Mixed Pfair/ERfair Scheduling of Asynchronous Periodic Tasks ⋆

James H. Anderson and Anand Srinivasan

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

Abstract

Pfair scheduling was proposed by Baruah, Cohen, Plaxton, and Varvel as a non-work-conserving way of optimally and efficiently scheduling periodic tasks on a multiprocessor. In this paper, we introduce a work-conserving variant of Pfair scheduling called "early-release" fair (ERfair) scheduling. We also present a new scheduling algorithm called PD^2 and show that it is optimal for scheduling any mix of early-release and non-early-release asynchronous, periodic tasks. In contrast, almost all prior work on Pfair scheduling has been limited to synchronous systems. PD^2 is an optimization of an earlier deadline-based algorithm of Baruah, Gehrke, and Plaxton called PD; PD^2 uses a simpler tie-breaking scheme than PD to disambiguate equal deadlines. We present a series of counterexamples that suggest that, in general, the PD^2 tie-breaking mechanism cannot be simplified. In contrast to this, we show that no tie-breaking information is needed on two-processor systems.

Key words: Asynchronous periodic tasks, ERfair, fairness, multiprocessors, optimality, Pfair, real time, scheduling

1 Introduction

Pfair scheduling was proposed by Baruah, Cohen, Plaxton, and Varvel as a way of optimally and efficiently scheduling periodic tasks on a multiprocessor [7]. Pfair scheduling differs from more conventional real-time scheduling disciplines in that tasks are explicitly required to make progress at steady rates. In most real-time scheduling disciplines, the notion of a rate is implicit. For example,

⋆ Work supported by NSF grants CCR 9732916, CCR 9972211, CCR 9988327, and ITR 0082866. Some of the results in this paper were presented in preliminary form at the 12th and 13th Euromicro Conferences on Real-time Systems [1,3].

in the periodic task model, each task T executes at a rate given by $T.e/T.p$, where $T.e$ is the *execution cost* of each job (*i.e.*, invocation) of T, and $T.p$ is the *period* (and also the relative deadline) of T: every $T.p$ time units T releases a new job with execution cost $T.e$ and that job must complete execution before T's next job release. Although there *is* a notion of a rate here, it is actually a bit inexact: a job of T may be allocated $T.e$ time units at the beginning of its period, or at the end of its period, or its computation may be spread out more evenly. Under Pfair scheduling, this implicit notion of a rate is strengthened to require each task to be executed at a rate that is uniform across all jobs.

Pfair scheduling algorithms ensure uniform execution rates by breaking tasks into quantum-length "subtasks." Each subtask must execute within a "window" of time slots, the end of which is its deadline. These windows divide each period of a task into potentially overlapping subintervals of approximately equal length. Different subtasks of a task may execute on different processors (*i.e.*, migration is allowed). By breaking tasks into smaller executable units, Pfair scheduling algorithms circumvent many of the bin-packing-like problems that lie at the heart of intractability results involving multiple-resource real-time scheduling problems. Intuitively, it is easier to evenly distribute small, uniform items among the available bins than larger, non-uniform items.

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. (This is because non-work-conserving algorithms may choose to leave processors idle resulting in later finishing times.) In addition, non-work-conserving algorithms often entail higher runtime overheads because of extra bookkeeping required to keep track of when a job is and is not eligible.

In two separate papers [6,7], Baruah *et al.* presented two optimal Pfair scheduling algorithms, called PF and PD. In both algorithms, subtasks are prioritized by their deadlines. The two algorithms differ in the way in which ties are broken when two subtasks have the same deadline. In PF, ties are broken by comparing future subtask deadlines, which is somewhat expensive. In PD, ties are broken in constant time by inspecting four tie-break parameters. In both [6] and [7], only synchronous, periodic task systems are considered — in a *synchronous* task system, all tasks are constrained to release their initial jobs at time 0.

Contributions of this paper. In this paper, we extend the work of Baruah *et al.* in several ways.

- First, we introduce a work-conserving variant of Pfair scheduling called "early-release" fair (ERfair) scheduling. Under ERfair scheduling, if two subtasks are part of the same job, then the second subtask becomes eligible for execution as soon as the first completes. In other words, a subtask may be released "early," *i.e.*, before the beginning of its Pfair window.
- Second, we introduce a new scheduling algorithm derived from PD called PD^2 and show that it is optimal for scheduling any mix of early-release and non-early-release periodic tasks. PD^2 is obtained from PD by eliminating two of PD's tie-break parameters.
- Third, we present a series of counterexamples that strongly suggest that the PD^2 tie-breaking mechanism cannot be further simplified. In particular, we show that if either of the two PD^2 tie-breaks is eliminated, then there exists a feasible task set that is not correctly scheduled. We further show that several other "obvious" tie-breaking schemes do not work.
- Fourth, we show that for the important special case of a two-processor system, *no* tie-breaking information is required.
- Fifth, all of our results apply to asynchronous task systems. In contrast, almost all prior work on Pfair scheduling has focused only on synchronous systems [4,6,7,9]. One exception is a recent paper by Moir and Ramamurthy in which a static-priority Pfair scheduling algorithm for asynchronous task systems is given [10]. Such static-priority algorithms are not optimal. In other recent work, we proposed a task model called the *intra-sporadic* model that generalizes the asynchronous model [2]. However, the algorithms given by us for that model are applicable only to two-processor systems. (In the time since the research in this paper was conducted, we have extended work on Pfair scheduling in several ways. In particular, we have obtained bounds on the amount by which a deadline is missed if no tie-breaking information is used [12]. We have also shown that PD^2 can be used to optimally schedule intra-sporadic tasks on multiprocessors [11]. The swapping proof technique used in this paper reveals many fundamental properties of Pfair- and ERfair-scheduled systems that were essential in showing the optimality of PD^2 for intra-sporadic tasks.)

A final, more subtle, contribution of this paper is our proof of correctness for PD^2. In [7], PD is proved correct by means of a simulation argument that shows that PD "closely" tracks the behavior of PF. This proof is quite "brittle" and is difficult to extend beyond the synchronous, periodic model [5]. In contrast, we prove that PD^2 is correct by means of an inductive "swapping" argument in which an arbitrary schedule is converted into one in accordance with the PD^2 priority definition by systematically interchanging pairs of subtasks. Thus, we have succeeded in showing that swapping arguments, which have proven quite useful in work on more "conventional" real-time scheduling disciplines, can be used to effectively reason about PD and related algorithms as well.

Some example schedules. Before continuing, we consider some example schedules that illustrate the scheduling schemes considered in this paper. In the Pfair scheduling literature, the ratio of a task T's execution cost and period, $T.e/T.p$, 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 three sets of tasks executing on two processors: a set A of one task of weight 5/16, a set B of three tasks of weight 4/16, and a set C of 15 tasks of weight 1/16. Inset (a) shows the schedule for these tasks up to time slot 15 in a Pfair-scheduled system. At time 0, the scheduler selects for execution the first subtask of the set-A task and the first subtask of one of the set-B tasks. At time 1, the first subtasks of the remaining two set-B tasks are scheduled. The rest of the schedule is similarly produced. Note that successive windows of a task are either disjoint or overlap by one slot. As explained later, this is a general property of Pfair-scheduled systems. Inset (b) shows the schedule for the same task set under ERfair scheduling. In this case, each subtask is eligible as soon as its predecessor completes. Notice that all set-A and set-B jobs have finished by time slot 8, which is much sooner than in the Pfair schedule. Inset (c) shows a schedule in which only the set-A task is an early-release task. Note that this task's response time is lower here than in either of the previous two examples. (These example schedules are used only to illustrate the basic ideas behind Pfair and ERfair scheduling; details as to how to obtain subtask windows and select subtasks for execution under PD^2 are given in later sections.)

The rest of this paper is organized as follows. In Section 2, we define Pfair and ERfair scheduling. Then, in Section 3, we present the PD^2 algorithm. In Section 4, we present the counterexamples mentioned above. In Section 5, we prove that PD^2 correctly schedules any feasible asynchronous, periodic task system consisting of a mix of early-release and non-early-release tasks. In Section 6, we show that both PD^2 tie breaks can be eliminated in two-processor systems. Concluding remarks appear in Section 7. A number of properties pertaining to Pfair windows are proved in Appendix A and the feasibility condition for asynchronous task systems is proved in Appendix B.

