left(\left\lfloor \frac{i-1}{wt(T)} \right\rfloor + 1 \right) \times wt(T) - (i-1) \quad , \quad \text{by (13)} \\ &\leq \left(\frac{i-1}{wt(T)} + 1 \right) \times wt(T) - (i-1) \quad , \quad \lfloor x \rfloor \leq x \\ &= wt(T) \quad , \quad \text{by simplification} \end{aligned}$$

If $t = d(T_i) - 1$, then

$$\begin{aligned} f(T_i) &= i - \left(\left\lceil \frac{i}{wt(T)} \right\rceil - 1 \right) \times wt(T) \quad , \quad \text{by (13)} \\ &\leq i - \left(\frac{i}{wt(T)} - 1 \right) \times wt(T) \quad , \quad \lceil x \rceil \geq x \\ &= wt(T) \quad , \quad \text{by simplification} \end{aligned}$$

We now only need to consider the time slot in which two consecutive PF-windows overlap. That is the case when $d(T_i) - 1 = r(T_{i+1})$ for some i. In this case, the total flow is $f(T_i, d(T_i) - 1) + f(T_{i+1}, r(T_{i+1}))$. Thus, $\mathit{flow}(T, d(T_i) - 1)$ is $i - \left(\left\lceil \frac{i}{wt(T)} \right\rceil - 1 \right) \times wt(T) + \left(\left\lfloor \frac{i}{wt(T)} \right\rfloor + 1 \right) \times wt(T) - i$, which

simplifies to $\left(\left\lfloor \frac{i}{wt(T)} \right\rfloor - \left\lceil \frac{i}{wt(T)} \right\rceil + 2\right) \times wt(T)$. Since $d(T_i) - 1 = r(T_{i+1})$, by (4) and (5), it follows that $\theta(T_i) = \theta(T_{i+1})$ and $\left\lceil \frac{i}{wt(T)} \right\rceil - 1 = \left\lfloor \frac{i}{wt(T)} \right\rfloor$. Therefore, $\left\lfloor \frac{i}{wt(T)} \right\rfloor - \left\lceil \frac{i}{wt(T)} \right\rceil = -1$. Hence, $\mathit{flow}(T, d(T_i) - 1) = wt(T)$. Thus, in all cases, we have $\mathit{flow}(T, t) \leq wt(T)$. □

(F2) Let T_i be a subtask of a GIS task and let T_k be its successor. If $b(T_i) = 1$ and $r(T_k) \geq d(T_i)$, then $\mathit{flow}(T, d(T_i) - 1) + \mathit{flow}(T, d(T_i)) \leq wt(T)$. (For example, in Figure 8(b), $\mathit{flow}(T, 3) + \mathit{flow}(T, 4) = 1/16 < 5/16$ and $\mathit{flow}(T, 14) + \mathit{flow}(T, 15) = 5/16$.)

(F3) Let T_i be a subtask of a heavy GIS 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 $\mathit{flow}(T, d(T_i)) + \mathit{flow}(T, u) \leq wt(T)$. (This is an extension of (F2) to heavy tasks.)

From (11) and (12), we get

$$\begin{aligned} lag(T, t+1) &= \sum_{u=0}^{t} (\mathit{flow}(T, u) - S(T, u)) \\ &= lag(T, t) + \mathit{flow}(T, t) - S(T, t). \end{aligned} \tag{14}$$

Similar to the notion of *lag* for tasks, we can define the total lag of a task system. The total lag for a schedule S and task system τ at time $t+1$, denoted by $LAG(\tau, t+1)$, is defined as follows.

$$LAG(\tau, t+1) = LAG(\tau, t) + \sum_{T \in \tau} (\mathit{flow}(T, t) - S(T, t)) \tag{15}$$

$LAG(\tau, 0)$ is defined to be 0. Note that the definitions of *lag* and *LAG* do not make any assumptions about the validity of the corresponding schedule. The lemma below is used in later proofs.

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

Proof. Let k be the number of subtasks scheduled in slot t. Then, by (15), $LAG(\tau, t+1) = LAG(\tau, t) + \sum_{T \in \tau} \mathit{flow}(T, t) - k$. If $LAG(\tau, t) < LAG(\tau, t+1)$, then $k < \sum_{T \in \tau} \mathit{flow}(T, t)$. Because $\mathit{flow}(T, t) \leq wt(T)$ (by (F1)), we have $\sum_{T \in \tau} \mathit{flow}(T, t) \leq \sum_{T \in \tau} wt(T)$, which by (8) implies that $\sum_{T \in \tau} \mathit{flow}(T, t) \leq M$. Therefore, $k < M$, *i.e.*, there is a hole in slot t. □

Because periodic job releases represent a "worst-case" scenario for an IS task, one may think that the optimality of PD^2 for IS tasks follows as a simple corollary from previous work. One proof technique that has been used in prior work is a "swapping" argument wherein an arbitrary schedule is systematically converted to one in accordance with PD^2 by swapping subtasks that violate the PD^2 priority definition [5]. To ensure that multiple subtasks of the same task are not scheduled at the same time, needed valid swappings may involve many subtasks of many tasks. To do this correctly, it is *crucial* that at any moment of time, future window alignments can be predicted. However, such predictions cannot be made for IS task systems.

Another approach that has been often used with uniprocessor scheduling algorithms is to reduce sporadic systems to periodic systems in the following way. Consider a scheduling algorithm A that has been shown to be correct for periodic tasks. Suppose that there exists a feasible sporadic task system τ that misses a deadline at some time t_d when scheduled using A. Let S be the corresponding schedule. We may assume that all jobs in S after t_d are released in a periodic fashion, because such jobs have no impact on the deadline miss at time t_d. Now, if we inductively "right-shift" all jobs released before time t_d in S until there are no sporadic separations among jobs before t_d, then we get a schedule S' that is in accordance with the periodic task model (see Figure 9(a)). Moreover, "right-shifting" such jobs in S can only increase demand near time t_d. Thus, a deadline is missed at time t_d, a contradiction.

To see why this argument cannot be applied in Pfair-scheduled multiprocessor systems, consider the situation shown in Figure 9(b). Here, subtask T_i is right-shifted into slot t. Before the shift, subtask U_j was scheduled at t and some processor was idle at t. After the shift, U_j has higher priority than T_i, so the two are swapped in the schedule as a result of shifting T_i. Note that U_j being scheduled at t makes U_{j+1} ineligible at t. However, after T_i and U_j are swapped, U_{j+1} *is* eligible at t and thus it may left-shift into slot t. This may cause a cascade of other left-shifts, which in turn can cause a presumed (future) missed deadline to be met. The root of the problem here is that right-shifting certain subtasks may in fact *reduce* demand in the future.

