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

E-mail: {anands,anderson}@cs.unc.edu

Phone: (919) 962-1757

May 2001

Abstract

We consider the intra-sporadic task model, which is a generalization of the sporadic task model motivated by recent work on Pfair scheduling. The intra-sporadic model is essentially a quantum-based, multiprocessor variant of the uniprocessor rate-based execution model of Jeffay and Goddard. In the intra-sporadic model, a task is specified by an average rate of execution, and there is no restriction on instantaneous execution rates. Such flexibility is useful in applications in which some processing steps may be highly jittered.

In previous work, we showed that an intra-sporadic task system is feasible on M processors iff its total utilization is at most M. We also gave an optimal algorithm for scheduling intra-sporadic tasks on two processors. In this paper, we show that the PD^2 Pfair algorithm can be used to schedule any intra-sporadic task system that is feasible on M processors. Because the sporadic model is a special case of the intra-sporadic model, our work shows that PD^2 is also optimal for scheduling sporadic tasks on a multiprocessor. This paper is the first to show that sporadic or intra-sporadic tasks can be optimally scheduled on systems of more than two processors.

*Work supported by NSF grants CCR 9732916, CCR 9972211, CCR 9988327, and ITR 0082866.

1 Introduction

In the real-time scheduling literature, the periodic [11] and sporadic [12] task models have received the most attention. In the periodic model, each task is invoked repeatedly, with consecutive invocations 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. A scheduling scheme that assumes periodic or sporadic arrivals complicates the design of such an application. 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 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, 4, 7, 8].

Under *Pfair scheduling*, each task is required to execute at a uniform rate, while respecting a fixed allocation quantum. Specifically, Pfairness requires 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 allocation error becomes too large or too small. "*Early-release*" *fair* (*ERfair*) differs from Pfair scheduling in a rather simple way: under ERfair scheduling, a subtask may become eligible for execution *early*, *i.e.*, before its Pfair window. In periodic task systems, eligibility times are ordinarily defined so that, within a single job (*i.e.*, task invocation), each successive subtask becomes eligible as soon as its predecessor completes. By doing this, job response times can be reduced.

In [5], we proposed the *intra-sporadic task model*, which generalizes both the sporadic and early-release task models. In the intra-sporadic model, subtasks are allowed to arrive late, *i.e.*, there may be separation between consecutive windows of the same task. In addition, subtasks are also allowed to execute before their Pfair windows, as in ERfair scheduling. As explained later, the intra-sporadic notion of a rate is quite similar to that found in the *uniprocessor* rate-based execution model of Jeffay and Goddard [10]. In [5], we gave a polynomial-time feasibility test for scheduling intra-sporadic tasks on M processors and also presented a simple algorithm to optimally schedule intra-sporadic tasks on two processors. However, we left open the problem of optimally scheduling intra-sporadic 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 [4] correctly schedules any feasible intra-sporadic task system on M processors. Because the intra-sporadic model is a generalization of the sporadic model, our work also shows that PD^2 is optimal for scheduling sporadic tasks in a Pfair or ERfair manner on multiprocessors. Since periodic task systems represent a "worst-case" scenario in the spectrum of intra-sporadic (or sporadic) task systems, one may think that the optimality of PD^2 would

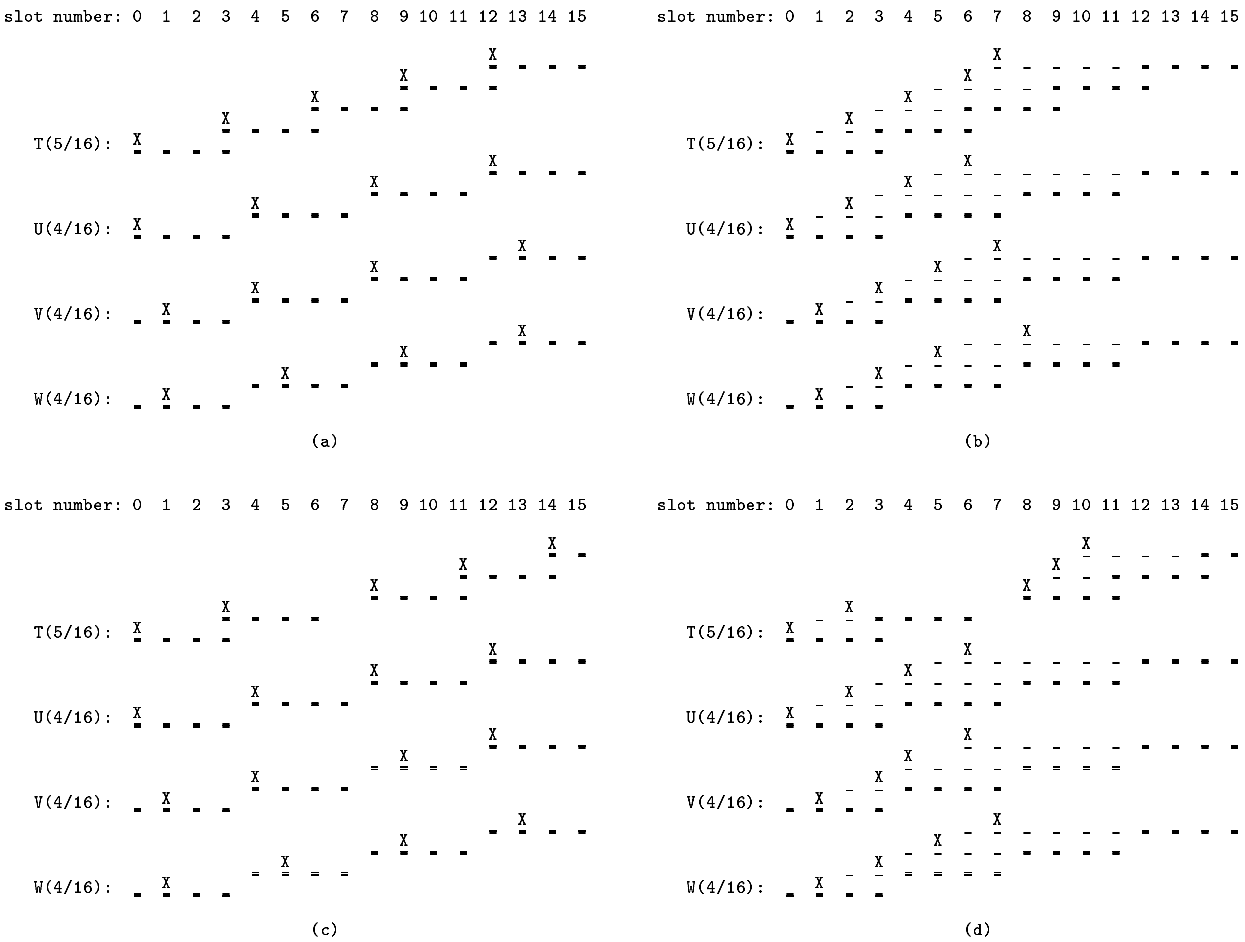

Figure 1: A partial schedule for an intra-sporadic task set under various conditions. In each of these schedules, subtasks of the same task are shown on different lines. Each subtask has an *eligibility interval* corresponding to the sequence of time slots in which it can be scheduled; a subtask's eligibility interval must include its Pfair window. In each schedule, each subtask's eligibility interval is denoted by a sequence of dashes, with its Pfair window denoted in bold; the time slot in which the subtask is scheduled is denoted by an 'X'. **(a)** All tasks are periodic and are Pfair-scheduled. **(b)** All tasks are periodic and are ERfair-scheduled. **(c)** Task T is intra-sporadic and no subtask is eligible before its Pfair window. **(d)** Task T is intra-sporadic and the eligibility intervals are like those in an ERfair-scheduled system.

follow as a simple corollary from previous work. However, as explained in Section 4, this is not the case. Indeed, all correctness proofs presented previously for Pfair and ERfair scheduling algorithms hinge critically on the assumption that all tasks are periodic. In this paper, we provide a new approach for dealing with Pfair- or ERfair-scheduled systems and use it to show that intra-sporadic tasks can be optimally scheduled on multiprocessors. This paper breaks new ground by being the first to show that sporadic or intra-sporadic tasks can be optimally scheduled on systems of more than two processors.

Some example schedules. Before continuing, we consider some example schedules that highlight the differences among the task models considered in this paper. In the Pfair scheduling literature, the ratio of a task's execution cost to its period is referred to as its *weight*. A task's weight determines the length and alignment of its Pfair windows. Figure 1 shows some schedules involving four tasks: a task T with weight 5/16, and three tasks U, V, and W with weight $4/16$. Inset (a) shows the schedule for these tasks up to time slot 15 in a Pfair-scheduled system. Inset (b) shows the same task set under ERfair scheduling. Insets (c) and (d) show schedules similar to those in insets (a) and (b), respectively, except that the third subtask of task T is released two time units late (perhaps this late release represents a packet that arrived late). In the intra-sporadic model, each subtask has an *eligibility interval* corresponding to the sequence of time slots in which it can be scheduled; a subtask's eligibility interval must include its Pfair window. For example, the eligibility interval of the second subtask of T in inset (b) is $[1, 6]$, whereas its Pfair window is $[3, 6]$. As we shall see later, the scheduling decisions in Figure 1 are consistent with PD^2.

The rest of this paper is organized as follows. In Section 2, we describe Pfair and ERfair scheduling in greater detail and also introduce the concept of an intra-sporadic task. Then, in Section 3, we describe the PD^2 algorithm. In Section 4, we prove that PD^2 is optimal for scheduling intra-sporadic tasks. We end the paper with concluding remarks in Section 5.

2 Definitions and Related Work

In the scheduling disciplines we consider, processor time is allocated in discrete time units, or quanta; the time interval $[t, t+1)$, where t is a nonnegative integer, is called *slot* t. A task T may be allocated time on different processors, as long as it is not allocated time on different processors in the same slot. The sequence of allocation decisions over time defines a "schedule." Formally, a *schedule* S is a mapping $S\colon \tau \times \mathcal{N} \mapsto \{0, 1\}$, where τ is a set of tasks and $\mathcal{N}$ is the set of nonnegative integers. $S(T, t) = 1$ means that task T is scheduled in slot t. In any schedule for M processors, at most M tasks can be scheduled in one slot. Therefore, $\sum_{T \in \tau} S(T, t) \leq M$ for all t. Note that the above definition of a schedule says nothing about any particular scheduling discipline and thus permits "invalid" scheduling decisions (*e.g.*, deadlines might be missed). We will state the conditions for the validity of a schedule later in Section 2.2.

Associated with each task T is a *weight* $wt(T) = T.e/T.p$, where $T.e$ and $T.p$ are positive integers satisfying $T.e \leq T.p$ (*i.e.*, the weight of any task is at most 1). A task's weight defines the rate at which it is to be scheduled. A task with weight less than $1/2$ is called a *light* task, while a task with weight at least $1/2$ is called a *heavy* task.

We now briefly describe Pfair and ERfair scheduling, after which we introduce the intra-sporadic task model.

2.1 Pfair and ERfair Scheduling.

Pfair scheduling was introduced by Baruah *et al.* [7] as a way of optimally scheduling synchronous, periodic task systems on multiprocessors. In this section, we only consider such systems. A periodic task T with a *period* $T.p$ and an *execution cost* $T.e$ has a weight of $T.e/T.p$. Every $T.p$ time units, T releases a new *job* with a cost of $T.e$ time units. Each job of a task must complete execution before the next job of that task is released.

A task T executing at the rate defined by its weight should ideally receive $(T.e/T.p)t$ processor time during each interval of length t. However, if a scheduling quantum is used, this is not possible. In a Pfair-scheduled system, this requirement is weakened to allow execution rates to deviate by a limited amount from those in an ideal system. Deviance from the ideal system is formally captured by the concept of *lag*. Formally, the *lag of task T at time t in a schedule S*, denoted $lag(T, t)$, is defined as follows:

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

(In this paper, we consider only *lag* values in a specific schedule S obtained by the PD^2 Pfair algorithm. Hence, we leave the schedule implicit and use $lag(T, t)$ instead of $lag(T, t, S)$.) A schedule is *Pfair* iff

$$(\forall T, t :: -1 < lag(T, t) < 1). \tag{2}$$

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

The Pfair lag bounds given above have the effect of breaking each task T into an infinite sequence of unit-time *subtasks*.[1] We denote the i^{th} subtask of task T as T_i, where $i \geq 1$. As in [7], we associate with each subtask T_i a *pseudo-release*

$$r(T_i) = \left\lfloor \frac{i-1}{wt(T)} \right\rfloor \tag{3}$$

and a *pseudo-deadline*

$$d(T_i) = \left\lceil \frac{i}{wt(T)} \right\rceil - 1. \tag{4}$$

Derivations of these expressions can be found in [4]. In a Pfair-scheduled system, $r(T_i)$ is the first slot into which T_i potentially could be scheduled, and $d(T_i)$ is the last such slot. (Note that the "-1" in (4) arises because we are referring to slots and not time.) For brevity, we often refer to pseudo-deadlines and pseudo-releases as simply deadlines and releases, respectively. The interval $[r(T_i), d(T_i)]$ is called the *window* of subtask T_i and is denoted by $w(T_i)$. The *length* of window $w(T_i)$, denoted $|w(T_i)|$, is defined as $d(T_i) - r(T_i) + 1$. As an example, consider a task T with weight $T.e/T.p = 8/11$. Each job of this task consists of eight windows, one for each of its unit-length subtasks. Using Equations (3) and (4), it is possible to show that the windows within each job of T are as depicted in Figure 2. Note that successive windows overlap by one slot. In general, consecutive windows of a task are either disjoint or overlap by one slot.