2 Pfair and ERfair Scheduling

Consider a collection of synchronous, periodic real-time tasks to be executed on a system of multiple processors. (For the moment, we are only considering synchronous, periodic tasks. Asynchronous tasks will be considered at the end of this section.) We assume that processor time in such a system is allocated

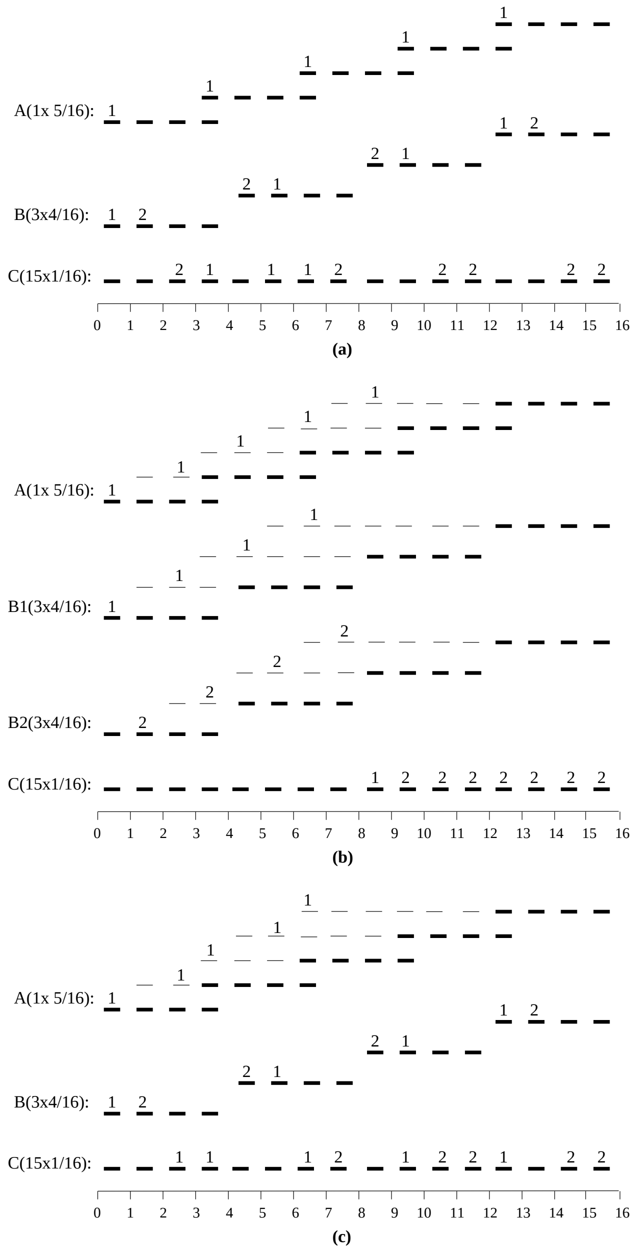

Fig. 1. A schedule for a synchronous, periodic task set under **(a)** Pfair scheduling, **(b)** ERfair scheduling, and **(c)** mixed Pfair and ERfair scheduling. With one exception (noted below), tasks of a given weight are shown together. Each column corresponds to a time slot. For each subtask, there is an interval of time slots during which it is eligible (denoted by dashes); this interval includes its Pfair window (denoted by bold dashes). An integer value n in slot t means that n of the tasks in the corresponding set have a subtask scheduled at t. In inset (b), set B has been split into B1 and B2; the set-B1 task happens to always be scheduled earlier than the set-B2 tasks, so its subsequent subtasks become eligible earlier.

in discrete time units, or 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.) Associated with each task T is a *period* $T.p$ and an *execution cost* $T.e$.

Every $T.p$ time units, a new invocation of T with a cost of $T.e$ time units is released into the system; we call such an invocation a *job* of T. Each job of a task must complete execution before the next job of that task begins. Thus, $T.e$ time units must be allocated to T in each interval $[(k-1) \cdot T.p, k \cdot T.p)$, where $k \geq 1$. T may be allocated time on different processors in such an interval, as long as it is not allocated time on different processors at the same time.

The sequence of allocation decisions over time defines a "schedule." Formally, a *schedule* S is a mapping $S : \tau \times \mathcal{N} \mapsto \{0, 1\}$, where τ is a set of periodic tasks and $\mathcal{N}$ is the set of nonnegative integers. If $S(T, t) = 1$, then we say that *task* T *is scheduled at slot* t. S_t denotes the set of tasks scheduled in slot t. The statements $T \in S_t$ and $S(T, t) = 1$ are equivalent.

Lag constraints. The ratio $T.e/T.p$ is called the *weight* of task T, denoted $wt(T)$. We assume each task's weight is strictly less than one — a task with weight one would require a dedicated processor, and thus is quite easily 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.

A task's weight defines the rate at which it is to be scheduled. Because processor time is allocated in quanta, we cannot guarantee that a task T will execute for *exactly* $(T.e/T.p)t$ time during each interval of length t. Instead, in a Pfair-scheduled system, processor time is allocated to each task T in a manner that ensures that its rate of execution never deviates too much from that given by its weight $T.e/T.p$. More precisely, correctness is defined by focusing on the *lag* between the amount of time allocated to each task and the amount of time that would be allocated to that task in an ideal system with a quantum approaching zero. Formally, the *lag of task* T *at time* t, 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 algorithm. Hence, we leave the schedule implicit and use $lag(T, t)$ instead of $lag(T, t, S)$.) A schedule is *Pfair* if and only if

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

Informally, the allocation error for each task is always less than one quantum. Our notion of early-release scheduling is obtained by simply dropping the -1

lag constraint. Formally, a schedule is *early-release fair* (*ERfair*) if and only if

$$(\forall T, t :: lag(T, t) < 1). \tag{3}$$

In a *mixed* Pfair/ERfair-scheduled task system, each task's lag is subject to *either* (2) *or* (3); such a task is called a *non-early-release* task in the former case, and an *early-release* task in the latter. Note that any Pfair schedule is ERfair, but not necessarily vice versa.

It is straightforward to show that in any ERfair schedule (and hence any Pfair or mixed Pfair/ERfair schedule), all job deadlines are met. In particular, in an ERfair schedule, $lag(T, t) = 0$ for $t = 0, T.p, 2T.p, 3T.p, \ldots$. This is because, for these values of t, $(T.e/T.p)t$ is an integer, and therefore by (1), $lag(T, t)$ is an integer as well. By (3), if $lag(T, t)$ is an integer, then it must be 0 or some negative integer. However, it cannot be a negative integer because this would imply that more processor time has been allocated to T than has been requested by jobs of T up to time t. Hence, $lag(T, t)$ is 0 for these values of t.

Feasibility. A synchronous, periodic task set τ has a Pfair schedule on M processors if and only if

$$\sum_{T \in \tau} \frac{T.e}{T.p} \leq M. \tag{4}$$

This result was proved by Baruah *et al.* [6] by means of a network flow construction. (We use a similar proof technique in Appendix B to obtain a corresponding feasibility condition for asynchronous periodic task systems.) Because every Pfair schedule is also an ERfair schedule, (4) is a feasibility condition for ERfair-scheduled systems as well. For similar reasons, it is also a feasibility condition for mixed Pfair/ERfair-scheduled task systems.