In the next subsection, we present a new, lag-based proof to establish the optimality of PD^2 for IS task systems.

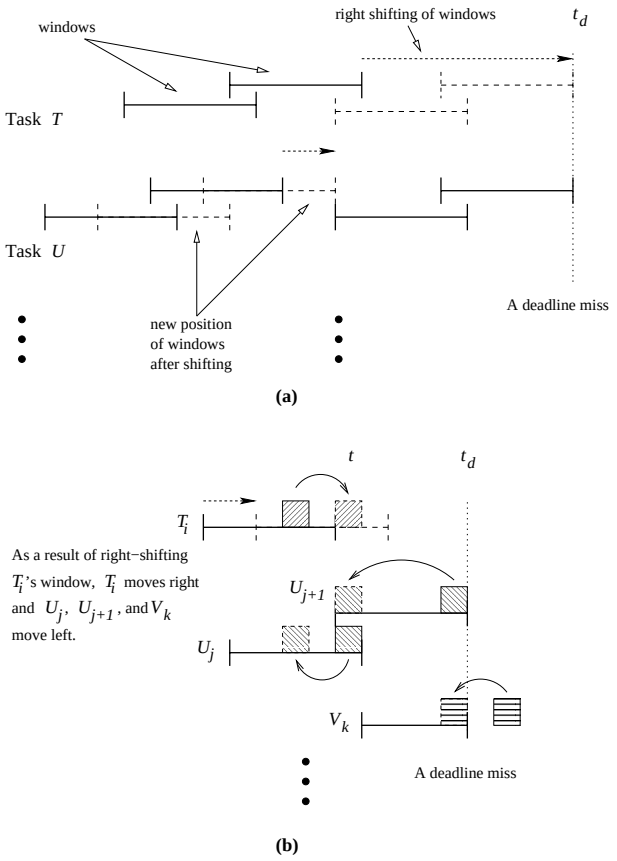

Fig. 9. The following notation is used in this figure. A right arrow over a window corresponds to a right-shift of that window. A "dashed" window is used to depict the window's new position after the shift. Subtasks are denoted by rectangles. An arrow over a subtask indicates the direction it is displaced as a result of a shift. Such a subtask's new position is indicated by a corresponding "dashed" rectangle. **(a)** Starting with a sporadic task set that misses a deadline at t_d, we can right-shift all windows towards t_d. Intuitively, we *should* get a periodic task system that misses a deadline at t_d. **(b)** Unfortunately, right-shifting a window need not increase future demand. Here, shifting T_i to the right by two slots decreases its priority and hence subtask U_j, which was scheduled later at time t in the original schedule, may now have higher priority. Note that at time t, a processor might be idle, in which case U_{j+1} can now be scheduled in that slot. This can cause a cascade of future left-shifts. Thus, right-shifting a window can in fact *decrease* future demand.

4.4 Correctness Proof

We now show that PD^2 correctly schedules any GIS task system. In particular, we prove that the following assumption leads to a contradiction: PD^2 misses a deadline for some feasible GIS task system. This assumption implies the existence of a time t_d and an instance of a task system τ as given by Definition 2 and Definition 3 below.

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

Definition 3 *τ is an instance of a task system with the following properties.*

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

Existence of τ follows from the fact that (T1)–(T3) are applied *in sequence*; *e.g.*, τ is not claimed to be of maximal rank — rather, its rank is maximal *among those task system instances satisfying* (T1) *and* (T2).

By (T1), (T2), and Definition 2, exactly one subtask in τ misses its deadline: if several subtasks miss their deadlines, all but one can be removed and the remaining subtask still misses its deadline, contradicting (T2).

In the rest of this proof, we use S to denote the PD^2 schedule of τ. We now prove several properties about τ and S.

Lemma 5 *The following properties hold for τ and S.*

(a) *For all subtasks T_i in τ, $e(T_i) \geq min(r(T_i), t)$, where t is the time at which T_i is scheduled in S. (Because $e(T_i) \leq r(T_i)$ and $e(T_i) \leq t$, this property actually implies $e(T_i) = min(r(T_i), t)$.)*
(b) *Let t be the time at which T_i is scheduled and let T_k be T_i's successor. If either $d(T_i) > t + 1$ or $d(T_i) = t + 1 \ \wedge \ b(T_i) = 0$, then T_k 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. Below, we prove each property separately.

Proof of (a): Suppose that $e(T_i) < min(r(T_i), t)$. Consider the task system instance τ' obtained from τ by changing $e(T_i)$ to $min(r(T_i), t)$. Note that $e(T_i)$ is still at most $r(T_i)$ and τ''s rank is larger than τ's. τ' is feasible because the feasibility proof produces a schedule in which each subtask is scheduled in its PF-window (refer to Section 2.2).

Since PD^2 priorities do not depend on eligibility times, it is easy to see that the relative priorities of the subtasks do not change for any slot $u \in \{0, \ldots, t_d - 1\}$.

Hence, τ' and τ have identical PD^2 schedules. Thus, τ' misses a deadline at t_d, contradicting (T3).

Proof of (b): By Claim 1, $r(T_k) \geq d(T_i) - 1$. Therefore, if $d(T_i) > t+1$ or $r(T_k) \geq d(T_{i+1})$, then $r(T_k) \geq t+1$. Further, since T_i is scheduled in $[t, t+1)$, T_k is scheduled at or after time $t+1$. Therefore, by part (a), $e(T_k) \geq t+1$.

We now consider the case when $d(T_i) = t+1$. Since $b(T_i) = 0$, by (9), it follows that $\left\lfloor \frac{i}{wt(T)} \right\rfloor = \left\lceil \frac{i}{wt(T)} \right\rceil$. Therefore, by (4) – (6), $r(T_{i+1}) \geq d(T_i)$. Therefore, $r(T_j) \geq t+1$, for all $j > i$. In particular, $r(T_k) \geq t+1$, where T_k is T_i's successor. As before, by part (a), it follows that $e(T_k) \geq t+1$.

Proof of (c): Suppose τ contains a subtask U_j with a deadline greater than t_d. Since S is obtained using an EPDF-based scheduling algorithm, U_j can be removed without affecting the scheduling of higher-priority subtasks with earlier deadlines. Thus, a deadline is still missed at t_d after U_j's removal. This contradicts (T2).

Proof of (d): Let U_j be the subtask that misses its deadline at t_d in S. (Recall that there is only one such subtask.) Because $d(U_j) = t_d$, $d(U_k) \leq t_d - 1$, where U_k is U_j's predecessor (if it exists). By the minimality of t_d, U_k meets its deadline and hence is scheduled before $t_d - 1$. Thus, if there is a hole in slot $t_d - 1$, then U_j is scheduled there, in which case it meets its deadline. Contradiction.