The notion of early-release fair (ERfair) scheduling [3] is obtained by simply dropping the -1 lag constraint in (2). Formally, a schedule is *ERfair* iff $(\forall T, t :: lag(T, t) < 1)$. This means that a subtask is now allowed to execute before its window. In particular, if two subtasks are part of the same job, then the second subtask is eligible to execute as soon as the first completes. Note that any Pfair schedule is ERfair, but not necessarily vice versa. It is easy to show that any ERfair schedule (and hence, any Pfair schedule) is periodic [3, 7].

[1] We have refrained from introducing the term "subjob" to refer to an invocation of a subtask because each subtask is invoked only once (each task consists of an *infinite* sequence of subtasks).

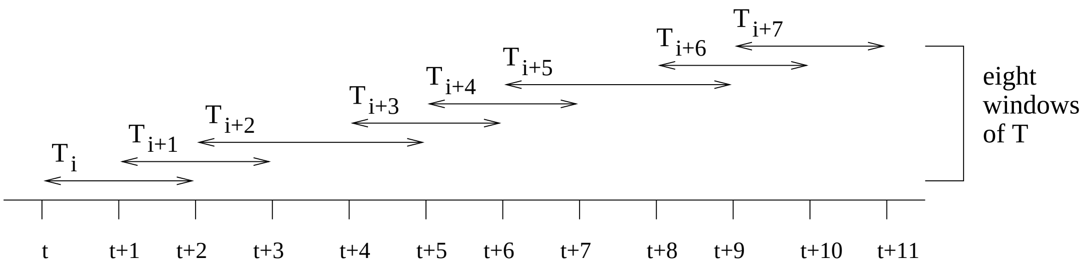

Figure 2: Pfair scheduling: the eight "windows" of a periodic task T with weight $T.e/T.p = 8/11$. Each of T's eight units of computation must be allocated processor time during its window, or else a lag-bound violation will result.

2.2 Intra-sporadic Tasks

Intra-sporadic tasks are a generalization of sporadic tasks. In a sporadic task system, there can be separation between consecutive jobs of a task. In an intra-sporadic task system, separation between consecutive subtasks of the same task is allowed. Furthermore, early subtask releases are allowed.

Formal definition. An *intra-sporadic task system* is defined by the tuple (τ, e), where τ represents a set of tasks, and e is a function that indicates the slot at which each subtask first becomes eligible for execution. Each task may release either a finite or infinite number of subtasks. We assume that $e(T_i) \geq e(T_{i-1})$ for all $i \geq 2$, *i.e.*, subtask T_i cannot become eligible before its predecessor T_{i-1}.

In the intra-sporadic model, each subtask has two windows of interest, namely its *Pfair window* (PF-window) and its *intra-sporadic window* (IS-window), respectively. A subtask's IS-window defines its eligibility interval, *i.e.*, the sequence of time slots in which it may be potentially scheduled. Its IS-window must include its PF-window. Formally, T_i's IS-window is defined to be $[e(T_i), d(T_i)]$ and its PF-window is defined to be $[r(T_i), d(T_i)]$. As we shall see, the terms $r(T_i)$ and $d(T_i)$ have a similar interpretation to that given previously for synchronous, periodic task systems. As before, we will use the term $w(T_i)$ to refer to the PF-window of subtask T_i.

As an example, consider the third subtask of the task T in Figure 1(d). The IS- and PF-windows of this subtask are both equal to $[8, 11]$. For the fourth subtask, the IS-window is $[9, 14]$ and the PF-window is $[11, 14]$.

The terms $r(T_i)$ and $d(T_i)$ are defined inductively by examining the alignment of the PF-windows of T in a synchronous, periodic task system. As mentioned earlier, in such a system, consecutive PF-windows of T are either disjoint or overlap by one slot. The bit $b(T_i)$, defined below, distinguishes between these two possibilities.

$$b(T_i) = \begin{cases} 1, & \text{if } \left\lceil \frac{i}{wt(T)} \right\rceil = \left\lfloor \frac{i}{wt(T)} \right\rfloor + 1 \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

By examining Equations (3) and (4) it should be clear that in a *synchronous*, *periodic* task system, $r(T_{i+1}) = d(T_i)$ (*i.e.*, $w(T_i)$ and $w(T_{i+1})$ overlap by one slot) if $b(T_i) = 1$, and $r(T_{i+1}) = d(T_i) + 1$ (*i.e.*, $w(T_i)$ and $w(T_{i+1})$ do not overlap) if $b(T_i) = 0$. For intra-sporadic tasks, we define $b(T_i)$ exactly as above. Given this definition, we can define $r(T_i)$, which defines the beginning of the PF-window of T_i.

$$r(T_i) = \begin{cases} e(T_i), & \text{if } i = 1 \\ max(e(T_i),\ d(T_{i-1}) + 1 - b(T_{i-1})) & \text{if } i \geq 2 \end{cases} \tag{6}$$

According to this definition, the PF-window of T_1 begins when it first becomes eligible. For $i \geq 2$, $r(T_i)$ is defined to be the later of $e(T_i)$ and $d(T_{i-1}) + 1 - b(T_{i-1})$. Thus, if T_i becomes eligible *during* T_{i-1}'s PF-window, then $r(T_i) = d(T_{i-1}) + 1 - b(T_{i-1})$, and hence, the spacing between $r(T_{i-1})$ and $r(T_i)$ is exactly as in a synchronous, periodic task system. (Note that the notion of a job is not mentioned here. For systems in which subtasks are grouped into jobs that are released in sequence, the definition of e would preclude T_i from becoming eligible during T_{i-1}'s PF-window if T_i is the first subtask of its job.) On the other hand, if T_i becomes eligible *after* T_{i-1}'s PF-window, then T_i's PF-window begins when T_i becomes eligible. Note that (6) implies that consecutive PF-windows of the same task are either disjoint or overlap by one slot, as in a synchronous, periodic task system. Also, if T_i is scheduled within its PF-window, then it is also scheduled within its IS-window.

T_i's deadline $d(T_i)$ is defined to be $r(T_i) + |w(T_i)| - 1$, where $|w(T_i)|$ is the length of T_i's PF-window. PF-window lengths are the same in intra-sporadic and synchronous, periodic systems. Thus, by (3) and (4), we have the following.

$$|w(T_i)| = \left\lceil \frac{i}{wt(T)} \right\rceil - \left\lfloor \frac{i-1}{wt(T)} \right\rfloor \tag{7}$$

$$d(T_i) = r(T_i) + \left\lceil \frac{i}{wt(T)} \right\rceil - \left\lfloor \frac{i-1}{wt(T)} \right\rfloor - 1 \tag{8}$$

Thus, if T_i becomes eligible early, *i.e.*, $e(T_i) < r(T_i)$, then its deadline is postponed to where it would have been if $e(T_i) = r(T_i)$.

It should be clear that our notion of an intra-sporadic task is obtained by allowing a task's PF-windows to be right-shifted from where they would appear if the task were synchronous and periodic. In addition, we allow a subtask to become eligible before the start of its PF-window, as in ERfair scheduling. A schedule for a system of intra-sporadic tasks is said to be *valid* iff each subtask is scheduled within its IS-window.

Some special cases. Using the definitions above, we can show that sporadic and asynchronous tasks are special cases of intra-sporadic tasks. In particular, a *sporadic task* T is an intra-sporadic task in which only the first subtask of each job may be released late, *i.e.*, separated from the previous subtask of T. In other words, if T_i and T_{i+1} are part of the same job, then $r(T_{i+1}) = d(T_i) + 1 - b(T_i)$. An *asynchronous task* T is an intra-sporadic task in which only the very first subtask may be released late, *i.e.*, $r(T_{i+1}) = d(T_i) + 1 - b(T_i)$ for all $i \geq 1$. Note that, by defining e appropriately, we can obtain eligibility intervals (*i.e.*, IS-windows) like those in either a Pfair- or ERfair-scheduled system.

Relation to the RBE task model. In the *rate-based execution* (RBE) model [10], 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 intra-sporadic task 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. In both models, the release rate expressed is an average rate and there is no restriction on the instantaneous rate of releases. In particular, 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 that the system is still feasible. Similarly, in the intra-sporadic model, deadlines of early subtasks are postponed using (6) and (8).

Feasibility. In [5], we proved that an intra-sporadic task system τ has a valid schedule on M processors iff

$$\sum_{T \in \tau} \frac{T.e}{T.p} \leq M. \tag{9}$$

Our proof is based on a network flow construction that is similar to that given by Baruah *et al.* [8] to prove that (9) is a feasibility condition for synchronous, periodic tasks. In fact, our proof shows that there exists a valid schedule in which each subtask is scheduled in its PF-window.

Lag values in an intra-sporadic task system. The lag of an intra-sporadic task at time t can be defined in the same way as it is defined for synchronous, periodic tasks:

$$lag(T, t) = ideal(T, t) - \sum_{u=0}^{t-1} S(T, u), \tag{10}$$

where $ideal(T, t)$ is the amount of processor time task T receives in an ideal system[2] in $[0, t)$. For a synchronous, periodic task, $ideal(T, t)$ is equal to $(T.e/T.p)t$, which is not generally true for an intra-sporadic task. To define the value of $ideal(T, t)$ for an intra-sporadic task, we need to consider the flow network used in the feasibility proof given in [5]. A valid schedule is shown to exist in that proof by constructing a flow graph with integral capacities, and then showing that a certain real-valued flow exists. The value of $ideal(T, t)$ is based on this real-valued flow and is defined as follows:

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

where $flow(T, u)$ is the flow (or share) assigned to task T in slot u.

We formally define $flow(T, u)$ below. First, we show how flows are calculated for an example task. Consider a task of weight $5/16$. In any valid schedule, each subtask of this task must receive 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 3. 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 intra-sporadic task of weight $5/16$ in which some subtasks are released late. Note that the shares for

[2]We assume the ideal system gives each task T a share of exactly $T.e/T.p$, and does not distribute the excess processor capacity that may be present. There may be other ways to define an "ideal" system, but the definition given here is sufficient for our proof.

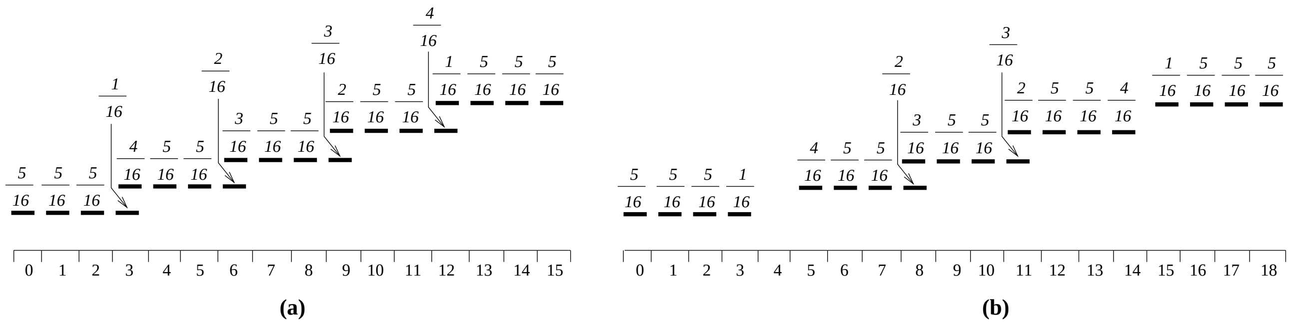

Figure 3: Windows of a task of weight $5/16$ and the share of each subtask in the slots of its window. **(a)** No subtask is released late. **(b)** The second and fifth subtasks are released late.

each subtask sum to one. For example, the total share for the first subtask is $5/16 + 5/16 + 5/16 + 1/16 = 1$. Also, note that the share in each slot is at most $5/16$, the weight of the task. For a periodic task (see inset (a)), the share in each slot *exactly* equals the weight of the task, whereas for an intra-sporadic task, the share may be less (see slot 3 in inset (b)). In the flow network, each subtask has flows corresponding to these shares, and therefore, the flow values give us the total share that a task receives in an ideal system.