Windows. The Pfair lag bounds given in (2) have the effect of breaking each task T into an infinite sequence of unit-time *subtasks*. We denote the i^{th} subtask of task T as T_i, where $i \geq 1$. Subtask T_{i+1} is called the *successor* of T_i and T_{i-1} is called the *predecessor* of T_i (defined for $i > 1$). As in [6], we associate with each subtask T_i a *pseudo-release* $r(T_i)$ and a *pseudo-deadline* $d(T_i)$. If T_i is synchronous and periodic, then as shown in Appendix A, $r(T_i)$ and $d(T_i)$ are as follows.

$$r(T_i) = \left\lfloor \frac{i-1}{wt(T)} \right\rfloor \tag{5}$$

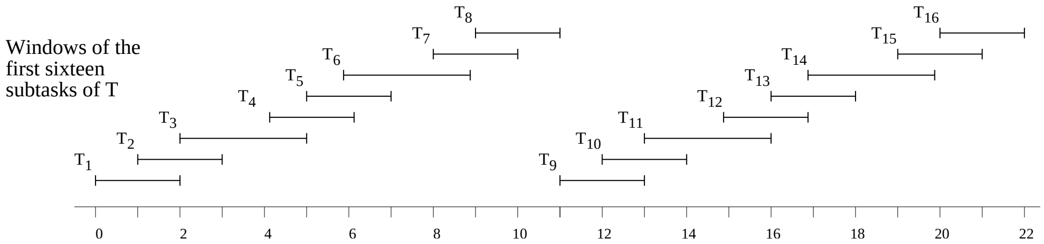

Fig. 2. The Pfair windows of the first two jobs (or sixteen subtasks) of a task T with weight 8/11 in a Pfair-scheduled system. During each job of T, each of the eight units of computation must be allocated processor time during its window, or else a lag-bound violation will result.

$$d(T_i) = \left\lceil \frac{i}{wt(T)} \right\rceil \tag{6}$$

(In our earlier work [1–3], pseudo-deadlines were defined to refer to slots; here, they refer to time. Hence, the formula for $d(T_i)$ given here is slightly different.)

$r(T_i)$ is the first slot into which T_i potentially could be scheduled, and $d(T_i) - 1$ is the last such slot. 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 $d(T_i) - r(T_i)$. A window of length n is called an *n-window*.

As an example, consider a task T with weight $wt(T) = 8/11$. Each job of this task consists of eight windows, one for each of its unit-time subtasks. Using Equations (5) and (6), it is possible to show that the windows within each job of T are as depicted in Figure 2.

Dropping the -1 lag constraint, as done in (3), is equivalent to allowing subtasks to execute before their Pfair windows. In a Pfair-scheduled system, a subtask T_i is eligible at time t if $t \in w(T_i)$ and if T_{i-1} has been scheduled prior to t but T_i has not. In an ERfair-scheduled system, if T_{i-1} and T_i are part of the same job, then T_i becomes eligible for execution immediately after T_{i-1} executes, which may be before T_i's Pfair window. (Obviously, no subtask can become eligible before the beginning of the job that contains it. Thus, the first subtask of a job cannot be released early.) This difference is illustrated in Figure 3. We will continue to use the term "release" to refer the first slot of a subtask's Pfair window, even though a subtask that is early-released can actually execute prior to this slot.

Asynchronous Pfair task systems. In the usual definition of an asynchronous periodic task system, each task may release its first job at any time. For Pfair- or ERfair-scheduled systems, this is equivalent to allowing the first

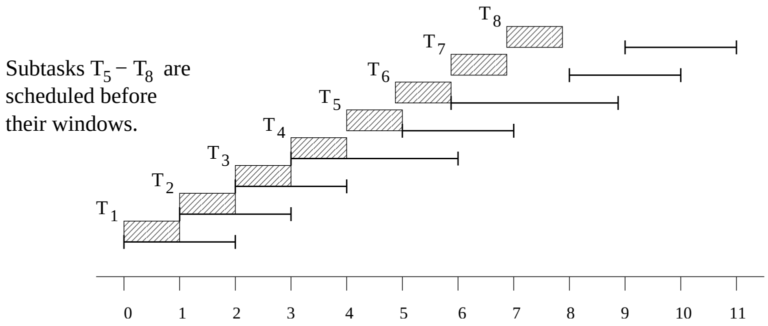

Fig. 3. The Pfair windows of the first jobs of a task T with weight $8/11$ are shown. The schedule shown is ERfair, but not Pfair.

subtask of each task T, namely T_1, to be released any time at or after time zero. Our notion of an asynchronous task system generalizes this: we allow a task T to begin execution with any of its subtasks, perhaps one other than T_1, and this subtask may be released any time at or after time zero. As we shall see, this added generality facilitates our correctness proof for PD^2. It is straightforward to modify the flow construction used in the feasibility proof of (4) to apply to asynchronous task systems as defined here. (Refer to Appendix B.) Thus, (4) is a feasibility condition for such systems. (In fact, the network flow construction produces a Pfair schedule, *i.e.*, each task is scheduled within its window.) In addition, it is possible to define the release and deadline of each subtask using simple formulae that are similar to those given in (5) and (6) for synchronous task systems. In particular, suppose task T releases its first subtask at time r and let T_i $(i \geq 1)$ be this subtask. Then, the release and deadline of any subtask T_j $(j \geq i)$ are given by the following formulae, where $\Delta(T) = r - \left\lfloor \frac{i-1}{wt(T)} \right\rfloor$.

$$r(T_j) = \left\lfloor \frac{j-1}{wt(T)} \right\rfloor + \Delta(T) \tag{7}$$

$$d(T_j) = \left\lceil \frac{j}{wt(T)} \right\rceil + \Delta(T) \tag{8}$$

Thus, the windows of T are shifted by an *offset* given by $\Delta(T)$, which determines the release time of its first subtask. Note that if T is an asynchronous periodic task according to the usual definition, then the first subtask released by T is T_1 and $\Delta(T)$ reduces to r.

3 The PD^2 Algorithm

For synchronous, periodic task systems, the most efficient Pfair scheduling algorithm previously proposed is an algorithm called PD [7]. PD prioritizes subtasks by pseudo-deadline (hence its name). It is related to an earlier algorithm called PF [6] in which ties among subtasks with the same deadline are broken by comparing vectors of future pseudo-deadlines. Although PF was originally presented only in the context of synchronous, periodic systems, its correctness proof is also applicable to asynchronous systems. Unfortunately, the runtime costs associated with PF are prohibitive, so it is not a practical algorithm.

In this paper, we propose a new algorithm called the PD^2 algorithm, which is an improvement over the PD algorithm. In both PD and PD^2, the pseudo-deadline vectors of PF are replaced by a constant number of tie-break parameters. Four tie-break parameters are used in PD, while in PD^2, only two tie-breaks are used. We now define the two PD^2 tie-break parameters. The rationale behind each tie-break is explained later when considering the PD^2 priority definition.

First tie-break: The successor bit. By (7) and (8), $r(T_{i+1})$ is either $d(T_i)$ or $d(T_i)$-1, *i.e.*, successive windows are either disjoint or overlap by one slot. We define a bit $b(T_i)$ that distinguishes between these two possibilities.

$$b(T_i) = \begin{cases} 1, \text{ if } r(T_{i+1}) = d(T_i) - 1 \\ 0, \text{ if } r(T_{i+1}) = d(T_i). \end{cases} \tag{9}$$

For example, in Figure 2, $b(T_i) = 1$ for $1 \leq i \leq 7$ and $b(T_8) = 0$. If $T.e$ divides i, then $(i \cdot T.p)/T.e$ is an integer, *i.e.*, $\left\lceil \frac{i}{wt(T)} \right\rceil = \left\lfloor \frac{i}{wt(T)} \right\rfloor$. Thus, by (7)–(9), we have the following property.

(B) If T_i is the last subtask of a job, then $b(T_i)$ is zero.

Second tie break: The group deadline. Consider a sequence $T_i, \ldots, T_j$ of subtasks of a heavy task T such that $|w(T_k)| = 2 \ \wedge \ b(T_k) = 1$ for all $i < k \leq j$ and either $|w(T_{j+1})| = 3$ or $w(T_{j+1}) = 2 \ \wedge \ b(T_{j+1}) = 0$ (*e.g.*, T_1, T_2 or T_3, T_4, T_5 or T_6, T_7 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 $d(T_j)+1$ to be the *group deadline* for the group of subtasks $T_i, \ldots, T_j$. 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 4, 8, 11, 15, 19, and 22.

We let $D(T_i)$ denote the group deadline of subtask T_i. Formally, if T is heavy, then

$$D(T_i) = (\mathbf{min}\ u :: u \geq d(T_i) \text{ and } u \text{ is a group deadline of } T).$$

For example, in Figure 2, $D(T_1) = 4$ and $D(T_6) = 11$. The above definition of D is valid only for heavy tasks. If T is light, then $D(T_i) = 0$.

The PD^2 Priority Definition. We can now state the PD^2 priority definition. Under PD^2, subtask T_i's priority is at least that of subtask U_j, denoted $T_i \preceq U_j$, if one of the following rules is satisfied.