Proof of (e): By (15), 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 τ. The second term equals the number of subtasks scheduled in S over the interval $[0, t_d)$. Since exactly one subtask misses its deadline in S, 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 5, there exists a time t ($< t_d - 1$) such that $LAG(\tau, t) < 1$ and $LAG(\tau, t+1) \geq 1$. Without loss of generality, let t be the latest such time. Thus, we have the following.

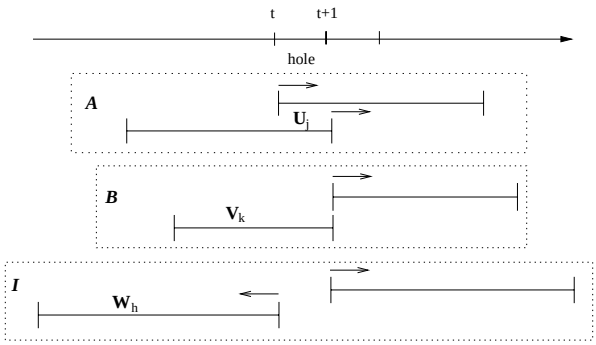

Fig. 10. Sets A, B, and I. The PF-windows of a sample task of each set are shown. The PF-windows are denoted by line segments. An arrow over a release (respectively, deadline) indicates that the release (respectively, deadline) could be anywhere in the direction of the arrow.

$$0 \leq t < t_d - 1 \ \wedge \ (LAG(\tau, t) < 1) \ \wedge \ (LAG(\tau, t+1) \geq 1) \\ \wedge \ (u \in [t+1, t_d] \Rightarrow LAG(\tau, u) \geq 1) \tag{16}$$

(Note that the last inequality partly follows from parts (e) and (f) of Lemma 5.) We now show that such a t cannot exist, thus contradicting our starting assumption that t_d and τ exist.

By (16), we have

$$LAG(\tau, t) < LAG(\tau, t+1). \tag{17}$$

Hence, by Lemma 4, *there is at least one hole in slot* t. We now group the tasks into sets A, B, and I (depending on how their subtask releases occur around time t).

(1) A denotes the set of tasks that are scheduled in slot t.
(2) B denotes the set of tasks not in A that are "active" at t. A task U is said to be *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 because of a late subtask release.)
(3) I denotes the set of the remaining tasks that are not active at time t.

Figure 10 shows how the tasks in A, B, and I are scheduled. Of these three sets, set B is the most interesting. As we show below, every task in B must have an IS separation in slot t. We use this to prove that LAG reduces below one before time t_d.

We now prove several properties of set B, and we start by showing that B is non-empty.

Lemma 6 $|B| > 1$.

Proof. Let the number of the holes in slot t be h. Then, $\sum_{T\in\tau} S(T,t) = M - h$. By (15), $LAG(\tau, t+1) = LAG(\tau, t) + \sum_{T\in\tau}(\mathit{flow}(T,t) - S(T,t))$. Thus, because $LAG(\tau,t) < LAG(\tau,t+1)$, we have $\sum_{T\in\tau} \mathit{flow}(T,t) > M - h$.

For every $V \in I$, since either $d(V_k) < t$ or $r(V_k) > t$ holds, by (13), $\mathit{flow}(V,t) = 0$. It follows that $\sum_{T\in A\cup B} \mathit{flow}(T,t) > M-h$. Thus, by (F1), $\sum_{T\in A\cup B} wt(T) > M - h$.

Because the number of tasks scheduled in slot t is $M-h$, $|A| = M-h$. Because $wt(T) \leq 1$ 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 Lemmas 7, 8, and 10, we use the following technique to prove the required result: 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 contradicting (T2).

Lemma 7 *Suppose there is a hole in slot $u \in \{0,\ldots,t_d-1\}$. Let U be a task that is not scheduled in slot u, but is active at u. Further, let U_j be the subtask with the largest index such that $e(U_j) \leq u < d(U_j)$. Then, $d(U_j) = u+1 \ \wedge \ b(U_j) = 1$.*

Proof. Because there is a hole in slot u and no subtask of U is scheduled at time u, and because $e(U_j) \leq t < d(U_j)$, U_j is scheduled before time u. Because $u < t_d$, by Definition 2, U_j meets its deadline. From the lemma statement, we have $d(U_j) \geq u+1$. Suppose that the following holds.

$$d(U_j) > u+1 \text{ or } d(U_j) = u+1 \ \wedge \ b(U_j) = 0 \tag{18}$$

Under these assumptions, we show that U_j can be removed and a deadline is still 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 subtask $X^{(i)}$ 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 (18), for each subtask $X^{(i)}, 1 \leq i \leq k+1$, either $d(X^{(i)}) > u+1$ or $d(X^{(i)}) = u+1 \ \wedge \ b(X^{(i)}) = 0$. Therefore, by part (b) of Lemma 5, the following property holds.

(E) The eligibility time of the successor of $X^{(i)}$ (if it exists in τ) is at least $u+1$ for all $i \in \{1,\ldots,k+1\}$.

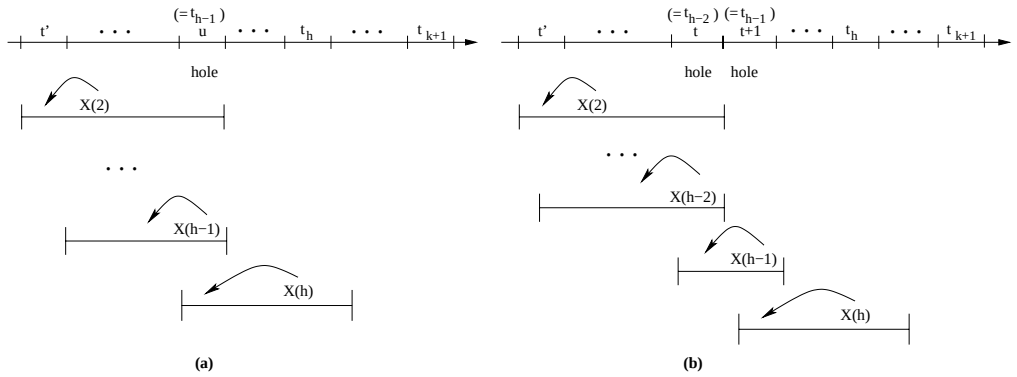

Fig. 11. **(a)** Lemma 7. IS-windows are denoted by line segments. $X^{(h)}$ must be the successor of $X^{(h-1)}$ because there is a hole in slot u. **(b)** Lemma 8. If there is a hole in both slots t and $t+1$, then $X^{(h-2)}$ and $X^{(h-1)}$ must be scheduled at t and $t+1$ in S, respectively. Also, $X^{(h)}$ must be the successor of $X^{(h-1)}$, which in turn, must be the successor of $X^{(h-2)}$.