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.

$$f(T_i, u) = \begin{cases} wt(T) & \text{if } u \in [r(T_i) + 1, d(T_i) - 1] \\ (\left\lfloor \frac{i-1}{wt(T)} \right\rfloor + 1) \times wt(T) - (i-1) & \text{if } u = r(T_i) \\ i - (\left\lceil \frac{i}{wt(T)} \right\rceil - 1) \times wt(T) & \text{if } u = d(T_i) \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

For example, consider the last slot of the second subtask in Figure 3(b) and also the first slot of the third subtask. $f(T_2, d(T_2)) = f(T_2, 8)$, which by (12) equals $2 - (\left\lceil \frac{2}{5/16} \right\rceil - 1) \times (5/16) = 2/16$, as shown in the figure. Similarly, $f(T_3, r(T_3)) = f(T_3, 8)$, which by (12) equals $(\left\lfloor \frac{2}{5/16} \right\rfloor + 1) \times (5/16) - (3-1) = 3/16$. Note that these two flows sum to $5/16$, the weight of the task.

The function $flow(T, u)$ can now be defined as $flow(T, u) = \sum_i f(T_i, u)$. It is easy to show the following.

(P1) For all time slots t, $flow(T, t) \leq wt(T)$.

From (10) and (11), we get the following relation between $lag(T, t)$ and $lag(T, t+1)$ for a schedule S.

$$lag(T, t+1) = \sum_{u=0}^{t} (flow(T, u) - S(T, u)) = lag(T, t) + flow(T, t) - S(T, t) \tag{13}$$

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.

$$LAG(\tau, t+1) = LAG(\tau, t) + \sum_{T \in \tau} flow(T, t) - \sum_{T \in \tau} S(T, t) \tag{14}$$

(As in the definition of lag, S is omitted.) $LAG(\tau, 0)$ is defined to be 0 for any task system τ. Note that the definitions of lag and LAG do not make any assumptions about the validity of the corresponding schedule, and in particular, the schedule under consideration may be invalid.

3 Algorithm PD^2

In this section, we describe the PD^2 scheduling algorithm, which is the most efficient Pfair algorithm proposed to date for optimally scheduling periodic tasks. In PD^2, eligible subtasks are prioritized by their deadlines. If two subtasks have the same deadline, then two tie-break parameters are inspected to break the tie. The first tie-break parameter is the successor bit $b(T_i)$ defined in Section 2.2 and the second is the "group deadline" defined below. The rationale behind each tie-break is explained below when considering the PD^2 priority definition.

The group deadline. We first define the group deadline for a periodic task and then explain how to calculate it for an intra-sporadic task. In [4], we showed that all windows of a heavy task are of length two or three. Consider a sequence $T_i, \ldots, T_j$ of subtasks of a heavy task T such $|w(T_k)| = 2$ 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_i, T_{i+1} or T_{i+2}, T_{i+3}, T_{i+4} or T_{i+5}, T_{i+6} in Figure 2). 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 group of 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 slots that remain empty exactly correspond to the group deadlines of T. For example, in Figure 2, T has group deadlines at slots $t + 3$, $t + 7$, and $t + 10$.

We let $D(T_i)$ denote the group deadline of subtask T_i. Formally, if T is heavy, then $D(T_i) = (\mathbf{min}\ u :: u \geq d(T_i)$ and u is a group deadline of $T)$. For example, in Figure 2, $D(T_i) = t + 3$ and $D(T_{i+5}) = t + 10$. If T is light, then $D(T_i) = 0$. For an intra-sporadic task, the group deadline is defined in exactly the same way, assuming that all the future subtasks are released as early as possible. As shown in [4], the group deadline can be calculated using a simple formula.

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 to be scheduled in slot t. Priorities are ordered according to the following relation.

$$(d', b', D') \preceq (d, b, D) \equiv [d < d'] \vee [(d = d') \wedge (b > b')] \vee [(d = d') \wedge (b = b') \wedge (D \geq D')]$$

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

According to the above priority definition, T_i has higher priority than U_j if it has an earlier pseudo-deadline.

If T_i and U_j have equal pseudo-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. (Not choosing to schedule T_i could ultimately cause it to be scheduled in the last slot of its window, reducing the the number of slots available to its successor by one.) If T_i and U_j have equal pseudo-deadlines and their b-bits are equal, 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 larger group deadline. Note that the subtask with the larger 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.

4 Proof of optimality of PD^2

As stated earlier, the fact that PD^2 correctly schedules intra-sporadic tasks does not follow as a simple corollary from previous work. Before presenting our proof, we discuss one "obvious" approach and why it is problematic.

Why demand-based arguments are problematic. For uniprocessor scheduling algorithms, one can often reduce sporadic systems to asynchronous, periodic systems in the following way. Consider a scheduling algorithm A that has been shown to be correct for asynchronous, 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. 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 that is in accordance with the asynchronous, periodic task model up to time t_d (see Figure 4(a)). Moreover, right-shifting such jobs in S can only increase demand near time t_d. Thus, a deadline must be missed at time t_d, a contradiction.

This argument cannot be applied in a Pfair-scheduled multiprocessor system in which subtasks are scheduled by pseudo-deadline. To see why, consider Figure 4(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, causing a presumed future missed deadline to be met.

Proof strategy. Our proof strategy is as follows. Assume that PD^2 is not optimal for scheduling intra-sporadic tasks. This implies that there exists a feasible task system not schedulable using PD^2. We consider one such task system with certain properties. By reasoning about lags in the PD^2 schedule, we achieve a contradiction of some property of the task system — thus showing that our starting assumption is wrong. Only light tasks are considered in detail in this section; needed lemmas involving heavy tasks are proved in Appendix C. The following property of light tasks is used in our proof.

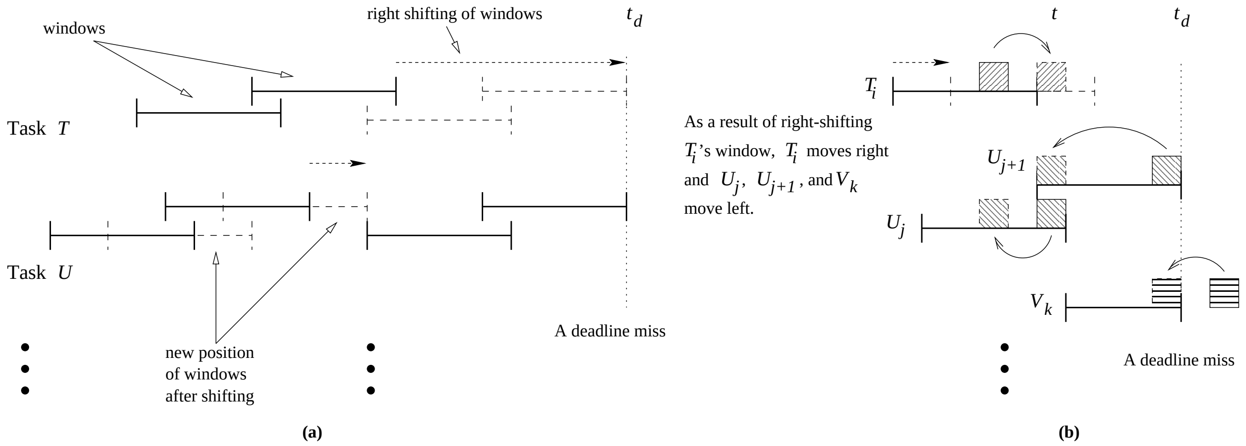

Figure 4: **(a)** Starting with a sporadic task set that misses a deadline at t_d, we can right-shift all windows towards t_d. Intuitively, we *should* get an asynchronous, periodic task system that misses a deadline at t_d. **(b)** Unfortunately, right-shifting a window need not increase future demand. 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.

(P2) For a light task T, $d(T_{i+1}) > d(T_i) + 1$ for all i.

Proof: In [4], we showed that each PF-window of a light task is of length at least three. Therefore, (8) implies that $d(T_{i+1}) \geq r(T_{i+1}) + 2$. By (5) and (6), $r(T_{i+1}) \geq d(T_i)$. Therefore, $d(T_{i+1}) > d(T_i) + 1$. □

Generalized intra-sporadic task systems. In our proof, we consider task systems obtained by removing subtasks from an intra-sporadic task system. Note that such a task system may no longer be intra-sporadic (see Figure 5). To circumvent this problem, we define a more general model called the *generalized* intra-sporadic task model, and show that PD^2 can optimally schedule task systems in this model. In a generalized intra-sporadic 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, $r(T_k)$ 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, $r(T_k)$ must be at least $\left\lfloor \frac{k-1}{wt(T)} \right\rfloor$. Thus, for every generalized intra-sporadic task system τ, there exists an intra-sporadic task system τ' such that τ can be obtained by simply removing certain subtasks in τ'. For generalized intra-sporadic task systems, the parameters in the PD^2 priority definition are defined and calculated in the same way as for intra-sporadic task systems.

A task T of a generalized intra-sporadic task system is said to be *active* at slot t if there exists a subtask T_i such that $e(T_i) \leq t \leq d(T_i)$. (Note that a task that is active at t isn't necessarily scheduled at t.) If a task T is inactive and its most-recently released subtask is T_i, then it can resume execution with any subtask T_k such that $r(T_i) + \left\lfloor \frac{k-1}{wt(T)} \right\rfloor - \left\lfloor \frac{i-1}{wt(T)} \right\rfloor \leq t$. 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_{i+3} is the successor of T_{i+1} in Figure 5).

Some properties of generalized intra-sporadic task systems are stated below. (P4) is proved in Appendix B.

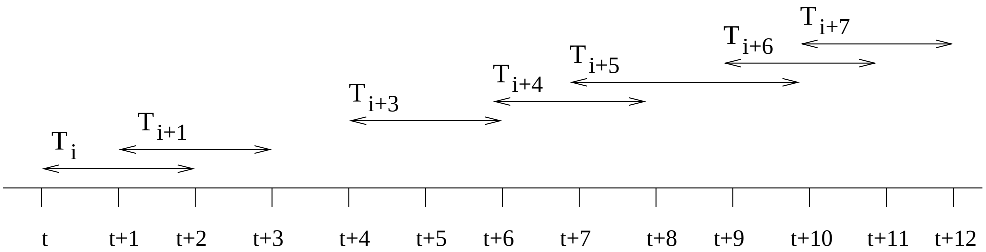

Figure 5: The PF-windows of a generalized intra-sporadic task T with weight $T.e/T.p = 8/11$. Subtask T_{i+2} is removed and T_{i+4} becomes eligible one unit late. (Because T_{i+2} is missing, this is not an intra-sporadic task.)

(P3) Removing a subtask from a generalized intra-sporadic task system results in another generalized intra-sporadic task system. (This follows from the definition of a generalized intra-sporadic task system.)

(P4) Let T_i be a subtask such that $b(T_i) = 1$, and let T_k be the successor of T_i. If $r(T_k) > d(T_i)$, then $flow(T, d(T_i)) + flow(T, d(T_i) + 1) \leq wt(T)$.

Henceforth, when we refer to a task system, we mean a generalized intra-sporadic task system.

In our proof, we focus on slots in which one or more processors are idle. In a schedule S, if k processors are idle at time slot t, then we say that there are k *holes* in S at slot t.

Given the above definitions, we are now ready to prove the optimality of PD^2 for scheduling generalized intra-sporadic tasks. If PD^2 is not optimal, then there exists a task system that is feasible and yet misses a deadline under PD^2. This implies that there exists a time slot t_d and a task system τ defined as follows.

Definition 1 t_d is defined as the minimal time slot at which some task system misses a deadline under PD^2, *i.e.*, some such task system misses a deadline at t_d, and no such task system misses a deadline prior to t_d. □

We define the *rank* of a task system τ to be the sum of the eligibility times of all the subtasks in τ with deadlines at most t_d.

Definition 2 τ is a feasible task system with the following properties.

(T1) τ misses a deadline under PD^2 at t_d.

(T2) Among all feasible task systems that miss a deadline under PD^2 at t_d, no task system releases fewer subtasks in $[0, t_d]$ than τ.

(T3) No task system satisfying (T1) and (T2) has a larger rank than τ. □

By (T1), (T2), and Definition 1, exactly one subtask in τ misses its deadline: if several subtasks miss their deadline, all but one can be removed and the remaining subtask will still miss its deadline, contradicting (T2).