(i) $d(T_i) < d(U_j)$.
(ii) $d(T_i) = d(U_j)$ and $b(T_i) > b(U_j)$.
(iii) $d(T_i) = d(U_j)$, $b(T_i) = b(U_j) = 1$, and $D(T_i) \geq D(U_j)$. $\square$

Any ties not resolved by these three rules can be broken arbitrarily. Thus, 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 scheduling a subtask with a b-bit of one earlier places fewer constraints on the future schedule. (In particular, if T_i were scheduled *very* late, *i.e.*, in the last slot of its window, then this would reduce the number of slots available to T_{i+1} by one.) If T_i and U_j have equal pseudo-deadlines and b-bits of one, 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. 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. If both are heavy and their group deadlines are equal, then the tie can be broken arbitrarily. (Recall that $D(T_i) = 0$ if T is light. Thus, in a light-only task system, PD^2 uses only the first two rules to determine subtask priorities.)

The priority definition used in PD is similar to ours, except that two additional tie-break parameters are used. The first of these is a task's weight. The second is a bit associated with each group deadline that is similar to the b-bit we associate with pseudo-deadlines. Our results show that these two additional tie-break parameters are not necessary.

All of the components within the PD^2 priority definition can be calculated in an asynchronous system using simple formulae. In particular, for any subtask T_i, $d(T_i)$ is given by (8), and $b(T_i)$ by (9). Also, as shown in [7], $D(T_i)$ can be calculated by using (8) to determine the deadlines of a task of weight $1 - wt(T)$.

(The deadlines of such a task coincide with the group deadlines of a task of weight $wt(T)$.) If a task T is subject to the Pfair lag constraint (2), then each subtask T_i becomes eligible for execution at time $r(T_i)$, unless its predecessor is scheduled there, in which case it becomes eligible at time $r(T_i) + 1$. If T is instead subject to the ERfair lag constraint (3), then T_i becomes eligible at time $r(T_i)$ if it is the first subtask of its job, and immediately after the execution of T_{i-1} otherwise.

Given the priority definition above, the PD^2 algorithm is simple to explain (in fact, the PD^2 algorithm is nearly identical to the algorithm given for PD in [7], except that a different priority definition is used). A priority-sorted "ready queue" is used to store eligible subtasks. In addition, there are a number of priority-ordered "release queues" associated with future time slots. At the beginning of each time slot, the M highest-priority subtasks in the ready queue (if that many subtasks are eligible) are selected for execution, where M is the number of processors in the system. If T_i is one of the selected subtasks, then T_{i+1} is inserted into the release queue associated with time t, where t is the time at which T_{i+1} becomes eligible. At the beginning of each time slot, the release queue for that slot is merged with the ready queue. An additional search structure is used in order to efficiently access the release queues (see [7] for details). Note that if T is an early-release task and T_i and T_{i+1} are part of the same job, then the algorithm can be optimized to insert T_{i+1} directly into the ready queue (which will then be used in the next time slot) instead of the release queue for the next slot (where T_{i+1} becomes eligible).

4 Minimality of the PD^2 Priority Definition

Before proving the optimality of PD^2, we consider other scheduling algorithms that determine subtask priorities using fewer or more-efficient tie-breaking rules.

According to the PD^2 priority definition, each task T is effectively prioritized at time t by the triple $(d(T_i), b(T_i), D(T_i))$, where T_i is the subtask of T eligible at time t. In this section, we present a collection of counterexamples that show that this priority definition cannot be substantially simplified.

In each proof in this section, an example task system is considered that fully utilizes a system of M processors for some M. Each such task system consists of a set A of tasks of one weight and a set B of tasks of another weight. We show that if this task system is scheduled with the newly-proposed priority definition, then a time slot is reached at which fewer than M tasks are scheduled. Since the task system fully utilizes the M processors, this implies that a deadline is missed at some future time. In the proof of Theorem 1, we

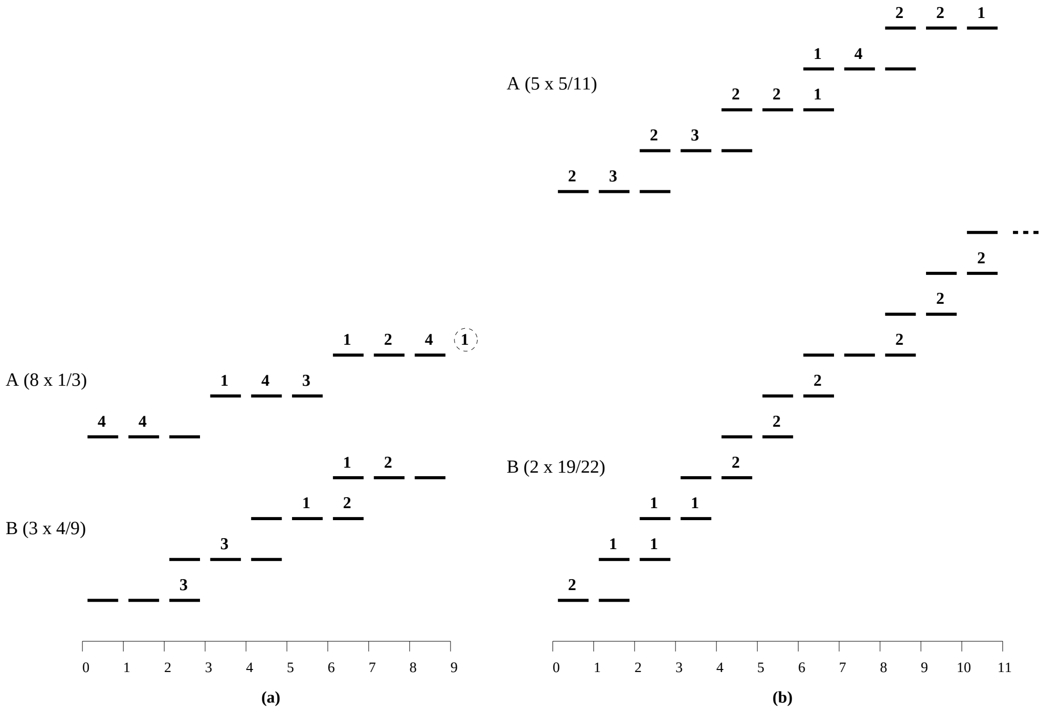

Fig. 4. The notation used here (and in Figures 5, 6, and 8) is the same as in Figure 1. **(a)** Theorem 1. A deadline is missed at time 9 by a task of weight 1/3. (We do not illustrate deadline misses in the subsequent figures, and show the schedule only until a slot is reached in which fewer than M tasks are scheduled.) **(b)** Theorem 2.

explain the resulting schedule in detail. The subsequent proofs in this section are sketched more briefly. We begin by considering the b-bit.

Theorem 1 *If the* PD^2 *priority definition is changed by eliminating* b (*i.e., Rule* (ii)), *then there exists a feasible task system that is not correctly scheduled.*

Proof. Consider a task system consisting of a set A of eight light tasks with weight 1/3 and a set B of three light tasks with weight 4/9. Because total utilization is four, Expression (4) implies that the system is feasible on four processors. Consider the schedule shown in Figure 4(a).

As seen in Figure 4(a), each job of a task with weight 1/3 consists of one three-slot window. Each job of a task with weight 4/9 consists of four three-slot windows, with consecutive windows overlapping by one slot. The first subtask of each task has a pseudo-deadline at slot 2. Because b has been eliminated, this tie can be broken arbitrarily. We break it in favor of the subtasks of the tasks in set A. Therefore the eight tasks in set A are scheduled in slots 0 and 1. In slot 2, the three tasks from set B are the only tasks with subtasks that are eligible for execution. Hence, only three subtasks can be scheduled in slot 2, causing a deadline miss later at time 9. $\square$

The definition of $D(T_i)$ ensures that if T is light and U is heavy and if $d(T_i) = d(U_j) \wedge b(T_i) = b(U_j) = 1$, then U_j has higher priority. The following theorem shows that it is necessary to tie-break such a situation in favor of the heavy task.