We now show that the displacements do not extend beyond slot u. Assume, to the contrary, that $t_{k+1} > u$. Consider $h \in \{2, \ldots, k+1\}$ such that $t_h > u$ and $t_{h-1} \leq u$, as depicted in Figure 11(a). Such an h exists because $t_1 < u < t_{k+1}$. Because there is a hole in slot u and $t_{h-1} \leq u < t_h$, by Lemma 3, $t_{h-1} = u$ and $X^{(h)}$ must be $X^{(h-1)}$'s successor. Therefore, by (E), $e(X^{(h)}) \geq u+1$. This implies that Δ_{h-1} is not valid.

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

The following corollary directly follows by applying Lemma 7 to slot t and tasks in set B.

Corollary 1 *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, $d(U_j) = t + 1 \ \wedge \ b(U_j) = 1$.*

We now consider two separate cases depending on whether B contains a light task.

4.4.1 At Least One Task in B is Light

The following property (proved in the appendix) is used in the proof of Lemma 8.

(L) For a light task T, if T_k is the successor of T_i, then $d(T_k) \geq d(T_i) + 2$.

Lemma 8 *If B has at least one light task, then there is no hole in slot $t+1$.*

Proof. By (16), $t < t_d - 1$, and therefore, $t + 1 \leq t_d - 1$. Suppose that there is a hole in slot $t + 1$. By part (d) of Lemma 5, $t + 1 < t_d - 1$, *i.e.*,

$$t + 2 \leq t_d - 1. \tag{19}$$

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 7. 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 Corollary 1), this implies the following.

(P) For all $i \in \{1, \ldots, k+1\}$, 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 $t_h > t+1$ and $t_{h-1} \leq t+1$. Because there is a hole in slot $t+1$ and $t_{h-1} \leq t+1 < t_h$, 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 Figure 11(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 $t + 1$, and $d(X^{(h-1)}) > t + 2$, by part (b) of Lemma 5, the successor of $X^{(h-1)}$ is not eligible before $t + 2$, *i.e.*, $e(X^{(h)}) \geq t + 2$. This implies that the displacement Δ_{h-1} is not valid. Thus, the chain of displacements cannot extend beyond time $t + 2$. Hence, because $t + 2 \leq t_d - 1$ (by (19)), 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 9 *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 the definition of I, only tasks in $A \cup B$ are active at time t. Therefore, $\sum_{T \in \tau} \mathit{flow}(T, t) = \sum_{T \in A \cup B} \mathit{flow}(T, t)$. Since $wt(T) \leq 1$ for any T, we have $\sum_{T \in A} wt(T) \leq |A|$. Thus, by (F1), $\sum_{T \in A} \mathit{flow}(T, t) \leq |A|$. Now, because there are h holes in slot t, $M - h$ tasks are scheduled at t, *i.e.*, $|A| = M - h$. Thus,

$\sum_{T \in A} \mathit{flow}(T,t) \leq M - h$ and

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

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 Corollary 1,

$$\text{for all } U_j \in C,\ d(U_j) = t+1 \ \wedge\ b(U_j) = 1. \tag{21}$$

Let A' denote the tasks in A that are active at time $t+1$. Similarly, let I' denote the tasks in 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, because τ is feasible,

$$\sum_{T \in A' \cup I' \cup B} wt(T) \leq M. \tag{22}$$

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 (20),

$$\begin{aligned}\sum_{T \in \tau} (\mathit{flow}(T,t) + \mathit{flow}(T,t+1)) \leq M - h + &\sum_{T \in A' \cup I'} wt(T) \\ &+ \sum_{T \in B} (\mathit{flow}(T,t) + \mathit{flow}(T,t+1))\end{aligned} \tag{23}$$

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, by Lemma 5(a), $r(U_k) \geq t+1$. By (21), we have $d(U_j) = t+1$ and $b(U_j) = 1$. Therefore, by (F2), $\mathit{flow}(U,t) + \mathit{flow}(U,t+1) \leq wt(U)$ for each $U \in B$. By (23), 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 (22), it follows that

$$\sum_{T \in \tau} (\mathit{flow}(T,t) + \mathit{flow}(T,t+1)) \leq M - h + M. \tag{24}$$

By the statement of the lemma, B contains at least one light task. Therefore, by Lemma 8, 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 (24), $\sum_{T \in \tau} (\mathit{flow}(T,t) + \mathit{flow}(T,t+1)) \leq \sum_{T \in \tau} (S(T,t) + S(T,t+1))$. Using this relation in the identity (obtained from (15)), $LAG(\tau, t+2) =$

$LAG(\tau, t) + \sum_{T \in \tau}(\mathit{flow}(T, t) + \mathit{flow}(T, t+1)) - \sum_{T \in \tau}(S(T, t) + S(T, t+1))$, and the fact that $LAG(\tau, t) < 1$, we obtain $LAG(\tau, t+2) < 1$. □

4.4.2 All Tasks in B are Heavy.

We now extend Lemmas 8 and 9 to the case in which B consists solely of heavy tasks. The following lemma is the counterpart of Lemma 8.

Lemma 10 *Let U be a heavy task in B and let U_j be the subtask of U with the largest index such that $e(U_j) \leq t < d(U_j)$. Then, there exists a slot with no holes in $[d(U_j), min(D(U_j), t_d))$.*

Proof. By Corollary 1, $d(U_j) = t + 1 \wedge b(U_j) = 1$. By (16), $t < t_d - 1$. Therefore $d(U_j) \leq t_d - 1$. If $min(D(U_j), t_d) = t_d$, then by part (f) of Lemma 5, 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.*,

$$t + 1 < v. \tag{25}$$

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 $i \in \{1, \ldots, k-1\}$. 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 Corollary 1, for all $i \in \{2, \ldots, k+1\}$, one of the following holds:

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

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,

$$\text{for all } i \in [g+1, h-1],\ X^{(i+1)} \text{ is the successor of } X^{(i)}. \tag{26}$$

Note that since Δ_i is valid, we have $e(X^{(i+1)}) \leq t_i$. Hence, for any $i \in \{g+1, \ldots, h-2\}$, since there are holes in the interval $[t_i, t_{i+1})$, we have $t_{i+1} \leq t_i + 1$. (Otherwise, $X^{(i+1)}$ would have been scheduled before t_{i+1}.) By Lemma 1, we have $t_{i+1} = t_i + 1$.