In the rest of this section, we use S to denote the PD^2 schedule of τ. (Note that we have assumed that S is an invalid schedule and our aim is to show that it is valid.) We henceforth assume that ties among subtasks in τ, not resolved by PD^2, are resolved consistently (for example, by using task identifiers).

We now prove some additional properties about τ.

Lemma 1 *There are no subtasks with deadlines greater than t_d in τ.*

Proof: If τ contains a subtask with a deadline greater than t_d, then it can be removed without causing the missed deadline to be met. This contradicts (T2). $\square$

Lemma 2 *Let T_i be any subtask in τ and let t be the slot at which it is scheduled in S. Then, the eligibility time of T_i, $e(T_i)$, is at least $min(r(T_i), t)$.*

Proof: Suppose that $e(T_i)$ is less than $min(r(T_i), t)$. Consider the task system τ' obtained from τ with the following change: the eligibility time $e'(T_i)$ in τ' is $min(r(T_i), t)$. (Note that τ' is feasible. This follows from the feasibility proof for intra-sporadic task systems, which gives us a schedule in which each subtask is scheduled in its PF-window.) Note that τ' has a larger rank than τ. It is easy to see that the the relative priorities of the subtasks do not change for any slot $u \in [0, t_d]$, and therefore, the PD^2 schedules for τ' and τ are the same. Therefore, τ' misses a deadline at t_d as well. This contradicts (T3). $\square$

Corollary 1 *Let subtask T_i be scheduled in slot t in S. If either $t < d(T_i)$ or $t = d(T_i) \wedge b(T_i) = 0$, then the eligibility time of the successor of T_i is at least $t + 1$.*

Proof: Because T_i is scheduled at t, the successor of T_i is scheduled after slot t. Also, if either $d(T_i) > t$ or $d(T_i) = t \wedge b(T_i) = 0$, then by (6), we have $r(T_{i+1}) > t$ and hence, $r(T_k) > t$ for all $k > i$. In particular, if T_k is the successor of T_i, then $r(T_k) \geq t + 1$. Therefore, by Lemma 2, the eligibility time of T_k is at least $t + 1$. $\square$

We now state and prove some properties about S and the total lag of the system at different times in S.

Lemma 3 *There are no holes in slot t_d in S.*

Proof: If there were a hole in slot t_d, then the subtask that misses its deadline at t_d would have been scheduled there by PD^2. Contradiction. $\square$

Lemma 4 $LAG(\tau, t_d + 1) > 0$.

Proof: By Lemma 1, the deadline of each subtask in τ is at most t_d. Therefore, each task in τ either is inactive at t_d or has a subtask with a deadline at t_d.

Consider the lag of each task at $t_d + 1$. Each task that is inactive at t_d has a lag of 0. Each task that meets its deadline at t_d also has a lag of 0. The remaining task T that misses a deadline at t_d has a positive lag. (In fact, it is exactly one.) To see why, note that because T misses a deadline under PD^2, the number of subtasks of T scheduled in $[0, t_d]$ in S is one less than the number of subtasks of T scheduled in the ideal system. This implies that the number of allocations it gets in S is less than the total flow for T in $[0, t_d]$. Therefore, by (10), T's lag at $t_d + 1$ is positive. Hence, the total lag is also positive. $\square$

Lemma 5 *If $LAG(\tau, t) < LAG(\tau, t+1)$, then there is a hole in slot t in S.*

Proof: We prove this by contradiction. Suppose that there are no holes in slot t in S, *i.e.*, there are M tasks scheduled in slot t. By (14), $LAG(\tau, t+1) = LAG(\tau, t) + \sum_{T \in \tau} flow(T, t) - M$.

Because $flow(T, t) \le wt(T)$ (by (P1)), we have $\sum_{T \in \tau} flow(T, t) \le \sum_{T \in \tau} wt(T)$, which by (9) implies that $\sum_{T \in \tau} flow(T, t) \le M$. Therefore, $LAG(\tau, t+1) \le LAG(\tau, t)$. Contradiction. □

Note that the above lemma holds for any schedule of any task system, because we have not used any properties specific to either τ or the PD^2 algorithm.

Lemma 6 $LAG(\tau, t_d) > 0$.

Proof: By Lemma 3, there are no holes in slot t_d. By Lemma 5, this implies that $LAG(\tau, t_d) \ge LAG(\tau, t_d + 1)$. Therefore, by Lemma 4, $LAG(\tau, t_d) > 0$. □

Lemma 7 *In schedule S, there exists $u \in [0, t_d - 1]$ such that $LAG(\tau, u) \le 0$ and $LAG(\tau, u+1) > 0$.*

Proof: This follows from the fact that $LAG(\tau, 0) = 0$ and $LAG(\tau, t_d) > 0$ (Lemma 6). □

We now define some additional terms that are used in the rest of our proof. Later, in Lemmas 14 and 15, we show that for any $t \in [0, t_d - 1]$, if $LAG(\tau, t) \le 0$ and $LAG(\tau, t+1) > 0$, then there exists $u \in [t+2, t_d+1]$ such that $LAG(\tau, u) \le 0$. This allows us to conclude that $LAG(\tau, t_d + 1) \le 0$, contradicting Lemma 4. To establish these properties, we consider certain schedules obtained by removing some subtasks. (Note that, by (P3), the resulting task system is a generalized intra-sporadic task system as well). Removing a subtask may cause other subtasks to shift earlier in the schedule. We now introduce some formal terms used to describe these shifts.

The term $X^{(i)}$ is used to denote a subtask of any task in τ. Assume that removing subtask $X^{(1)}$ scheduled at slot t_1 in S causes the subtask $X^{(2)}$ to be shifted from slot t_2 to slot t_1, which in turn may cause other shifts. We call this shift a *displacement* and represent it by a 4-tuple $\langle X^{(1)}, t_1, X^{(2)}, t_2 \rangle$. This is equivalent to saying that $X^{(2)}$ originally scheduled at t_2 in S displaces $X^{(1)}$ scheduled at t_1 in S. Note that if $t_2 = t_1$, then the displacement is void and is equivalent to simply removing $X^{(1)}$ from slot t_1. A displacement $\langle X^{(1)}, t_1, X^{(2)}, t_2 \rangle$ is *valid* iff $e(X^{(2)}) \le t_1$. Because there can be a cascade of shifts, we may have a *chain* of displacements. This chain is represented by a sequence of 4-tuples. An example of this is given in Figure 6.

The next two lemmas give some properties about displacements. Lemma 8, proved in Appendix A, states that only left-shifts can result from removing a subtask, as shown in Figure 6. Lemma 9 states that a left-shift into a slot with a hole (before the shift) can only happen if the subtasks being shifted are of the same task.

Lemma 8 *Let $X^{(1)}$ be a subtask that is removed from τ, and let the chain of displacements caused in S by this removal be $C = D_1, D_2, \ldots, D_k$, where $D_i = \langle X^{(i)}, t_i, X^{(i+1)}, t_{i+1} \rangle$. Then $t_{i+1} > t_i$ for all $i \in [1, k]$.*

Lemma 9 *Let $\langle X^{(1)}, t_1, X^{(2)}, t_2 \rangle$ be a valid displacement in any PD^2 schedule. If $t_1 < t_2$ and there is a hole in slot t_1 in that schedule, then $X^{(2)}$ is the successor of $X^{(1)}$.*

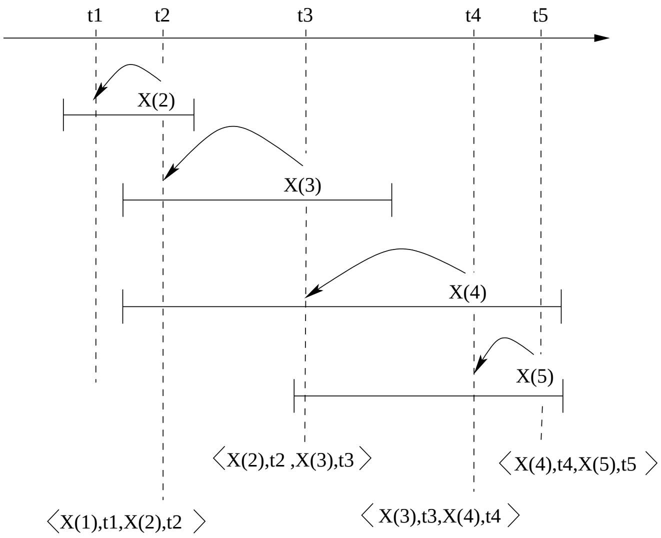

Figure 6: Figure shows a chain of 4 displacements, caused by removing $X^{(1)}$ that was scheduled in slot t_1.

Proof: Assume that $X^{(2)}$ is not the successor of $X^{(1)}$. Because the displacement is valid, $e(X^{(2)}) \leq t_1$. Because there is a hole in slot t_1, this implies that $X^{(2)}$ should have been scheduled there by PD^2. Contradiction. $\square$

The next three lemmas state conditions that must hold for S. In each of these lemmas, we show that if these conditions do not hold, then a subtask can be removed from τ without causing the missed deadline in S to be met, contradicting (T2). (Note that removing a subtask might cause other subtasks to be left-shifted. A sequence of such left-shifts may possibly lead to the missed deadline being met.) We remind the reader that, by our assumption given earlier, ties among subtasks are resolved consistently. Thus, if task system τ' is obtained by removing a subtask from τ, then the relative priorities of two subtasks in τ' are the same as in τ.

Lemma 10 *Suppose there is a hole in slot $t \in [0, t_d]$ in S. Let U_j be a subtask scheduled at $t' < t$. If the eligibility time of the successor of U_j is at least $t+1$, then either $d(U_j) < t$ or $d(U_j) = t \wedge b(U_j) = 1$.*

Proof: We prove this by contradiction. Assume that the following is true.

$$d(U_j) > t \;\text{ or }\; d(U_j) = t \wedge b(U_j) = 0 \tag{15}$$

Let τ' be the task system obtained by removing the subtask U_j from τ. Let S' denote the PD^2 schedule for τ'.

Let the chain of displacements caused by removing U_j be $D_1, D_2, \ldots, D_k$, where $D_i = \langle X^{(i)}, t_i, X^{(i+1)}, t_{i+1} \rangle$, $X^{(1)} = U_j$ and $t_1 = t'$. By Lemma 8, $t_{i+1} > t_i$ for all $i \in [1, k]$. Note that at slot t_i, the priority of subtask $X^{(i)}$ is higher than the priority of $X^{(i+1)}$, because $X^{(i)}$ was chosen over $X^{(i+1)}$ in S. Thus, because $X^{(1)} = U_j$, by (15), for each subtask $X^{(i)}, i \in [1, k+1]$, either $d(X^{(i)}) > t$ or $d(X^{(i)}) = t \wedge b(X^{(i)}) = 0$. Therefore, by Corollary 1, we have the following property.

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

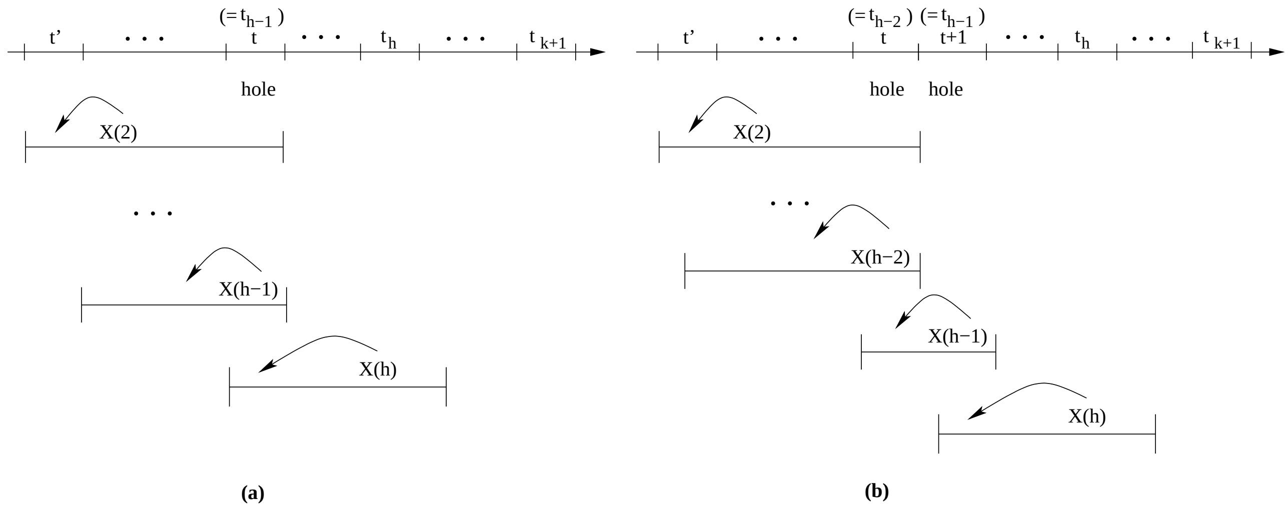

Figure 7: **(a)** Lemma 10. There is a hole in slot t. $X^{(h)}$ must be the successor of $X^{(h-1)}$. **(b)** Lemma 11. $X^{(h-2)}$, $X^{(h-1)}$, and $X^{(h)}$ must be consecutive subtasks of the same task because there are holes in slots t and $t+1$.