Theorem 2 *Suppose the definition of D is changed as follows: if T is light, then $D(T_i)$ is a randomly-selected value. Then, there exists a feasible task system that is not correctly scheduled.*

Proof. Consider a task system consisting of a set A of five light tasks with weight $5/11$ and a set B of two heavy tasks with weight $19/22$. Because total utilization is four, this task system is feasible on four processors. Consider the schedule shown in Figure 4(b), which is possible given the proposed priority definition. In particular, at times 1, 3, and 7, the set-A tasks are favored over the set-B tasks. This causes only three subtasks to be eligible for execution in slot 10. $\Box$

The previous counterexamples give rise to the possibility that $D(T_i)$ is actually only needed to tie-break heavy tasks over light tasks. The next theorem shows that this is not the case.

Theorem 3 *Suppose the definition of D is changed as follows: if T is heavy, then $D(T_i)$ is one. (If T is light, then $D(T_i)$ is zero as before.) Then, there exists a feasible task system that is not correctly scheduled.*

Proof. Consider a task system, to be scheduled on four processors, consisting of a set A of three heavy tasks with weight $5/7$ and a set B of two heavy tasks with weight $13/14$. The proposed priority definition allows the schedule shown in Figure 5. Note that only three subtasks are eligible in slot 3. $\Box$

Given the previous counterexample, one may wonder if the definition of D can be weakened so that ties among heavy tasks are statically resolved. The following theorem shows that this is unlikely.

Theorem 4 *If D is changed so that ties among heavy tasks are statically broken by weight, then there exists a feasible task system that is not correctly scheduled.*

Proof. Consider a task system, to be scheduled on 12 processors, consisting of a set A of three heavy tasks with weight $8/9$ and a set B of ten heavy tasks with weight $14/15$. First, suppose that D is defined to statically tie-break the set-A tasks over the set-B tasks. Then, the schedule shown in Figure 6(a) is possible. In this schedule, only 11 subtasks are eligible at time slot 8. Second, suppose that D is defined to statically tie-break the set-B tasks over the set-A tasks. In this case, the schedule shown in Figure 6(b) is possible. In this schedule, only 11 subtasks are eligible at time slot 14. $\Box$

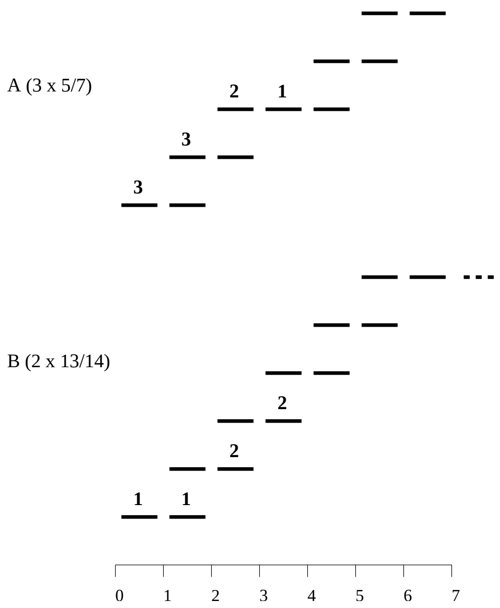

Fig. 5. Theorem 3 and Theorem 6.

From the previous theorem, it follows that D *almost* certainly must be defined to dynamically tie-break heavy tasks. (Note, for example, that Theorem 4 leaves open the possibility of statically defining D so that some set-A tasks are favored over set-B tasks, but other set-A tasks are not favored over set-B tasks.) One obvious approach to try that is less dynamic than ours is to define $D(T_i)$ based on the deadline of the current *job* of T. The next two theorems show that using job deadlines does not work; in the first of these theorems, later job deadlines are given higher priority, and in the second, nearer job deadlines are given higher priority.

Theorem 5 *Suppose the definition of D is changed as follows: if T is heavy, then $D(T_i)$ is the deadline of the current job of T. Then, there exists a feasible task system that is not correctly scheduled.*

Proof. Consider a task system, to be scheduled on 17 processors, consisting of a set A of nine heavy tasks with weight $7/9$ and a set B of 12 heavy tasks with weight $5/6$. The proposed priority definition allows the schedule shown in Figure 7. (Note that the newly-proposed definition of D favors set-A tasks over set-B tasks.) In this schedule, only 16 subtasks are eligible at time slot 4. □

Theorem 6 *Suppose the definition of D is changed as follows: if T is heavy, then $D(T_i)$ is $1/t$, where t is the deadline of the current job of T. Then, there exists a feasible task system that is not correctly scheduled.*

Proof. This can be proved by using the task system and schedule shown in Figure 5, which was used previously in the proof of Theorem 3. (Note that the newly-proposed definition of D favors set-A tasks.) □

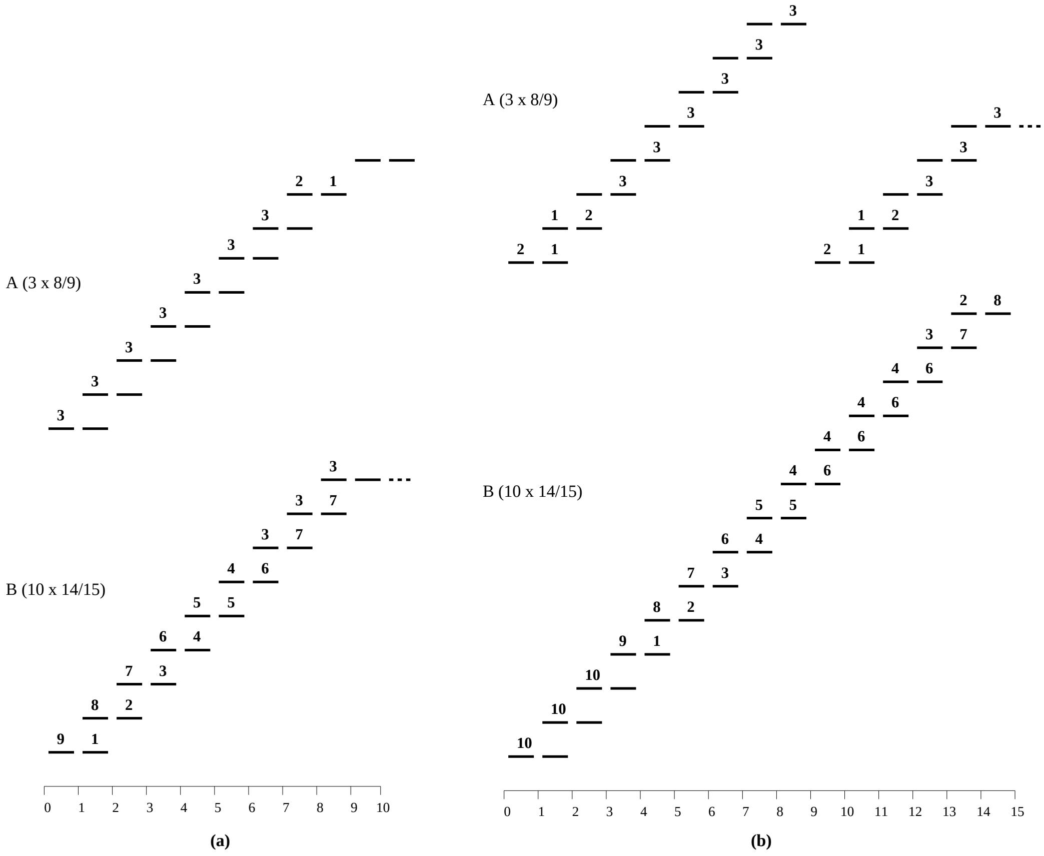

Fig. 6. Theorem 4.

As we will show later in Section 6, neither b nor D is needed on two processors. We now show that at least one tie-breaking rule is needed in any system with three or more processors. (Note that Theorems 1, 2, 3 and 6 apply on systems with four or more processors.)

Theorem 7 *If our priority definition is changed by eliminating both b and D, then there exists a task system that is feasible on three processors that is not correctly scheduled.*

Proof. Consider a task system, to be scheduled on three processors, consisting of a set A of three heavy tasks with weight $1/2$ and a set B of two heavy tasks with weight $3/4$. The proposed priority definition allows the schedule shown in Figure 8(a). In this schedule, only two subtasks are eligible at time slot 1. (Note that *either b or D* would correctly tie-break these tasks.) □

One "obvious" potential priority definition that comes to mind is to use the rational value $\frac{i}{wt(T)}$ as the deadline of subtask T_i, which is tantamount to omitting the ceiling brackets in the deadline formula (6). As it turns out, this new definition works for light-only task systems, but does not work if heavy tasks are present.