Also, because there are holes in the interval $[t, t_{g+1})$ (by (H)) and $t_g < t$, we have $t_{g+1} = t$. Similarly, $t_{h-1} = v - 1$. (Otherwise, $X^{(h)}$ is scheduled at or before $v - 1$.) Thus, we have the following.

$$t_{g+1} = t \ \wedge\ t_{h-1} = v - 1 \tag{27}$$

$$\text{for all } i \in \{g+1, \ldots, h-1\},\ t_i = t + i - (g+1) \tag{28}$$

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 5, part (b), $e(X^{(g+2)}) \geq t + 1$ (recall that, by (26), $X^{(g+2)}$ is the successor of $X^{(g+1)}$). In other words, the displacement Δ_g is not valid. Therefore,

$$d(X^{(g+1)}) = t + 1 \wedge b(X^{(g+1)}) = 1 \wedge D(X^{(g+1)}) \leq v. \tag{29}$$

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 (25), $t + 1 \leq v - 1$ and hence, by (H), there is a hole in both t and $t + 1$. Also, by (27) and (28), we have $v - 1 = t + (h-1) - (g+1) = t + h - g - 2$. Because $t < v - 1$ (by (25)), 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 8 applies. Thus, the displacement Δ_{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 (29), $v' \leq v$. We now show that the displacements cannot extend beyond slot $v' - 1$ (and hence, slot $v - 1$). By (28), $X^{(i)}$ is scheduled in slot $t + i - (g+1)$ in S for all $i \in \{g+1, \ldots, h-1\}$. By (26), all $X^{(i)}$ where $g + 1 \leq i \leq h$

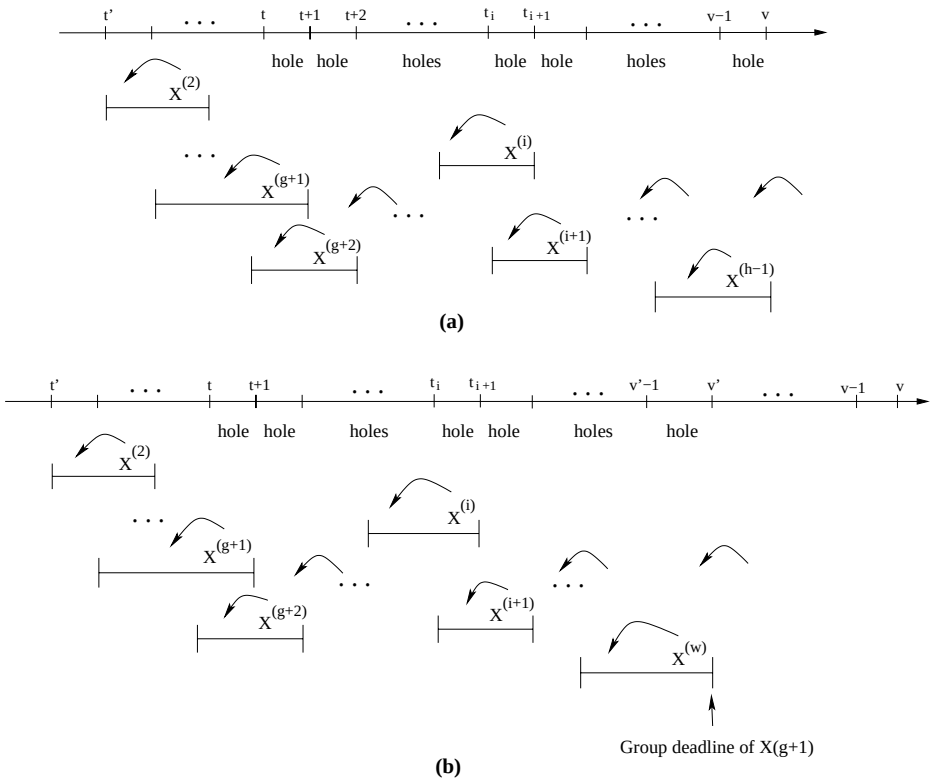

Fig. 12. Lemma 10. **(a)** There are holes in all slots in $[u, v]$. $X^{(i)}$ scheduled at t_i displaces $X^{(i-1)}$ scheduled at t_{i-1}. Also, by (28), the t_i's are consecutive and satisfy $t_i = u + i - (g + 1)$. Further, $X^{(h-1)}$ is the subtask scheduled in slot v. **(b)** Case 2. $D(X^{(g+1)}) = v'$. Hence, either $d(X^{(v'-u+g+1)}) = v' \ \wedge \ b(X^{(v'-u+g+1)}) = 0$ (as depicted) or $d(X^{(v'-u+g+1)}) > v'$.

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 (28), $t_w = v' - 1$. 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 (29)),

$$\text{for all } i \in \{g + 1, \ldots, w\}, \ d(X^{(i)}) \geq t + i - g. \tag{30}$$

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

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 (30), $d(X^{(w-1)}) \geq v' - 1$. Therefore, $d(X^{(w-1)}) = v' - 1$. Similarly, by induction, $d(X^{(i)}) = t + i - g$ for all $i \in \{g + 1, \ldots, w\}$. (Refer to Figure 12(b).) Because $D(X^{(g+1)}) = v'$ and $d(X^{(v'-u+g+1)}) = v'$, 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 5, part (b), the eligibility time of the successor of $X^{(w)}$ is at least v'. Hence, Δ_w is not valid. Thus, the displacements do not extend beyond slot $v'-1$. □

Lemma 11 below extends Lemma 9 by allowing B to consist solely of heavy tasks. The following claims are used in its proof.

Claim 2 *If U_j is scheduled in slot u, where $0 \leq u < t_d$ and $u < d(U_j)$, and if there is a hole in slot u, then $d(U_j) = u+1$.*

Proof. Because $u < t_d$, by Definition 2, no deadline is missed in $[0, u+1)$. Because U_j is scheduled in slot u, *i.e.*, $[u, u+1)$, we have $d(U_j) \geq u+1$. Suppose that $d(U_j) > u+1$. Then, by part (b) of Lemma 5, 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, $d(U_j) = u+1$. □

Claim 3 *Suppose there is a hole in slot $u \in \{0, \ldots, t_d-1\}$. Let U_j be a subtask scheduled at $t' \leq u$. If the eligibility time of the successor of U_j is at least $u+1$, then $d(U_j) \leq u+1$.*

Proof. If $t' = u$, then clearly $u < d(U_j)$ and hence by Claim 2, $d(U_j) = u+1$. On the other hand, if $t' < u$ and $d(U_j) > u$, then we have $e(U_j) \leq u < d(U_j)$. In this case, Lemma 7 implies that $d(U_j) = u+1$. Thus, $d(U_j) \leq u+1$. □

Lemma 11 *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 (16)),

$$LAG(\tau, t) < LAG(\tau, t+1). \tag{31}$$

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 9. If any task in B is light, then by Lemma 9, $LAG(\tau, t+2) \leq 0$, 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 Corollary 1,

$$\text{for all } U_j \in C,\ d(U_j) = t+1 \ \wedge\ b(U_j) = 1. \tag{32}$$

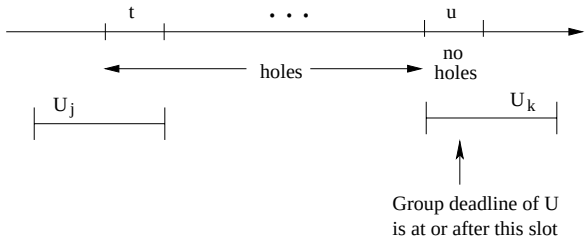

Fig. 13. Lemma 11. U_j is the critical subtask of a task in A and U_k is the successor of U_j. There is a hole in each slot in $[t, u-1]$ and there is no hole in slot u. The earliest time at which U_k's PF-window starts is u, *i.e.*, $r(U_k) \geq u$.