We now show that the displacements do not extend beyond slot t, which implies that a deadline is missed at t_d in S', contradicting (T2). Suppose that these displacements extend beyond slot t, *i.e.*,

$$t_{k+1} > t. \tag{16}$$

Refer to Figure 7(a). Let h be the smallest $i \in [2, k+1]$ such that $t_i > t$. Therefore, $t_{h-1} \leq t$. Because D_{h-1} is a valid,

$$e(X^{(h)}) \leq t_{h-1}.$$

Now, if $t_{h-1} < t$, then by the above expression, $X^{(h)}$ is eligible at t. Because there is a hole in slot t, this implies that $X^{(h)}$ should have been scheduled at t in S instead of at $t_h > t$. Therefore, $t_{h-1} = t$. Because there is a hole in slot t, by Lemma 9, $X^{(h)}$ is the successor of $X^{(h-1)}$. However, because $e(X^{(h)}) \leq t_{h-1} = t$, by (E), $X^{(h)}$ cannot be the successor of $X^{(h-1)}$.

Thus, we have a contradiction of (16), implying that no subtask scheduled after t can get left-shifted. Hence, a deadline is still missed at t_d in S', contradicting (T2). Thus, either $d(U_j) < t$ or $d(U_j) = t \ \wedge\ b(U_j) = 1$. □

Lemma 11 *Let U_j be a subtask of a light task scheduled at $t' < d(U_j)$ in S. If the eligibility time of the successor of U_j is at least $d(U_j)+1$, then there cannot be holes in both $d(U_j)$ and $d(U_j)+1$.*

Proof: Note that by Lemma 1, $d(U_j) \leq t_d$. Therefore, $t' < t_d$. Let $d(U_j) = t$. If $t = t_d$, then t satisfies the stated requirement because there is no hole in slot t_d (by Lemma 3). In the rest of the proof, we assume that $t < t_d$, and hence $t+1 \leq t_d$. Suppose that there are holes in both t and $t+1$. Because there is a hole in slot t and (from the statement of the lemma) the eligibility time of the successor of U_j is at least $t+1$, by Lemma 10, either $d(U_j) < t$ or $d(U_j) = t \ \wedge\ b(U_j) = 1$. Because $d(U_j) = t$, the latter in fact must hold, *i.e.*,

$d(U_j) = t \ \wedge \ b(U_j) = 1$. We now show that U_j can be removed without causing the missed deadline to be met, contradicting (T2). In particular, we show that the sequence of left-shifts caused by removing U_j does not extend beyond slot $t + 1$.

Let the chain of displacements caused by removing U_j be $D_1, D_2, \ldots, D_k$, where $D_i = \langle X^{(i)}, t_i, X^{(i+1)}, t_{i+1} \rangle$, $X^{(1)} = U_j$ and $t_1 = t'$. By Lemma 8, we have $t_{i+1} > t_i$ for all $i \in [1, k]$. Also, the priority of $X^{(i)}$ is greater than the priority of $X^{(i+1)}$ at t_i, because $X^{(i)}$ was chosen over $X^{(i+1)}$ in S. Because $d(U_j) = t \ \wedge \ b(U_j) = 1$, this implies the following property.

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

Suppose this chain of displacements extends beyond $t+1$, *i.e.*, $t_{k+1} > t+1$. Let h be the smallest $i \in [1, k+1]$ such that $t_i > t + 1$. Then, $t_{h-1} \leq t + 1$.

If $t_{h-1} < t + 1$, then $X^{(h)}$ is eligible to be scheduled in slot $t + 1$ because $e(X^{(h)}) \leq t_{h-1}$ (by the validity of displacement D_{h-1}). Because there is a hole in slot $t + 1$ in S, $X^{(h)}$ should have been scheduled there in S. Therefore, $t_{h-1} = t + 1$ and by Lemma 9, $X^{(h)}$ must be the successor of $X^{(h-1)}$. By similar reasoning, because there is a hole in slot t, $t_{h-2} = t$ and $X^{(h-1)}$ must be the successor of $X^{(h-2)}$ (see Figure 7(b)).

By (P), either $d(X^{(h-2)}) > t$ or $d(X^{(h-2)}) = t$ and $X^{(h-2)}$ is the subtask of a light task. In either case, $d(X^{(h-1)}) > t + 1$. To see this, note that if $d(X^{(h-2)}) > t$, then because $X^{(h-1)}$ is the successor of $X^{(h-2)}$, by (6), $d(X^{(h-1)}) > t + 1$. On the other hand, if $d(X^{(h-2)}) = t$ and $X^{(h-2)}$ is the subtask of a light task, then, by (P2), $d(X^{(h-1)}) > t + 1$. Now, because $X^{(h-1)}$ is scheduled at $t + 1$, by Corollary 1, 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 D_{h-1} is not valid. Thus, the chain of displacements cannot extend beyond $t + 1$ and because $t + 1 \leq t_d$, removing U_j cannot cause a missed deadline at t_d to be met. This contradicts (T2). Therefore, there cannot be holes in both t and $t + 1$. □

The following lemma is the counterpart of Lemma 11 for heavy tasks. It is proved in Appendix C.

Lemma 12 *Let U_j be a subtask of a heavy task scheduled in slot t' in S. If $t' < d(U_j)$ and $b(U_j) = 1$, and if the eligibility time of the successor of U_j is at least $d(U_j) + 1$, then there exists a slot $t \in [d(U_j), min(D(U_j), t_d)]$ such that there is no hole in slot t.*

We use the following property in the proof of our main lemma.

Lemma 13 *If $LAG(\tau, t) < LAG(\tau, t + 1)$ for some t in S, then there exists a task that is active at t but not scheduled at t.*

Proof: Let I denote the set of tasks that are inactive at slot t. Let A denote the set of tasks that are active but not scheduled in slot t, and let B denote the set of active tasks scheduled in slot t. Note that these three sets are disjoint and $I \cup A \cup B = \tau$. Our aim is to show that A is non-empty.

By Lemma 5, there are holes in slot t. Let the number of these holes be h. Then, the number of allocations in slot t in S is $M - h$, *i.e.*, $\sum_{T \in \tau} S(T, t) = M - h$. By (14), $LAG(\tau, t + 1) = LAG(\tau, t) + \sum_{T \in \tau}(flow(T, t) - S(T, t))$.

Thus, because $LAG(\tau, t) < LAG(\tau, t+1)$, we have $\sum_{T \in \tau} flow(T, t) > M - h$. Because $\sum_{T \in I} flow(T, t) = 0$, it follows that $\sum_{T \in A \cup B} flow(T, t) > M - h$. Therefore, by (P1), $\sum_{T \in A \cup B} wt(T) > M - h$. Because the number of tasks scheduled at t is $M - h$, $|B| = M - h$. Because each task's weight is at most one, $\sum_{T \in B} wt(T) \leq M - h$. Therefore, $\sum_{T \in A} wt(T) > 0$. This implies that A is not empty. $\square$

In the proof of the next lemma, we concentrate on certain subtasks defined as follows.

Definition 3 Subtask U_j is said to be the *critical subtask of U at slot t* iff $e(U_j) \leq t \leq d(U_j)$ and no other subtask U_k of U, where $k > j$, satisfies $e(U_k) \leq t \leq d(U_k)$. $\square$

In the following lemma, we show that if LAG is non-positive at time t and positive at time $t+1$, then it becomes non-positive again at a later time. We then use this to contradict the fact that $LAG(\tau, t_d + 1) > 0$ (Lemma 4). As we shall see in the proof below, if the set A (defined in the statement of the lemma) contains at least one light task, then the next time at which LAG becomes non-positive is $t+2$. However, if A consists only of heavy tasks, then we might need to go a little further, in particular, until the group deadline of a certain subtask. We defer the proof of this case to Appendix C. The main insight here is that if LAG transits from non-positive to positive, then afterwards, it continues to decrease until it becomes non-positive again. In the case of task systems consisting solely of light tasks, this decrease is very fast, while in task systems with heavy tasks, it may be slower. Note that these remarks apply only to task systems like τ.

Lemma 14 *Suppose that $LAG(\tau, t) \leq 0$ and $LAG(\tau, t+1) > 0$ for some $t \in [0, t_d - 1]$. Let A denote the set of tasks that are active but not scheduled in slot t. If any task in A is light, then $LAG(\tau, t+2) \leq 0$.*

Proof: Because $LAG(\tau, t) \leq 0$ and $LAG(\tau, t+1) > 0$, we have the following.

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

By Lemma 5, we have the following property for some $h \geq 1$.

(H) There are h holes in slot t.

Let I denote the set of tasks that are inactive at slot t, and let B denote the set of tasks scheduled in slot t. Note that I, A, and B are disjoint and $I \cup A \cup B = \tau$. By (17) and Lemma 13, A is non-empty.

Let U be any task in A. Because U is active at t, t lies within the eligibility interval of some subtask of U. Therefore, there exists a subtask U_j that is critical at slot t. Let t' be the slot at which U_j is scheduled. We now show that $t' < t$. Suppose $t' > t$. Then there exists a subtask U_k satisfying $e(U_k) \leq t \leq d(U_k)$ such that U_k is scheduled after t and its predecessor (if it exists) is scheduled before t. (Because $U \in A$, no subtask of U is scheduled at t.) Because there is a hole in slot t (by (H)), and because U_k is eligible at t, it should have been scheduled there by PD^2. Thus, there cannot exist any such subtask, implying that $t' < t$.

Note that, by Definition 3, the successor of U_j has an eligibility time of at least $t+1$. Because there is a hole in slot t, by Lemma 10, either $d(U_j) < t$ or $d(U_j) = t \wedge b(U_j) = 1$. Also, by Definition 3, $d(U_j) \geq t$. Therefore,

$d(U_j) = t \ \wedge \ b(U_j) = 1$. This is true for the critical subtask of each task in A. Let C denote the set of critical subtasks of all the tasks in A. Then,

$$\text{for all } U_j \in C,\ d(U_j) = t \ \wedge \ b(U_j) = 1. \tag{18}$$

Because A is not empty, by the statement of the lemma, A contains at least one light task. Let $U \in A$ be light. Because there is a hole in slot $t = d(U_j)$, by Lemma 11, there is no hole in slot $t+1 = d(U_j)+1$. We are now ready to show that $LAG(\tau, t+2) \leq 0$.

Let the sum of the weights of all tasks in set B be W. Then, the total sum of the weights of all tasks in $I \cup A$ is at most $M - W$. For all tasks $T \in I$, $flow(T,t) = 0$. By (P1), this implies that for all tasks $T \in I$, $flow(T,t) + flow(T,t+1) \leq wt(T)$. Consider $U \in A$. Let U_j denote its critical subtask, and let U_k denote the successor of U_j. Because the eligibility time of U_k is at least $t+1$, $r(U_k) \geq t+1 = d(U_j)+1$. Therefore, by (P4), $flow(U,t) + flow(U,t+1) \leq wt(U)$ for all $U \in A$. Hence,

$$\sum_{U \in I \cup A} (flow(U,t) + flow(U,t+1)) \leq \sum_{U \in I \cup A} wt(U) \leq M - W.$$

Also, by (P1), $\sum_{U \in B}(flow(U,t) + flow(U,t+1)) \leq 2\sum_{U \in B} wt(U) = 2W$. Therefore,

$$\sum_{U \in \tau}(flow(U,t) + flow(U,t+1)) \leq 2W + M - W = M + W. \tag{19}$$

By (H), there are h holes in slot t. Because there are no holes in slot $t+1$,

$$\sum_{U \in \tau}(S(U,t) + S(U,t+1)) = M - h + M = 2M - h. \tag{20}$$

Because the number of tasks scheduled at t is $M - h$, and the weight of each task is at most one, we have $\sum_{U \in B} wt(U) \leq M - h$, *i.e.*, $W \leq M - h$. This relation and (19) and (20) imply that

$$\sum_{U \in \tau}(flow(U,t) + flow(U,t+1)) \leq \sum_{U \in \tau}(S(U,t) + S(U,t+1)).$$

Using this relation in the identity (obtained from (14))

$$LAG(\tau, t+2) = LAG(\tau, t) + \sum_{U \in \tau}(flow(U,t) + flow(U,t+1)) - \sum_{U \in \tau}(S(U,t) + S(U,t+1)),$$

and the fact that $LAG(\tau, t) \leq 0$, we have $LAG(\tau, t+2) \leq 0$. □

The following lemma, which is proved in Appendix C, generalizes Lemma 14 by allowing the set A to consist solely of heavy tasks. If τ consists only of light tasks, then by Lemma 13, A has at least one light task. Thus, for systems of light-only tasks, the following lemma is a direct consequence of Lemma 14.