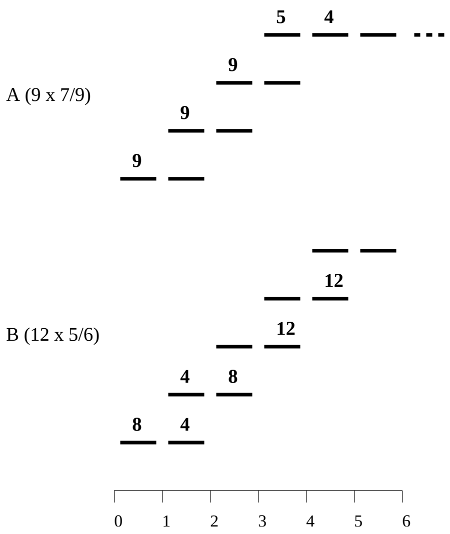

Fig. 7. Theorem 5.

The optimality of this priority definition for light-only task systems follows from the optimality of PD^2. Recall that PD^2 only uses Rules (i) and (ii) for light tasks, *i.e.*, it prioritizes light tasks using the pair $(d(T_i), b(T_i))$. If $\frac{i}{wt(T)} \leq \frac{j}{wt(U)}$, then $\left\lceil \frac{i}{wt(T)} \right\rceil \leq \left\lceil \frac{j}{wt(U)} \right\rceil$, *i.e.*, $d(T_i) \leq d(U_j)$. Further, if $\frac{i}{wt(U)}$ is an integer, then $b(T_i) = 0$; in this case either $\frac{j}{wt(U)} > \frac{i}{wt(U)}$, in which case $d(U_j) > d(T_i)$ or $\frac{j}{wt(U)} = \frac{i}{wt(U)}$, in which case $b(U_j) = 0$. Thus, all the scheduling decisions are in accordance with PD^2.

It might appear that the expression $\frac{i}{wt(T)}$ reduces a task's priority to a single number. However, to avoid rounding errors, this rational number must be stored as two integers. PD^2 actually improves upon this by using an integer for $d(T_i)$ and a bit for $b(T_i)$. (In fact, the PD^2 priority definition for light tasks can be reduced to a single integer $d'(T_i)$ that equals $2 \cdot d(T_i) - b(T_i)$. Under this new priority definition, T_i's priority is at least U_j's if $d'(T_i) \leq d'(U_j)$. It is straightforward to show that this algorithm makes the same scheduling decisions as PD^2 because the b-bit is either zero or one.)

We now show that the using rational deadlines can sometimes lead to missed deadlines in systems with heavy tasks.

Theorem 8 *If the priority definition is changed so that T_i's priority is at least U_j's if $\frac{i}{wt(T)} \leq \frac{j}{wt(U)}$, and any ties are broken arbitrarily, then there exists a feasible task system such that it is not correctly scheduled.*

Proof. Consider a task system consisting of a set A of 15 tasks with weight $3/5$ and a set B of ten tasks with weight $9/10$. Total utilization is 18, so we should be able to schedule this task system on 18 processors. Consider the schedule shown in Figure 8(b), which is possible given the proposed priority

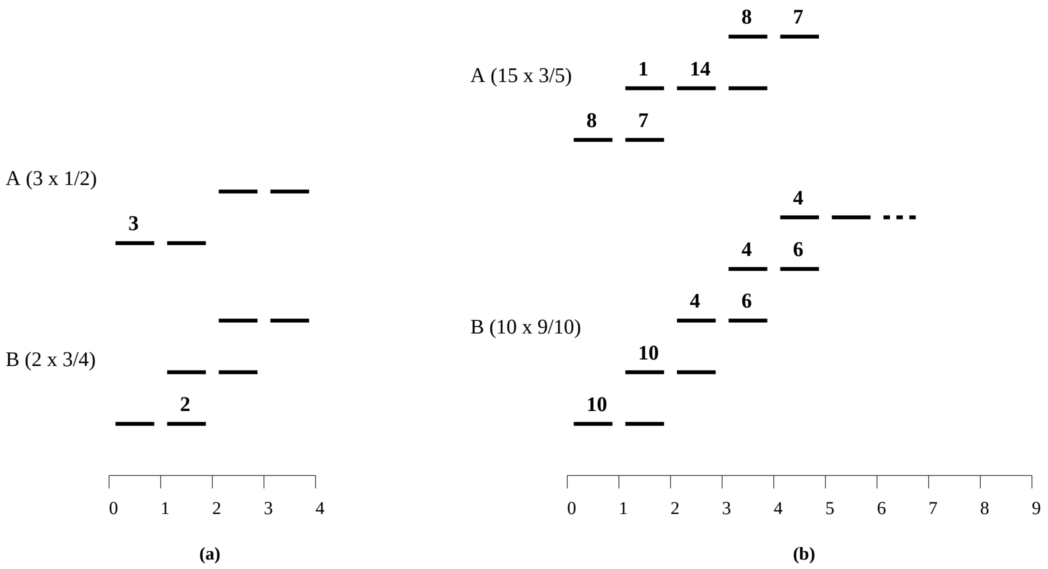

Fig. 8. **(a)** Theorem 7. **(b)** Theorem 8.

definition. In particular, at time 2, the priority of any task in A is given by $\frac{2}{3/5} = \frac{10}{3}$, while the priority of a task in B is given by $\frac{3}{9/10} = \frac{10}{3}$. Hence, tasks in set A may be favored, as shown in Figure 8(b). This causes only 17 subtasks to be eligible for execution in slot 4. $\Box$

5 Optimality Proof of PD^2

In this section, we show that PD^2 is optimal for scheduling the class of task systems considered in this paper. We begin in Section 5.1 by stating several properties that are used extensively in the proof. The optimality proof itself is then given in Section 5.2.

Note that by (5) and (6), the Pfair windows of two tasks T and U for which $\frac{T.e}{T.p} = \frac{U.e}{U.p}$ are identical. This implies that, under Pfair scheduling, they will be scheduled in precisely the same way. Thus, for notational simplicity, it is reasonable to assume that $T.e$ and $T.p$ are relatively prime for each task T, *i.e.*, $gcd(T.e, T.p) = 1$, where $gcd(a, b)$ is the greatest common divisor (GCD) of a and b. *We do make this assumption in our proof.* Unfortunately, this creates a slight problem, because under ERfair scheduling, two tasks with equal weights but different periods may be scheduled differently. In particular, they may differ with regard to which subtasks may be released early because their job releases occur at different times. However, the above assumption is still valid because our proof applies even if subtasks are early-released across jobs. (In fact, our proof applies even if the scheduler *dynamically* decides whether to early release subtasks or not, or bounds early releases by a threshold — *e.g.*, a subtask may be allowed to release early, but only up to two time slots before its Pfair window.)

5.1 Properties About Subtask Windows

We now state several properties about subtask windows and group deadlines. These properties are proved in Appendix A. Each property pertains to just a single task. For brevity, we let T denote this task, and abbreviate $T.e$ and $T.p$ as e and p, respectively.

Lemma 9 *The following properties hold for any task T.*

(a) *$r(T_{i+1})$ is either $d(T_i)$ or $d(T_i)-1$, which implies that $r(T_{i+1}) \geq d(T_i)-1$.*
(b) *The sequence of windows within any two jobs are identical,* i.e., $|w(T_{ke+i})| = |w(T_i)|$, *where $1 \leq i \leq e$ and $k \geq 0$.*
(c) *The windows are symmetric within each job,* i.e., $|w(T_{ke+i})| = |w(T_{ke+e+1-i})|$, *where $1 \leq i \leq e$, and $k \geq 0$.*
(d) *The length of each window is either $\left\lceil \frac{p}{e} \right\rceil$ or $\left\lceil \frac{p}{e} \right\rceil + 1$.*
(e) $|w(T_i)| = \left\lceil \frac{p}{e} \right\rceil$ *if $(i-1)$ is a multiple of e.*

Property (P2) below refers to a "minimal" window of a task. Note that by part (d) of Lemma 9, the windows of any task are of at most two different lengths. We refer to a window of task T with length $\left\lceil \frac{T.p}{T.e} \right\rceil$ as a *minimal* window of T.