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

$$\text{for all } U_j \in C,\ d(U_j) = t+1 \ \wedge\ b(U_j) = 1 \ \wedge\ D(U_j) \geq D(L_i). \tag{33}$$

By Lemma 10, 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.

Figure 13 depicts this situation. By (U) and (H),

$$u \geq t+1, \tag{34}$$

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

Claim 4 *All tasks in B are inactive over the interval $[t+1, u)$.*

Proof of Claim: 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 a hole in slot v, by Claim 2, $d(V_i) = v+1$. 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 $\{t, \ldots, v-1\}$, V_i 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 (32)). Hence, no subtask of any task in B is scheduled in $[t, u)$ (see Figure 13). 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 (32), this implies that all the tasks in B are inactive in $[t+1, u-1]$. □

Let U_j be any subtask in C, and let U_k be the successor of U_j. By Claim 4, $r(U_k) \geq u$. Furthermore, by (32) – (34) 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

$$\text{for all } U \in B,\ \mathit{flow}(U,t) + \mathit{flow}(U,u) \leq wt(U). \tag{35}$$

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

Claim 5 *$LAG(\tau, v+1) \leq LAG(\tau, v)$ for all $v \in \{t+1, \ldots u-1\}$,*

Proof of Claim: 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 6, 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

$$e(V_k) \leq v < d(V_k). \tag{36}$$

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 3, we have $d(V_k) \leq v$, which contradicts (36). Therefore, $LAG(\tau, v+1) \leq LAG(\tau, v)$ for all $v \in \{t+1, \ldots, u-1\}$. □

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 (15) and Claim 5, $\sum_{T \in \tau} \mathit{flow}(T, v) \leq \sum_{T \in \tau} S(T, v)$. Therefore,

$$\text{for all } v \in \{t+1, \ldots, u-1\},\ \sum_{T \in \tau} \mathit{flow}(T, v) \leq M - H_v. \tag{37}$$

Because τ is feasible, by (8), we have $\sum_{T \in B \cup I_u \cup A_u} wt(T) \leq M$. Hence, by (35) and (F1), we get $\sum_{T \in B} (\mathit{flow}(T,t) + \mathit{flow}(T,u)) + \sum_{T \in I_u \cup A_u} \mathit{flow}(T,u) \leq M$. Thus,

$$\sum_{T \in B} \mathit{flow}(T,t) + \sum_{T \in B \cup I_u \cup A_u} \mathit{flow}(T,u) \leq M. \tag{38}$$

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,

$$\sum_{T \in A_t} \mathit{flow}(T,t) \leq \sum_{T \in A_t} wt(T) \leq M - H_t. \tag{39}$$

We are now ready to show that $LAG(\tau, u+1) \leq 0$. Because $S(T,v) = M - H_v$, by (15), $LAG(\tau, u+1) - LAG(\tau, t) = R$, where $R = \sum_{v=t}^{u} \left(\sum_{T \in \tau} \mathit{flow}(T,v)\right) - \sum_{v=t}^{u}(M - H_v)$. By (U), there are no holes in slot u, hence, $H_u = 0$. Therefore,

$$R = \sum_{v=t}^{u} \left(\sum_{T \in \tau} \mathit{flow}(T,v) \right) - \sum_{v=t}^{u-1} (M - H_v) - M. \tag{40}$$

The right-hand side of (40) can be rewritten as follows.

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

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

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

By (37) – (39), the above value is non-positive. Hence, by (40), $LAG(\tau, u+1) - LAG(\tau, t) \leq 0$. Because $LAG(\tau, t) < 1$, this implies that $LAG(\tau, u+1) < 1$.

By (U) and (34), $t + 1 \leq u \leq min(D(U_j), t_d) - 1$. Hence, $t + 2 \leq u + 1 \leq t_d$. Thus, there exists a $v \in \{t+2, \ldots, t_d\}$ such that $LAG(\tau, v) < 1$. □

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 (16)), we have $t + 2 \leq t_d$. By Lemma 11, $LAG(\tau, v) \leq 0$ for some $v \in \{t+2, \ldots, t_d\}$. By Lemma 5, parts (e) and (f), v cannot be t_d or $t_d - 1$. Thus, $v \leq t_d - 2$. Because $LAG(\tau, t_d) \geq 1$, this contradicts the maximality of t. Therefore, t_d and τ as defined cannot exist. Thus, we have the following.

Theorem 1 PD^2 *correctly schedules any feasible GIS task system.*

The following corollaries are immediate.

Corollary 2 PD^2 *is optimal for scheduling IS task systems on multiprocessors.*

Corollary 3 PD^2 *is optimal for scheduling sporadic task systems on multiprocessors.*

5 Concluding Remarks

In this paper, we have shown that PD^2, the most efficient optimal Pfair scheduling algorithm proposed to date, correctly schedules any feasible IS or GIS task system on M processors. This paper is the first to show that IS, GIS, or sporadic tasks can be optimally scheduled on systems of more than two processors.

Two key insights led to our proof: the development of a notion of lag for GIS systems that can be used to sufficiently predict where holes exist in a schedule, and the identification of certain minimality conditions (Definitions 2 and 3) that facilitate the reasoning. It is these notions that distinguish our proof from previous proofs for Pfair/ERfair scheduling algorithms. Since the presentation of this work as a conference paper [18], the proof techniques of this paper have been used in work involving various Pfair-like scheduling algorithms and task models. In particular, these techniques have been used to establish the correctness of algorithms for systems with soft real-time tasks that require only bounded deadline tardiness [11,13,12,19], for dynamic systems that permit tasks to leave and join at runtime [20], and for systems where task weights can be changed dynamically [2,8].

The IS task model incorporates a very flexible notion of a rate. Indeed, as shown herein, the resulting multiprocessor execution model has many characteristics in common with the uniprocessor rate-based execution model proposed by Jeffay and Goddard [14]. The IS task model also generalizes the model considered in our prior work on scheduling mixed early-release/non-early-release periodic task systems [5].