Lemma 15 *For any $t \in [0, t_d - 1]$, if $LAG(\tau, t) \leq 0$ and $LAG(\tau, t+1) > 0$, then there exists $u \in [t+2, t_d+1]$ such that $LAG(\tau, u) \leq 0$.*

By Lemma 7, there exists a $u \in [0, t_d - 1]$ such that $LAG(\tau, u) \leq 0$ and $LAG(\tau, u+1) > 0$. Let v be the largest such u. Because $v \leq t_d - 1$, we have $v + 2 \leq t_d + 1$. By Lemma 15, $LAG(\tau, t) \leq 0$ for some $t \in [v+2, t_d+1]$. Therefore, by Lemmas 4 and 6, t cannot be t_d or $t_d + 1$. Therefore, $t \in [0, t_d - 1]$. Because $LAG(\tau, t_d) > 0$, this contradicts the maximality of v. Therefore, τ cannot exist, *i.e.*, our original supposition that PD^2 is not optimal is incorrect. Thus, we have the following theorem and corollary.

Theorem 1 *PD^2 is optimal for scheduling generalized intra-sporadic tasks on M processors.*

Corollary 2 *PD^2 is optimal for scheduling intra-sporadic tasks on M processors.*

5 Concluding Remarks

In this paper, we have closed the problem left open by us in [5], namely, the optimal scheduling of intra-sporadic tasks on more than two processors. In particular, we have shown that PD^2, the most efficient optimal Pfair scheduling algorithm proposed to date, correctly schedules any feasible intra-sporadic task system on M processors. This paper is the first to show that either sporadic or intra-sporadic tasks can be optimally scheduled on systems of more than two processors.

Two key insights led to our proof. One was the development of a notion of lag for intra-sporadic systems that can be used to sufficiently predict where holes exist in a schedule. The other was the identification of certain minimality conditions (given in Definitions 1 and 2) that facilitate the reasoning. It is these notions that distinguish our proof from others given previously for Pfair and ERfair scheduling algorithms.

The intra-sporadic 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 [10]. The intra-sporadic task model also generalizes that considered in our recent work on scheduling mixed early-release/non-early-release periodic, asynchronous task systems [6].

The results of this paper show that PD^2 can be used to solve the parallel switching problem in ATM networks mentioned in [2]. In this setting, *packets* are to be scheduled, which can be highly jittered (hence the usefulness of rate-based approaches). We also believe that PD^2 can be used to efficiently schedule aperiodic tasks on multiprocessors. In this case, tasks must have the ability to join and leave the system dynamically, *e.g.*, as done in the work on proportional-share scheduling [13] and constant bandwidth servers [1]. In recent related work, Chandra *et al.* considered multiprocessor scheduling algorithms that use variable-sized quanta and that allow tasks to join and leave dynamically [9]. However, the main focus of their work is simulation and performance studies. In particular, they provide no formal correctness proofs for the algorithms they consider.

References

[1] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proc. of the 19th IEEE Real-Time Systems Symposium*, pages 4–13, Dec. 1998.

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

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

[4] J. Anderson and A. Srinivasan. A new look at pfair priorities. Technical Report TR00-023, University of North Carolina at Chapel Hill, Sept. 2000. Available at `http://www.cs.unc.edu/~anderson/papers.html`.

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

[6] J. Anderson and A. Srinivasan. Mixed pfair/erfair scheduling of asynchronous periodic tasks. In *Proc. of the 13th Euromicro Conference on Real-Time Systems*, June 2001. To appear.

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

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

[9] A. Chandra, M. Adler, and P. Shenoy. Deadline fair scheduling: Bridging the theory and practice of proportionate-fair scheduling in multiprocessor servers. In *Proc. of the 7th IEEE Real-Time Technology and Applications Symposium*, June 2001. To appear.

[10] K. Jeffay and S. Goddard. The rate-based execution model. In *Proc. of the 20th IEEE Real-Time Systems Symposium*, pages 304–314, Dec. 1999.

[11] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

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

[13] 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 *Proc. of the 17th IEEE Real-Time Systems Symposium*, pages 288–299, Dec. 1996.

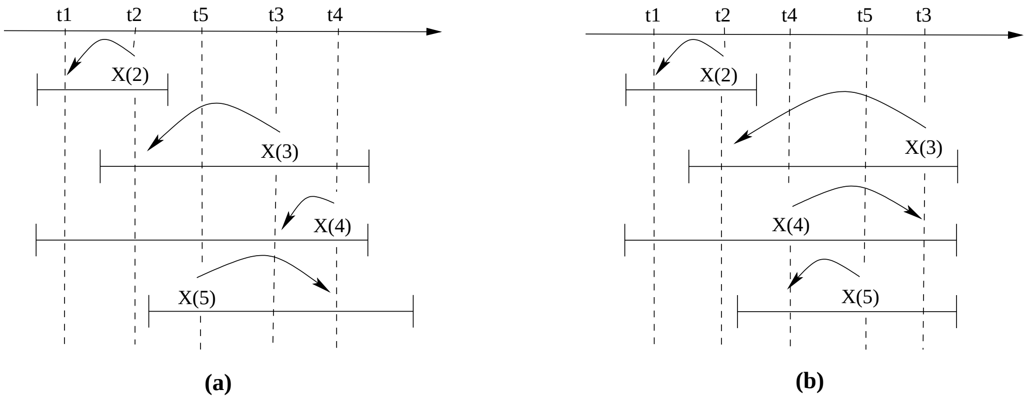

Figure 8: Lemma 8. A chain of $k = 4$ displacements is shown. **(a)** The leftmost right shift occurs when $X^{(5)}$ shifts from t_5 to t_4, *i.e.*, $j = k$. **(b)** The leftmost right shift occurs when $X^{(4)}$ shifts from t_4 to t_3, *i.e.*, $j < k$ (here, $t_j = t_3, t_{j+1} = t_4$, and $t_{j+2} = t_5$).

Appendix A: Proof of Lemma 8

Lemma 8 *Let $X^{(1)}$ be a subtask that is removed from τ, and let the chain of displacements caused in S (the PD^2 schedule for τ) by this removal be $D_1, D_2, \ldots, D_k$, where $D_i = \langle X^{(i)}, t_i, X^{(i+1)}, t_{i+1} \rangle$. Then $t_{i+1} > t_i$ for all $i \in [1, k]$.*

Proof: Let τ' be the task system obtained by removing $X^{(1)}$ from τ, and let S' denote its PD^2 schedule. Note that the last displacement creates a hole in slot t_{k+1} in S'. Suppose $t_{i+1} > t_i$ is not true for some $i \in [1, k]$. Let

$$j = \{i \mid \text{smallest } t_i \text{ such that } t_{i+1} < t_i\}$$

(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 will be as shown in Figure 8(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 choice of j). Because there will be a hole in slot t_{k+1} in S' and $t_{k+1} < t_k$, this contradicts the greedy behavior of the PD^2 algorithm.

If $j < k$, then by choice of j, we have $t_{j+1} < t_j$ and the displacements will be as seen in Figure 8(b). By minimality of t_j, we have $t_{j+2} > t_{j+1}$. Thus, at t_{j+1}, $X^{(j+1)}$ was chosen over $X^{(j+2)}$ in S. After the displacements, $X^{(j+1)}$ is scheduled at t_j and $X^{(j+2)}$ is scheduled at t_{j+1}. This contradicts our assumption that ties among subtasks are broken in a consistent manner, *i.e.*, the same tie-breaks are used in S and S'. Therefore, $t_{i+1} > t_i$ for all $i \in [1, k]$. $\square$

Appendix B: Properties of (Generalized) Intra-sporadic Task Systems

In this appendix, we prove Property (P4), stated earlier in Section 4. We also prove certain other properties that are used in proof of Lemma 15 in Appendix C. All of the properties in this appendix apply to generalized intra-sporadic task systems, except for (B3), which applies only to intra-sporadic tasks.

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

Proof: By (12), $f(T_i, d(T_i)) = i - (\left\lceil \frac{i}{wt(T)} \right\rceil - 1) \times wt(T)$, and $f(T_{i+1}, r(T_{i+1})) = (\left\lfloor \frac{i}{wt(T)} \right\rfloor + 1) \times wt(T) - i$. Because $b(T_i) = 1$, by (5), $\left\lceil \frac{i}{wt(T)} \right\rceil = \left\lfloor \frac{i}{wt(T)} \right\rfloor + 1$. This implies that $f(T_{i+1}, r(T_{i+1})) = \left\lceil \frac{i}{wt(T)} \right\rceil \times wt(T) - i$. Therefore, $f(T_i, d(T_i)) + f(T_{i+1}, r(T_{i+1})) = wt(T)$. □

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

Proof: By (6) and (8), it follows that $r(T_k) > d(T_i) + 1$ for all $k > i + 1$. Similarly, $d(T_j) < d(T_i)$ for all $j < i$. This implies that the slot $d(T_i)$ lies within the PF-window of only one subtask, namely, T_i, and the slot $d(T_i) + 1$ can lie within the PF-window of only one subtask, namely, T_{i+1}. Thus, the contribution to the flow in slot $d(T_i)$ is $f(T_i, d(T_i))$ and the contribution to slot $d(T_i) + 1$ is at most $f(T_{i+1}, r(T_{i+1}))$. Therefore, by (B1), $flow(T, d(T_i)) + flow(T, d(T_i) + 1) \leq wt(T)$. □

(B3) Let T_i be a subtask of an *intra-sporadic* heavy task T and let T_k be a subtask such that $k > i$ and $r(T_k) \leq D(T_i)$. Then, $f(T_i, d(T_i)) + f(T_k, r(T_k)) \leq wt(T)$.

Proof: If $b(T_i) = 0$, then $D(T_i) = d(T_i)$. In this case, $r(T_k) > D(T_i)$ holds, since (6) implies $r(T_k) > d(T_i)$ (thus, no task T_k exists such that $k > i$ and $r(T_k) \leq D(T_i)$). In the rest of the proof, we assume that $b(T_i) = 1$. Note that, by the definition of the group deadline, because $r(T_k) \leq D(T_i)$, for all $j \in [i+1, k-1]$, $|w(T_j)| = 2$ and $b(T_j) = 1$. Because $|w(T_j)| = 2$, we have $d(T_j) = r(T_j) + 1$. Because the total flow for a subtask is one, this implies that

$$\text{for all } j \in [i+1, k-1], f(T_j, r(T_j)) + f(T_j, d(T_j)) = 1. \tag{21}$$

Because $b(T_j) = 1$, by (B1), we have

$$\text{for all } j \in [i, k-1], f(T_j, d(T_j)) + f(T_{j+1}, r(T_{j+1})) = wt(T).$$

Therefore, $\sum_{j=i}^{k-1} (f(T_j, d(T_j)) + f(T_{j+1}, r(T_{j+1}))) = (k - i) \times wt(T)$. Rewriting, we get

$$f(T_i, d(T_i)) + f(T_k, r(T_k)) + \sum_{j=i+1}^{k-1} (f(T_j, r(T_j)) + f(T_j, d(T_j))) = (k - i) \times wt(T).$$

By (21), this implies that

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

Therefore,

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

Because $wt(T) \leq 1$ for all T, we have $f(T_i, d(T_i)) + f(T_k, r(T_k)) \leq wt(T)$. $\square$

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

Proof: Note that, by the definition of a generalized intra-sporadic task, there is an intra-sporadic task U such that $wt(U) = wt(T)$, all subtasks between U_i and U_k are present, and $r(U_k) = r(T_k)$. Therefore, $r(U_k) \leq D(U_i)$. By (B3), $f(U_i, d(U_i)) + f(U_k, r(U_k)) \leq wt(U)$. The flows for each subtask of T are the same as for the corresponding subtask of U. Therefore, $f(T_i, d(T_i)) + f(T_k, r(T_k)) \leq wt(T)$. $\square$

(B5) 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 $r(T_k) \geq u$, where $u \in [d(T_i) + 1, D(T_i)]$, then $flow(T, d(T_i)) + flow(T, u) \leq wt(T)$.