(P1) If $b(T_i) = 0$, then $|w(T_i)| = |w(T_{i+1})|$.

(P2) If $b(T_i) = 0$, then $w(T_i)$ is a minimal window of T.

(P3) For all i and j, $|w(T_j)| \leq |w(T_i)| + 1$.

(P4) For all i and j, $|w(T_j)| \geq |w(T_i)| - 1$.

(P5) If T is light, then all of its windows are of length at least three.

(P6) T has a 2-window if and only if it is heavy.

(P7) If T is heavy, then all its windows are of length two or three.

(P8) If T is heavy and $b(T_i) = 0$, then $|w(T_i)| = 2$.

(P9) If t and t' are successive group deadlines of a heavy task T, then $t' - t$ is either $\left\lceil \frac{1}{1-wt(T)} \right\rceil$ or $\left\lceil \frac{1}{1-wt(T)} \right\rceil - 1$.

(P10) Let T be a heavy task. Let t and t' be consecutive group deadlines of T, where t is the last group deadline within some job of T (for the first job of T, take t to be 0). Then $t' - t$ is at least the difference between any pair of consecutive group deadlines of T.

We now show that PD^2 produces a "valid" schedule for any feasible asynchronous task system. A schedule is *valid at time slot* t if **(i)** for each subtask T_i scheduled in slot t, t lies within the interval during which T_i is eligible (which implies that T_i meets its deadline), **(ii)** no two subtasks of the same task are scheduled at t, and **(iii)** the number of tasks scheduled at t is at most the number of processors. A schedule is *valid* if it is valid at every time slot. We begin by assuming, to the contrary, that PD^2 fails to correctly schedule some task system. Then, there exists a time t_d as follows.

Definition 10 *t_d is the earliest time at which some feasible asynchronous task system misses a deadline under* PD^2.

In other words, PD^2 does not miss any deadline before time t_d for any feasible asynchronous task system. Let τ be a feasible asynchronous 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 a larger number of subtasks in $[0, t_d)$ than τ.

In the remainder of this section, we assume that τ is as defined here. The existence of such a τ follows from our assumption that PD^2 is not optimal. We now show that PD^2 produces a valid schedule for τ over $[0, t_d]$, thus contradicting our starting assumption.

In the proofs that follow, we consider 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 slot t in S.

The following lemma gives an important property of the task set τ. (Note that the proof of this lemma relies on (T2) and the fact that we have generalized the notion of an asynchronous system to allow a task to begin execution with any of its subtasks.)

Lemma 11 *If task $T \in \tau$ releases its first subtask at time $t > 0$, and if this first subtask is T_i, $i > 1$, then either $b(T_{i-1}) = 0$ and $|w(T_{i-1})| > t$ or $b(T_{i-1}) = 1$ and $|w(T_{i-1})| > t + 1$.*

Proof. We only consider the case when $b(T_{i-1}) = 0$ holds; in this case, we show that $|w(T_{i-1})| > t$ holds. (The proof for the case when $b(T_{i-1}) = 1$ holds is similar.) Suppose, to the contrary, that $|w(T_{i-1})| \leq t$. Consider the task system τ' obtained by adding the subtask T_{i-1} with a release at time $t - |w(T_{i-1})| \geq 0$. (We assume that the relative priorities of two subtasks in τ

do not change in τ'.) Then, τ' satisfies the following properties.

- It has one more subtask than τ.
- It misses a deadline at t_d.

To see the latter, note that upon adding T_{i-1} to τ, if T_{i-1} does not miss its deadline, then it will either be scheduled in a slot where there is a hole, or it will cause a lower-priority subtask to be scheduled at a later slot. Inductively, this lower-priority subtask either misses a deadline or is scheduled correctly, in which case it may cause other subtasks to get scheduled later. Thus, no subtask will "shift" to an earlier slot. Repeating this argument, it is easy to see that adding T_{i-1} cannot cause the missed deadline at t_d to be met. Thus, τ' misses a deadline at t_d or earlier. This contradicts either (T2) or the minimality of t_d (refer to Definition 10). Therefore, $|w(T_{i-1})| > t$. $\square$

To avoid distracting boundary cases, we henceforth assume that the first subtask for each task T is some T_i, where $i > 1$. This can be assumed without loss of generality, because if T starts with T_1, then we can instead require it to start with T_{x+1}, where $x = T.e$; by part (b) of Lemma 9, the resulting release times and deadlines of the subtasks of T will be identical, implying that the schedule produced by PD^2 is identical as well.

Our proof proceeds by showing the existence of certain schedules for task set τ. To facilitate our description of these schedules, we find it convenient to totally order all subtasks in τ. Let $\prec$ be an irreflexive total order that is consistent with the $\preceq$ relation in the PD^2 priority definition, *i.e.*, $\prec$ is obtained by arbitrarily breaking any ties left by $\preceq$.

Definition 12 *A schedule S is defined to be k-compliant if and only if*

(i) *S is valid,*
(ii) *the first k subtasks according to $\prec$ are scheduled in accordance with PD^2, and*
(iii) *the remaining subtasks are scheduled within their Pfair windows (i.e., they are not early-released).*

We now present two lemmas. The second of these, Lemma 14, allows us to inductively prove that τ does not miss a deadline at t_d as originally assumed. Lemma 13 deals with a situation arising in one of the cases in Lemma 14. According to Lemma 13, if subtasks T_i, U_j, and U_{j+1} are scheduled as shown in Figure 9(a), then by swapping some subtasks, it is possible to obtain a schedule in which U_j is not scheduled in slot t.

Lemma 13 *Let S be a valid schedule for τ such that for light tasks T and U and $t < t'$, U_j is scheduled in slot t, T_i is eligible at t, and U_{j+1} and T_i are both scheduled in slot t'. Further, $r(U_j) = t$, $d(U_j) = t' + 1$, $r(U_{j+1}) = t'$,*

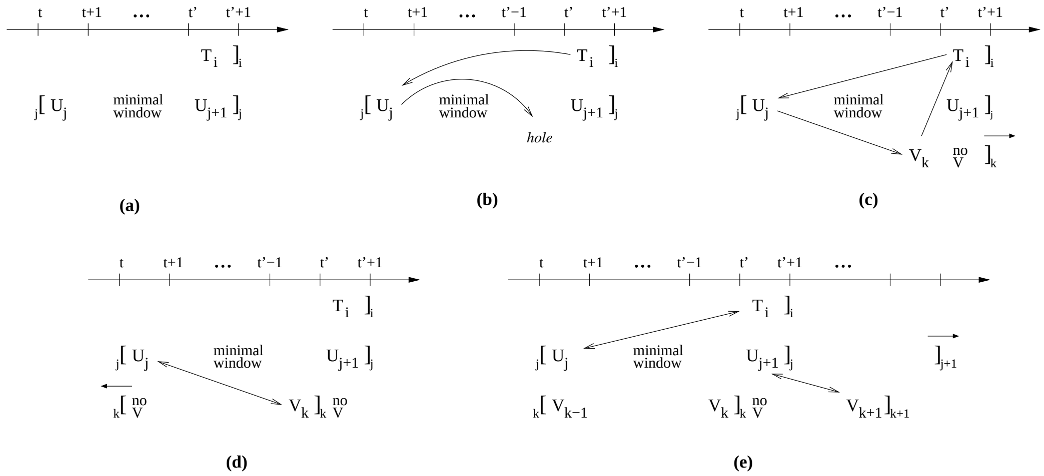

Fig. 9. We use the following notation in this and subsequent figures. "[" and "]" indicate the release and deadline of a subtask; subscripts indicate which subtask. Each task is shown on a separate line. An arrow from subtask T_i to subtask U_j indicates that T_i is now scheduled in place of U_j. An arrow over "[" (or "]") indicates that the actual position of "[" (or "]") can be anywhere in the direction of the arrow. Time is divided into unit-time slots that are numbered. (Although all slots are actually of the same length, due to formatting concerns, they do not necessarily appear as such in our figures.) If T_i is released at slot t, then "[" is aligned with the left side of slot t. If T_i has a deadline at time $t+1$, then "]" is aligned with the right side of slot t. In insets (c)–(e), no subtask of V is scheduled in slot t'. **(a)** Conditions of Lemma 13. **(b)** There is a hole in slot $t'-1$. **(c)** $d(V_k) > t'$. **(d)** $d(V_k) = t'$ and $r(V_k) \leq t$. **(e)** $d(V_k) = t'$ and $r(V_k) \geq t$.