The rate-based properties of PD^2 make it potentially useful in several application domains. One such application (potentially) is the scheduling of rate-based packet flows in wave-division-multiplexing (WDM) networks. In WDM networks, optical multiplexing techniques are used to send multiple packets over the same link in parallel. In a similar vein, PD^2 can be used to solve the parallel switching problem in ATM networks mentioned in [1]. Also, as noted earlier, PD^2 might be useful in multiprocessor real-time applications that have processing steps that are triggered by messages sent over a network.

The fairness properties of PD^2 also make it useful for multiplexing independently-authored applications on the same system. This is because such algorithms ensure *temporal isolation* among applications (no "misbehaving" application can execute faster than its proscribed rate, unless there is spare processing capacity). This observation led researchers at the University of Massachusetts and Ensim Corp. to investigate the use of fair scheduling algorithms on multiprocessor servers for commercial web-hosting sites [9,10].

However, this prior investigation was entirely empirical in nature. In this paper, we have given the first ever general optimality proof for a multiprocessor rate-based scheduling algorithm that provides fairness guarantees. As noted above, the techniques used in our proof are not unique to PD^2 and have been applied to other rate-based and fair scheduling algorithms as well.

Acknowledgement: We are grateful to Uma Devi and Phil Holman for their comments on earlier drafts of this paper.

A Properties about Flows for IS and GIS Tasks

We now prove properties (F2) and (F3) of GIS tasks that are used in the proofs in Section 4. We establish (F2) and (F3) by deriving several other useful properties about GIS and IS tasks. With the exception of (B3) (which applies only to IS tasks), all the properties below hold for GIS tasks.

(L) For a light task T, if T_k is the successor of T_i, then $d(T_k) \geq d(T_i) + 2$.

Proof. Because T_k is T_i's successor, $k \geq i+1$. Hence, $d(T_k) \geq \theta(T_k) + \left\lceil \frac{i+1}{wt(T)} \right\rceil$. By (6), $d(T_k) \geq \theta(T_i) + \left\lceil \frac{i+1}{wt(T)} \right\rceil$. Therefore, by (5), $d(T_k) - d(T_i) \geq \left\lceil \frac{i+1}{wt(T)} \right\rceil - \left\lceil \frac{i}{wt(T)} \right\rceil$. Thus,

$$\begin{aligned}
d(T_k) - d(T_i) &\geq \frac{i+1}{wt(T)} - \left\lceil \frac{i}{wt(T)} \right\rceil && , \ \lceil x \rceil \geq x \\
&> \frac{i+1}{wt(T)} - \frac{i}{wt(T)} - 1 && , \ \lceil x \rceil < x + 1 \\
&= \frac{1}{wt(T)} - 1 && , \ \text{by simplification} \\
&> 2 - 1 && , \ wt(T) < 1/2 \Rightarrow \frac{1}{wt(T)} > 2
\end{aligned}$$

Therefore, $d(T_k) > d(T_i) + 1$, *i.e.*, $d(T_k) \geq d(T_i) + 2$. □

(B1) Let T_i be a subtask with $b(T_i) = 1$. If T_{i+1} exists, then $f(T_i, d(T_i) - 1) + f(T_{i+1}, r(T_{i+1})) = wt(T)$.

Proof. By (13), $f(T_i, d(T_i) - 1) = i - \left(\left\lceil \frac{i}{wt(T)} \right\rceil - 1 \right) \times wt(T)$, and $f(T_{i+1}, r(T_{i+1})) = \left(\left\lfloor \frac{i}{wt(T)} \right\rfloor + 1 \right) \times wt(T) - i$. Since $b(T_i) = 1$, by (9), $\left\lceil \frac{i}{wt(T)} \right\rceil = \left\lfloor \frac{i}{wt(T)} \right\rfloor + 1$. Hence, $f(T_{i+1}, r(T_{i+1})) = \left\lceil \frac{i}{wt(T)} \right\rceil \times wt(T) - i$.

Therefore, $f(T_i, d(T_i)) + f(T_{i+1}, r(T_{i+1})) = wt(T)$. (See Figure 8.) $\Box$

(B2) Let T_i be a subtask such that $b(T_i) = 1$. If T_{i+1} exists and is released late, *i.e.*, $r(T_{i+1}) \geq d(T_i)$, then $\mathit{flow}(T, d(T_i) - 1) + \mathit{flow}(T, d(T_i)) \leq wt(T)$.

Proof. Because T_{i+1} is released late, by (7), we have $r(T_{i+1}) \geq d(T_i)$. By (4) and (5), it follows that $r(T_k) > d(T_i)$ for all $k > i+1$. Similarly, $d(T_j) < d(T_i)$ for all $j < i$. This implies that the slot $d(T_i) - 1$ lies within the PF-window of only one subtask, namely, T_i, and the slot $d(T_i)$ can lie within the PF-window of only one subtask, namely, T_{i+1}. Thus, the contribution to the flow in slot $d(T_i) - 1$ is $f(T_i, d(T_i) - 1)$ and the contribution to slot $d(T_i)$ is at most $f(T_{i+1}, r(T_{i+1}))$. Hence, by (B1), $\mathit{flow}(T, d(T_i) - 1) + \mathit{flow}(T, d(T_i)) \leq wt(T)$. $\Box$

(B3) If T_i and T_k are subtasks of an *IS* heavy task T such that $k > i$ and $r(T_k) < D(T_i)$, then $f(T_i, d(T_i) - 1) + f(T_k, r(T_k)) \leq wt(T)$.

Proof. If $b(T_i) = 0$, then $D(T_i) = d(T_i)$. In this case, $r(T_k) \geq D(T_i)$ holds, since (4) implies $r(T_k) \geq d(T_i)$ (thus, no task T_k exists such that $k > i$ and $r(T_k) < D(T_i)$). In the rest of the proof, we assume that $b(T_i) = 1$.