Proof: Because $b(T_i) = 1$, by the definition of the group deadline, $D(T_i) > d(T_i)$. Because $u > d(T_i)$ and T_k is the successor of T_i, if $r(T_k) > u$, then $flow(T, u) = 0$. Hence, by (P1), $flow(T, d(T_i)) + flow(T, u) \leq wt(T)$. The other possibility is that $r(T_k) = u$, which implies that $r(T_k) \leq D(T_i)$. In this case, by (B4), $f(T_i, d(T_i)) + f(T_k, r(T_k)) \leq wt(T)$. Therefore, $flow(T, d(T_i)) + flow(T, u) \leq wt(T)$. $\square$

(P4) Let T_i be a subtask such that $b(T_i) = 1$ and let T_k be the successor of T_i. If $r(T_k) \geq d(T_i) + 1$, then $flow(T, d(T_i)) + flow(T, d(T_i) + 1) \leq wt(T)$.

Proof: If $k = i + 1$, then by (B2), $flow(T, d(T_i)) + flow(T, d(T_i) + 1) \leq wt(T)$. Also, if $r(T_k) > d(T_i) + 1$, then $flow(T, d(T_i) + 1) = 0$. Hence, by (P1), $flow(T, d(T_i)) + flow(T, d(T_i) + 1) \leq wt(T)$.

In the rest of the proof, we assume that $k > i + 1$ and $r(T_k) = d(T_i) + 1$. We first show that T must be heavy. If T is light, then by (P2), we have $d(T_{i+1}) > d(T_i) + 1$. By (6), we also have $r(T_k) \geq d(T_{i+1})$ and therefore, $r(T_k) > d(T_i) + 1$, which contradicts $r(T_k) = d(T_i) + 1$.

Thus, T is heavy. Because $b(T_i) = 1$, by the definition of the group deadline, $D(T_i) > d(T_i)$. Therefore, because $r(T_k) = d(T_i) + 1$, we have $r(T_k) \leq D(T_i)$. Thus, by (B4), $flow(T, d(T_i)) + flow(T, d(T_i) + 1) \leq wt(T)$. $\square$

Appendix C: Proofs of Lemmas 12 and 15

Lemma 12 *Let U_j be a subtask of a heavy task scheduled in slot t' in S. If $t' < d(U_j)$ and $b(U_j) = 1$, and if the eligibility time of the successor of U_j is at least $d(U_j) + 1$, then there exists a slot $t \in [d(U_j), min(D(U_j), t_d)]$ such that there is no hole in slot t.*

Proof: Note that by Lemma 1, $d(U_j) \leq t_d$. Therefore, $t' < t_d$. If $min(D(U_j), t_d) = t_d$, then by Lemma 3, $t = t_d$ satisfies the stated requirement. In the rest of the proof, assume that $D(U_j) < t_d$. Let $u = d(U_j)$ and $v = D(U_j)$. Because $b(U_j) = 1$, by the definition of the group deadline, we have $D(U_j) > d(U_j)$, *i.e.*,

$$u < v. \tag{22}$$

Suppose that the following property holds.

(H) There is a hole in slot t for all $t \in [u, v]$.

Assuming (H), we show that U_j can be removed without causing the missed deadline to be met, which contradicts (T2). Let the chain of displacements caused by removing U_j be $D_1, D_2, \ldots, D_k$, where $D_i = \langle X^{(i)}, t_i, X^{(i+1)}, t_{i+1} \rangle$, $X^{(1)} = U_j$, and $t_1 = t'$. By Lemma 8, we have $t_{i+1} > t_i$ for all $i \in [1, k-1]$. Also, the priority of $X^{(i)}$ is greater than the priority of $X^{(i+1)}$ at t_i because $X^{(i)}$ is scheduled at t_i in S. Therefore, for all $i \in [2, k+1]$, one of the following holds:

(a) $d(X^{(i)}) > u$,

(b) $d(X^{(i)}) = u \ \wedge \ b(X^{(i)}) = 0$, or

(c) $d(X^{(i)}) = u \ \wedge \ b(X^{(i)}) = 1 \ \wedge \ D(X^{(i)}) \leq v$.

We now show that the displacements do not extend beyond v. This will allow us to conclude that U_j can be removed without causing the missed deadline to be met. Suppose, to the contrary, that they do extend beyond v, *i.e.*, $t_{k+1} > v$.

Let t_g be the largest t_i such that $t_i < u$ and let t_h be the smallest t_i such that $t_i > v$. (Note that such a t_g exists because $t_1 < u$.) Then, by (H), there are holes in all slots in $[t_{g+1}, t_{h-1}]$. Thus, by Lemma 9,

$$X^{(i+1)} \text{ is the successor of } X^{(i)} \text{ for all } i \in [g+1, h-1]. \tag{23}$$

We now show that the situation is exactly as illustrated in Figure 9(a). For all $i \in [g+1, h-2]$, by Lemma 8, we have $t_{i+1} \geq t_i + 1$. If $t_{i+1} > t_i + 1$, then because **(i)** there is a hole in slot $t_i + 1$, **(ii)** $e(X^{(i+1)}) \leq t_i$, and **(iii)** $X^{(i)}$, the predecessor of $X^{(i+1)}$, is scheduled at t_i, $X^{(i+1)}$ should have been scheduled at $t_i + 1$ in S, a contradiction. Therefore,

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

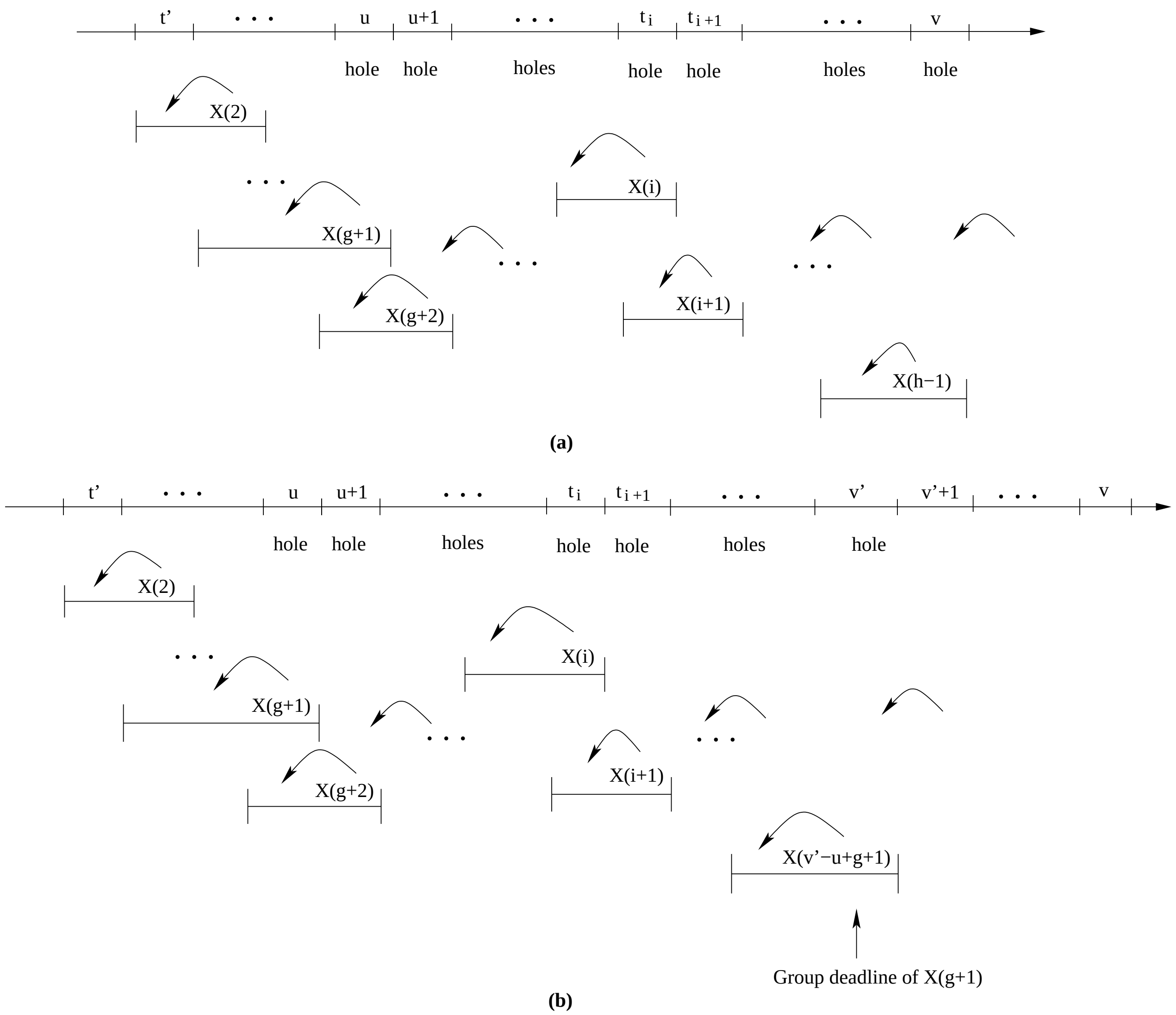

Figure 9: Lemma 12. **(a)** There are holes in all slots in $[u, v]$. $X^{(i)}$ scheduled at t_i displaces $X^{(i-1)}$ scheduled at t_{i-1}. Also, by (25), all the t_i's are consecutive and satisfy $t_i = u + i - (g+1)$. Further, $X^{(h-1)}$ is the subtask scheduled in slot v. **(b)** Case 2. $D(X^{(g+1)}) = v'$. Therefore, either $d(X^{(v'-u+g+1)}) = v' \ \wedge \ b(X^{(v'-u+g+1)}) = 0$ (as depicted) or $d(X^{(v'-u+g+1)}) > v'$.

By similar reasoning, we can show that

$$t_{g+1} = u \ \wedge \ t_{h-1} = v. \tag{24}$$

Therefore,

$$t_i = u + i - (g+1) \text{ for all } i \in [g+1, h-1]. \tag{25}$$

Earlier, we showed that one of (a), (b), or (c) holds for all $i \in [2, k+1]$. If either $d(X^{(g+1)}) > u$ or $d(X^{(g+1)}) = u \ \wedge \ b(X^{(g+1)}) = 0$, then because $X^{(g+1)}$ is scheduled at u, by Corollary 1, $e(X^{(g+2)}) \geq u + 1$ (recall that, by (23), $X^{(g+2)}$ is the successor of $X^{(g+1)}$). In other words, the displacement D_g is not valid. Therefore,

$$d(X^{(g+1)}) = u \ \wedge \ b(X^{(g+1)}) = 1 \ \wedge \ D(X^{(g+1)}) \leq v. \tag{26}$$

We now consider two cases depending on whether $X^{(g+1)}$ is the subtask of a light task or a heavy task.

Case 1: $X^{(g+1)}$ is the subtask of a light task. By (22), $u+1 \leq v$ and hence, by (H), there is a hole in both u and $u+1$. Also, by (24) and (25), we have $v = u + (h-1) - (g+1) = u + h - g - 2$. Hence, because $u < v$ (by (22)), we have $h > g+2$, *i.e.*,

$$h \geq g+3.$$

We now show that the displacement D_{g+2} is not valid.

By (25), $X^{(i)}$ is scheduled at $u + i - (g+1)$ in S for all $i \in [g+1, h-1]$. In particular, $X^{(g+1)}$ is scheduled at u and $X^{(g+2)}$ is scheduled at $u+1$. Observe that $d(X^{(g+1)}) = u$ (by 26), $X^{(g+1)}$ is the subtask of a light task (our assumption for Case 1), and $X^{(g+2)}$ is the successor of $X^{(g+1)}$ (by (23)). Thus, by (P1), $d(X^{(g+2)}) \geq u+2$. Because $X^{(g+2)}$ is scheduled at $u+1$, by Corollary 1, the eligibility time of the successor of $X^{(g+2)}$ is at least $u+2$, *i.e.*, $e(X^{(g+3)}) \geq u+2$. (Note that $X^{(g+3)}$ exists because $h \geq g+3$.) This implies that the displacement D_{g+2} is not valid. Hence, the displacements cannot extend beyond $u+1$. Because $u+1 \leq v < t_d$, U_j can be removed without causing the missed deadline to be met, contradicting (T2). Hence, $X^{(g+1)}$ cannot be the subtask of a light task.

Case 2: $X^{(g+1)}$ is the subtask of a heavy task. Let $v' = D(X^{(g+1)})$. By (26), $v' \leq v$. We now show that the displacements cannot extend beyond v' and hence, cannot go beyond v. By (25), $X^{(i)}$ is scheduled at $u + i - (g+1)$ in S for all $i \in [g+1, h-1]$. By (23), we know that all $X^{(i)}$ for $i \in [g+1, h]$ are subtasks of the same heavy task. We now show that the displacement $D_{v'-u+g+1}$ is not valid.

By (25), $t_{v'-u+g+1} = v'$. Because $X^{(i)}$ is the subtask scheduled at t_i, the subtask scheduled at v' is $X^{(v'-u+g+1)}$. Because $X^{(i+1)}$ is the successor of $X^{(i)}$, by (8), $d(X^{(i)}) > d(X^{(i-1)})$ for all $i \in [g+2, v'-u+g+1]$. Because $d(X^{(g+1)}) = u$ and $b(X^{(g+1)}) = 1$ (by (26)), this implies that

$$d(X^{(i)}) \geq u + i - (g+1) \text{ for all } i \in [g+1, v'-u+g+1]. \tag{27}$$

In particular, $d(X^{(v'-u+g+1)}) \geq v'$.

We now show that if $d(X^{(v'-u+g+1)}) = v'$, then $b(X^{(v'-u+g+1)}) = 0$. In this case, because $d(X^{(v'-u+g)}) < d(X^{(v'-u+g+1)})$, we have $d(X^{(v'-u+g)}) < v'$. By (27), $d(X^{(v'-u+g)}) \geq v'-1$. Therefore, $d(X^{(v'-u+g)}) = v'-1$. Similarly, by induction, $d(X^{(i)}) = u + i - (g+1)$ for all $i \in [g+1, v'-u+g+1]$. (Refer to Figure 9(b).) Because $D(X^{(g+1)}) = v'$, by the definition of the group deadline, $b(X^{(v'-u+g+1)}) = 0$.

Thus, either $d(X^{(v'-u+g+1)}) > v'$ or $d(X^{(v'-u+g+1)}) = v' \wedge b(X^{(v'-u+g+1)}) = 0$. Because $X^{(v'-u+g+1)}$ is scheduled at v', by Corollary 1, the eligibility time of the successor of $X^{(v'-u+g+1)}$ is at least $v'+1$. Therefore, the displacement $D_{v'-u+g+1}$ is not valid. Thus, the displacements cannot extend beyond v'. Because $v' \leq v < t_d$, U_j can be removed without causing the missed deadline to be met, contradicting (T2). Hence, $X^{(g+1)}$ cannot be the subtask of a heavy task.

Thus, we have obtained a contradiction in both cases, implying that (H) is not true, *i.e.*, there exists a slot $t \in [d(U_j), min(D(U_j), t_d)]$ such that there is no hole in slot t. $\Box$

The following claims are used in the proof of the next lemma.

Claim 1 *Let U_j be a subtask scheduled in slot $t \in [0, t_d]$ in S, where $t \leq d(U_j)$. If there is a hole in slot t, then $d(U_j) = t$.*

Proof: We show that if $d(U_j) > t$, then we can remove subtask U_j without causing any displacements. Suppose that removing U_j causes displacements. Because $d(U_j) > t$ and U_j is scheduled at t, by Corollary 1, the eligibility of its successor (if it exists) is at least $t+1$. Therefore, the subtask that shifts into slot t on removal of U_j is not the successor of U_j. If a subtask of any task other than U can shift into slot t, then because there is a hole in slot t in S, it should have been scheduled there by PD^2. Thus, if $d(U_j) > t$, we can remove U_j without causing the missed deadline to be met. This contradicts (T2). Therefore, $d(U_j) = t$. $\Box$

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

Proof: If $t' = t$, then by Lemma 1, $d(U_j) = t$. On the other hand, if $t' < t$, then by Lemma 10, $d(U_j) \leq t$. $\Box$

Lemma 15 *For any $t \in [0, t_d - 1]$, if $LAG(\tau, t) \leq 0$ and $LAG(\tau, t+1) > 0$, then there exists $v \in [t+2, t_d+1]$, such that $LAG(\tau, v) \leq 0$.*

Proof: Because $LAG(\tau, t) \leq 0$ and $LAG(\tau, t+1) > 0$, we have the following.

$$LAG(\tau, t) < LAG(\tau, t+1) \tag{28}$$

Thus, by Lemma 5, we have the following property.

(H) There are holes in slot t.

Let I denote the set of tasks that are inactive at slot t. Let A denote the set of tasks that are active but not scheduled in slot t and let B denote the set of tasks scheduled in slot t. Note that these three sets are disjoint and $I \cup A \cup B = \tau$. By (28) and Lemma 13, A is non-empty.

Let U be any task in A. Because U is active at t, t lies within the eligibility interval of some subtask of U. Therefore, there exists a subtask U_j that is critical at slot t (refer to Definition 3 in Section 4). Let t' be the slot at which U_j is scheduled. We now show that $t' < t$. Suppose $t' > t$. Then there exists a subtask U_k satisfying $e(U_k) \leq t \leq d(U_k)$ such that U_k is scheduled after t and its predecessor (if it exists) is scheduled before t. (Because $U \in A$, no subtask of U is scheduled at t.) Because there is a hole in slot t (by (H)), and because U_k is eligible at t, it should have been scheduled there by PD^2. Thus, there cannot exist any such subtask, implying that $t' < t$.

Note that, by Definition 3, the successor of U_j has an eligibility time of at least $t+1$. Because there is a hole in slot t, by Lemma 10, either $d(U_j) < t$ or $d(U_j) = t \wedge b(U_j) = 1$. Also, by Definition 3, $d(U_j) \geq t$. Therefore, $d(U_j) = t \wedge b(U_j) = 1$. This is true for the critical subtask of each task in A. Let C denote the set of critical subtasks of all the tasks in A. Then,

$$\text{for all } U_j \in C,\ d(U_j) = t \wedge b(U_j) = 1. \tag{29}$$

If any task $U \in A$ is light, then by Lemma 14, $LAG(\tau, t+2) \leq 0$, which establishes our proof obligation. In the rest of this proof, we assume that all the tasks in A are heavy.

Let L_i be the lowest-priority subtask in C. Then, by (29),

$$\text{for each subtask } U_j \text{ in } C,\ d(U_j) = t \wedge b(U_j) = 1 \wedge D(U_j) \geq D(L_i). \tag{30}$$

Note that, because L_i is a critical subtask, the eligibility time of L_k, the successor of L_i, is at least $t+1$. Thus, by Lemma 12, there exists a slot in $[t, \min(D(L_i), t_d)]$ such that there is no hole in that slot. Let u be the slot defined as follows.

(U) u is the earliest slot in $[t, \min(D(L_i), t_d)]$ such that there is no hole in u.

Figure 10 depicts this situation. By our choice of u and by (H),

$$u \geq t+1, \tag{31}$$

and there are holes in all slots in $[t, u-1]$. We now establish the following property about tasks in A.

Claim 3 *All tasks in A are inactive over the interval* $[t+1, u-1]$.

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

Note that because $V \in A$, no subtask of V is scheduled in slot t. Let V_i be the earliest subtask of V scheduled in $[t+1, u-1]$ and let v be the slot in which it is scheduled. Because there is hole in slot v, by Lemma 1, $d(V_i) = v$. By (6) and (8), this implies that $r(V_i) < v$ and hence, $e(V_i) < v$. Thus, because there are holes in all slots in $[t, v-1]$, it should have been scheduled earlier. Contradiction. Hence, no subtask of any task in A is scheduled in $[t, u-1]$ (see Figure 10). Moreover, because there are holes in all slots in $[t, u-1]$, the earliest slot after t at which a subtask of a task in A is eligible to be scheduled is u. By (29), this implies that all the tasks in A 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 3, $r(U_k) \geq u$. Furthermore, by (29), (30), (31), and (U), $d(U_j) = t < u \leq D(U_j)$. Hence, by property (B5), $flow(U,t) + flow(U,u) \leq wt(U)$. Because this argument is applicable to all tasks in A, we have

$$\text{for all } U \in A,\ flow(U,t) + flow(U,u) \leq wt(U). \tag{32}$$

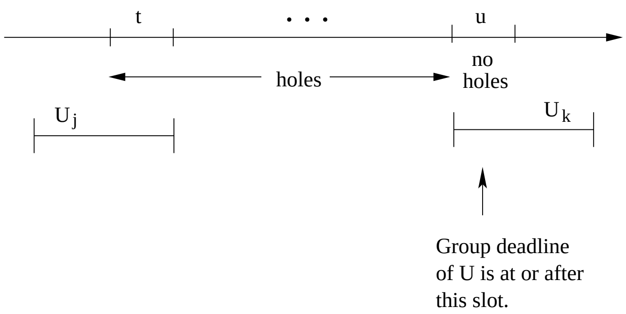

Figure 10: Lemma 15. U_j is the critical subtask of a task in A and U_k is the successor of U_j. There are holes in all the slots in $[t, u-1]$ and there is no hole in slot u. The earliest time at which U_k's PF-window starts is u, *i.e.*, $r(U_k) \geq u$.

We now show that $LAG(\tau, v+1) \leq LAG(\tau, v)$ for all $v \in [t+1, u-1]$.

Claim 4 *For all* $v \in [t+1, u-1]$, $LAG(\tau, v+1) \leq LAG(\tau, v)$.

Proof: If the interval $[t+1, u-1]$ is empty, then the claim is vacuously true, so assume that $[t+1, u-1]$ is nonempty. Suppose for some $v \in [t+1, u-1]$, $LAG(\tau, v+1) > LAG(\tau, v)$. Then, by Lemma 13, 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 critical subtask of V at slot v. Then, by Definition 3,

$$e(V_k) \leq v \leq d(V_k). \tag{33}$$

Because no subtask of V is scheduled at v, V_k is scheduled before v. (It cannot be scheduled later by PD^2 because there is a hole at v, by (U).) By (U), there is a hole at $v-1$; moreover, because $v \in [t+1, u-1]$, we have $v-1 \in [t, u-2] \subseteq [0, t_d]$. Hence, by Claim 2, we have $d(V_k) \leq v-1$, which contradicts (33). Therefore, $LAG(\tau, v+1) \leq LAG(\tau, v)$ for all $v \in [t+1, u-1]$. □

Thus, the LAG continues to decrease in the interval $[t+1, u-1]$. We now show that $LAG(\tau, u+1) \leq 0$, which establishes our proof obligation.

For any slot $v \in [t, u]$, let H_v denote the number of holes in slot v. Then, the number of tasks scheduled in slot v is $M - H_v$.

By Claim 3, only tasks in I and B may be active in slot $v \in [t+1, u-1]$. Therefore, by (14) and Claim 4,

$$\sum_{T \in \tau} flow(T, v) = \sum_{T \in I \cup B} flow(T, v) \leq \sum_{T \in I \cup B} S(T, v).$$

Therefore,

$$\text{for all } v \in [t+1, u-1], \ \sum_{T \in \tau} flow(T, v) \leq M - H_v. \tag{34}$$

Because $I \cup A \cup B = \tau$, by (9), $\sum_{T \in I \cup A \cup B} wt(T) \leq M$. Therefore, by (32) and (P1),

$$\sum_{T \in A} (flow(T,t) + flow(T,u)) + \sum_{T \in I \cup B} flow(T,u) \leq M. \tag{35}$$

Because the tasks in B are the ones scheduled in slot t, the number of tasks in set B is $M - H_t$. Hence, because the weight of each task is at most one,

$$\sum_{T \in B} flow(T,t) \leq \sum_{T \in B} wt(T) \leq M - H_t. \tag{36}$$

We are now ready to show that $LAG(\tau, u+1) \leq 0$. Because $S(T,v) = M - H_v$, by (14),

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

Because, by (U), there are no holes in slot u, $H_u = 0$. Therefore,

$$LAG(\tau, u+1) - LAG(\tau, t) = \sum_{v=t}^{u} \left(\sum_{T \in \tau} flow(T,v) \right) - \sum_{v=t}^{u-1} (M - H_v) - M. \tag{37}$$

The right-hand side of (37) can be rewritten as follows.

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

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

$$\sum_{T \in A} (flow(T,t) + flow(T,u)) + \sum_{T \in B} flow(T,t) + \sum_{T \in I \cup B} flow(T,u) + \sum_{v=t+1}^{u-1} \left(\sum_{T \in \tau} flow(T,v) \right) - \sum_{v=t}^{u-1} (M - H_v) - M.$$

Rearranging once again, we get

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

By (34), (35), and (36), we see that the above value is non-positive. Hence, by (37), $LAG(\tau, u+1) - LAG(\tau, t) \leq 0$. Because $LAG(\tau, t) \leq 0$, this implies that $LAG(\tau, u+1) \leq 0$.

By (U) and (31), $u \in [t+1, min(D(U_j), t_d)]$. Hence, $u+1 \in [t+2, t_d+1]$. Thus, there exists a $v \in [t+2, t_d+1]$ such that $LAG(\tau, v) \leq 0$. □