Since $D(T_i)$ denotes the group deadline of T_i and $r(T_k) < D(T_i)$, by definition of a group deadline, we have $|w(T_j)| = 2$ and $b(T_j) = 1$ for all $j \in \{i + 1, \ldots, k-1\}$. Note that $|w(T_j)| = 2$ implies that $d(T_j) = r(T_j) + 2$. Because the total flow for a subtask is one, this implies the following.

$$\text{for all } j \in \{i+1, \ldots, k-1\}, f(T_j, r(T_j)) + f(T_j, d(T_j) - 1) = 1 \qquad \text{(A.1)}$$

Because $b(T_i) = 1$, we have $b(T_j) = 1$ for all $j \in \{i, \ldots, k-1\}$. Therefore, by (B1), $f(T_j, d(T_j)-1)+f(T_{j+1}, r(T_{j+1})) = wt(T)$. Therefore, $\sum_{j=i}^{k-1} f(T_j, d(T_j)-1) + f(T_{j+1}, r(T_{j+1})) = (k-i) \times wt(T)$. Rewriting, we get $f(T_i, d(T_i) - 1) + f(T_k, r(T_k)) + \sum_{j=i+1}^{k-1}(f(T_j, r(T_j)) + f(T_j, d(T_j) - 1)) = (k-i) \times wt(T)$. By (A.1), this implies that

$$f(T_i, d(T_i) - 1) + f(T_k, r(T_k)) + k - i - 1 = (k-i) \times wt(T).$$

Therefore, $f(T_i, d(T_i) - 1) + f(T_k, r(T_k)) = wt(T) + (k - i - 1)(wt(T) - 1)$. Because $k \geq i + 1$ and $wt(T) \leq 1$ for all T, we have $f(T_i, d(T_i) - 1) + f(T_k, r(T_k)) \leq wt(T)$. $\Box$

(B4) Let T_i be a subtask of a heavy GIS task T and let T_k $(k > i)$ be a subtask such that $r(T_k) < D(T_i)$. Then, $f(T_i, d(T_i) - 1) + f(T_k, r(T_k)) \leq wt(T)$.

Proof. Because T is a GIS task, there is an IS task U such that $wt(U) = wt(T)$, all subtasks between U_i and U_k are present, and $r(U_k) = r(T_k)$. Hence,

$r(U_k) < D(U_i)$. By (B3), $f(U_i, d(U_i) - 1) + f(U_k, r(U_k)) \leq wt(U)$. Corresponding subtasks in T and U have identical flows. Thus, $f(T_i, d(T_i) - 1) + f(T_k, r(T_k)) \leq wt(T)$. □

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

Proof. If $k = i+1$, then by (B2), $\mathit{flow}(T, d(T_i)-1)+\mathit{flow}(T, d(T_i)) \leq wt(T)$. Also, if $r(T_k) > d(T_i)$, then $\mathit{flow}(T, d(T_i)) = 0$. Hence, by (F1), $\mathit{flow}(T, d(T_i)-1) + \mathit{flow}(T, d(T_i)) \leq wt(T)$.

In the rest of the proof, we assume that $k > i + 1$ and $r(T_k) = d(T_i)$. We first show that T must be heavy. If T is light, then by (L), we have $d(T_{i+1}) > d(T_i)+1$. By (7), we also have $r(T_k) \geq d(T_{i+1})-1$ and therefore, $r(T_k) > d(T_i)$, which contradicts $r(T_k) = d(T_i)$.

Thus, T is heavy. Because $b(T_i) = 1$, by the definition of D, $D(T_i) > d(T_i)$. Hence, because $r(T_k) = d(T_i)$, we have $r(T_k) < D(T_i)$. Thus, by (B4), $\mathit{flow}(T, d(T_i) - 1) + \mathit{flow}(T, r(T_k)) \leq wt(T)$. Because $r(T_k) = d(T_i)$, we have $\mathit{flow}(T, d(T_i) - 1) + \mathit{flow}(T, d(T_i)) \leq wt(T)$. □

(F3) Let T_i be a subtask of a heavy GIS 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 $\mathit{flow}(T, d(T_i)) + \mathit{flow}(T, u) \leq wt(T)$.

Proof. Since $b(T_i) = 1$, by the definition of D, $D(T_i) > d(T_i)$. Since $u \geq d(T_i)$ and T_k is T_i's successor, if $r(T_k) > u$, then $\mathit{flow}(T, u) = 0$. Thus, by (F1), $\mathit{flow}(T, d(T_i) - 1) + \mathit{flow}(T, u) \leq wt(T)$. The other possibility is $r(T_k) = u$, which implies $r(T_k) < D(T_i)$. In this case, by (B4), $f(T_i, d(T_i) - 1) + f(T_k, r(T_k)) \leq wt(T)$. Thus, $\mathit{flow}(T, d(T_i) - 1) + \mathit{flow}(T, u) \leq wt(T)$. □

References

[1] J. Anderson, S. Baruah, and K. Jeffay. Parallel switching in connection-oriented networks. In *Proceedings of the 20th IEEE Real-time Systems Symposium*, pages 200–209. IEEE, December 1999.

[2] J. Anderson, A. Block, and A. Srinivasan. Quick-release fair scheduling. In *Proceedings of the 24th IEEE Real-time Systems Symposium*, pages 130–141. IEEE, December 2003.

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

[4] J. Anderson and A. Srinivasan. Pfair scheduling: Beyond periodic task systems. In *Proceedings of the Seventh International Conference on Real-time Computing*

Systems and Applications, pages 297–306, December 2000.

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

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

[7] 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.

[8] A. Block, J. Anderson, and G. Bishop. Fine-grained task reweighting on multiprocessors. In *Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 429–435, August 2005.

[9] 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 Fourth Symposium on Operating System Design and Implementation (OSDI 2000)*, pages 45–58. ACM, October 2000.

[10] 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 IEEE Real-time Technology and Applications Symposium*, pages 3–14. IEEE, June 2001.

[11] U. Devi and J. Anderson. Fair integrated scheduling of soft real-time tardiness class on multiprocessor platforms. In *Proceedings of the 10th IEEE Real-time and Embedded Technology and Applications Symposium*, pages 554–561. IEEE, May 2004.

[12] U. Devi and J. Anderson. Schedulable utilization bounds for EPDF fair multiprocessor scheduling multiprocessor scheduling. In *Proceedings of the 10th International Conference on Real-time and Embedded Computing Systems and Applications*, pages 261–280, August 2004.

[13] U. Devi and J. Anderson. Improved conditions for bounded tardiness under EPDF fair multiprocessor scheduling. In *Proceedings of 12th International Workshop on Parallel and Distributed Real-Time Systems*. IEEE, April 2004 (on CD ROM).

[14] K. Jeffay and S. Goddard. A theory of rate-based execution. In *Proceedings of the* 20^{th} *IEEE Symposium on Real-time Systems*, pages 304–314. IEEE, December 1999.

[15] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 30:46–61, January 1973.

[16] A. K. Mok. Fundamental design problems of distributed systems for the hard-real-time environment. Technical Report MIT/LCS/TR-297, Massachusetts Institute of Technology, 1983.

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

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

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

[20] A. Srinivasan and J. Anderson. Fair scheduling of dynamic task systems on multiprocessors. In *Proceedings of the 11th International Workshop on Parallel and Distributed Real-time Systems*. IEEE, April 2003, on CD ROM.

[21] A. Srinivasan, P. Holman, J. Anderson, S. Baruah, and J. Kaur. Multiprocessor scheduling in processor-based router platforms: Issues and ideas. In *Proceedings of the Second Workshop on Network Processors*, pages 48–62, February 2003.