$d(T_i) = t'+1$, and $w(U_j)$ is a minimal window of U. If all subtasks scheduled at or after t in S are scheduled within their Pfair windows, then there exists a valid schedule S' also satisfying this property such that $U \notin S'_t$, $S_u = S'_u$ for $0 \leq u < t$, and $S_t - \{U\} \subset S'_t$.

Proof. Our goal is to construct S' by swapping U_j with a later subtask. Unfortunately, T_i and U_j cannot be swapped directly because this would result in a schedule in which two subtasks of U are scheduled in the same slot. Instead, we identify another subtask V_k that can be used as an intermediate between U_j and T_i for swapping. Because T and U are both light, by (P5), all windows of each span at least three slots. Because $r(U_j) = t$ and $d(U_j) = t'+1$, this implies that $t' \geq t+2$ and T and U are not scheduled in slot $t'-1$.

If there is a hole in slot $t'-1$, then the swapping shown in Figure 9(b) gives the required schedule.

We henceforth assume that there is no hole in slot $t'-1$. In this case, because U is scheduled at t' but not at $t'-1$, there exists a task V that is scheduled at $t'-1$ but not at t'. Let V_k be the subtask of V scheduled at $t'-1$. If $d(V_k) > t'$, then the swapping shown in Figure 9(c) gives the desired schedule. In the rest of this proof, we assume the following.

$$d(V_k) = t' \tag{10}$$

If $r(V_k) < t$ or if $r(V_k) = t \wedge V \notin S_t$, then the swapping shown in Figure 9(d) produces the desired schedule. The remaining possibility to consider is

$$(r(V_k) > t) \ \vee \ (r(V_k) = t \ \wedge \ V \in S_t). \tag{11}$$

In this case, we show that the swapping in Figure 9(e) is valid. (This inset actually depicts the case $r(V_k) = t \wedge V \in S_t$.) From (10) and the statement of the lemma, we have $d(V_k) = d(U_j) - 1$. Also, from (11), and the statement of the lemma, we have $r(V_k) \geq r(U_j)$. Therefore,

$$|w(V_k)| < |w(U_j)|. \tag{12}$$

Because $w(U_j)$ is a minimal window of U, $|w(U_{j+1})| \geq |w(U_j)|$. By definition, $|w(U_{j+1})| = d(U_{j+1}) - r(U_{j+1})$, which implies that $d(U_{j+1}) = |w(U_{j+1})| \ + \ t'$. Therefore,

$$d(U_{j+1}) \geq t' + |w(U_j)|. \tag{13}$$

Now, by (10) and by part (a) of Lemma 9, $r(V_{k+1})$ is either $t' - 1$ or t'. We now show that in either case, $d(V_{k+1}) \leq t' + |w(V_k)|$. If $r(V_{k+1}) = t'$ (in which case $b(V_k) = 0$), then by (P1), $|w(V_{k+1})| = |w(V_k)|$. By definition, $|w(V_{k+1})| = d(V_{k+1}) - r(V_{k+1})$. Therefore, $d(V_{k+1}) = t' + |w(V_k)|$.

On the other hand, if $r(V_{k+1}) = t' - 1$, then $|w(V_{k+1})| \leq |w(V_k)| + 1$ (by (P3)). Because $|w(V_{k+1})| = d(V_{k+1}) - r(V_{k+1})$, it follows that $d(V_{k+1}) \leq t' + |w(V_k)|$.

Thus, in both cases, we have $d(V_{k+1}) \leq t' + |w(V_k)|$. By (12) and (13), this implies that $d(U_{j+1}) > d(V_{k+1})$. Therefore, by part (a) of Lemma 9, $r(U_{j+2}) \geq d(V_{k+1})$. This implies that no subtask of U is scheduled in the interval $[t' + 1, d(V_{k+1}))$. Thus, the swapping shown in Figure 9(e) is valid, and produces the required schedule. $\Box$

We now prove that a k-compliant schedule exists by induction on k. Note that a 0-compliant schedule is just a Pfair schedule (with no early releases), and the existence of such a schedule is guaranteed for any feasible task system. Also, if n subtasks are released in $[0, t_d)$, then an n-compliant schedule is a valid schedule that is fully in accordance with PD^2 over $[0, t_d]$. The following lemma gives the inductive step of the proof.

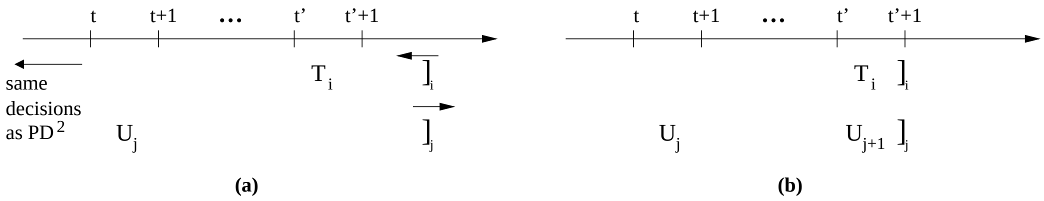

Fig. 10. **(a)** Conditions of Lemma 14. **(b)** The "difficult" case to consider.

Lemma 14 *If S is a valid k-compliant schedule for τ, then there exists a valid schedule S' for τ that is $(k+1)$-compliant.*

Proof. Let T_i be the $(k+1)^{st}$ subtask according to $\prec$. If T_i is scheduled in S in accordance with PD^2, then take S' to be S. Otherwise, there exists a time slot t such that T_i is eligible at t but scheduled later, and either **(i)** there is a hole in t, or **(ii)** some subtask ordered after T_i by $\prec$ is scheduled at t. In the former case, we can easily rectify the situation by scheduling T_i at t. Hence, in the rest of the proof, we assume that (ii) holds.

Let t be the earliest such time slot, and let U_j be the lowest-priority subtask scheduled at t. Thus, $T_i \prec U_j$. Let t' be the slot where T_i is scheduled, as depicted in Figure 10(a). Note that t may or may not lie within T_i's Pfair window; this depends on whether T_i is an early-release subtask. However, because S is k-compliant, t lies within U_j's Pfair window and t' lies within T_i's Pfair window. In the rest of the proof, we show that S' can be obtained from S by swapping T_i and U_j and perhaps some other subtasks. In all cases, the subtasks that are swapped include T_i and subtasks ranked after T_i by $\prec$. Since S is k-compliant, all such subtasks are scheduled within their Pfair windows.

Because S is a valid schedule and $T_i \prec U_j$, by the PD^2 priority definition, we have

$$t < t' < d(T_i) \leq d(U_j). \qquad (14)$$

We first show that U_{j+1} cannot be scheduled before slot t'. Because $T_i \prec U_j$, we have $T_i \prec U_{j+1}$, and hence U_{j+1} is not early-released. Now, because $d(U_j) > t'$ (by (14)), by part (a) of Lemma 9, $r(U_{j+1}) \geq t'$. Thus, U_{j+1} cannot be scheduled before t'. Further, it can be scheduled at t' if and only if $r(U_{j+1}) = t'$.

If no subtask of U is scheduled in the interval $[t+1, t'+1)$, then T_i and U_j can be directly swapped to get the required schedule. In the rest of the proof, we assume that U_{j+1} is scheduled in slot t' (*i.e.*, in the interval $[t', t'+1)$). As shown above, in this case $r(U_{j+1}) = t'$. Therefore, by part (a) of Lemma 9, $d(U_j)$ is either t' or $t'+1$. By (14), it follows that $d(U_j) = t'+1$ and hence, $d(T_i) = t'+1$. Because $d(U_j) = r(U_{j+1}) + 1$, we have $b(U_j) = 1$. This implies

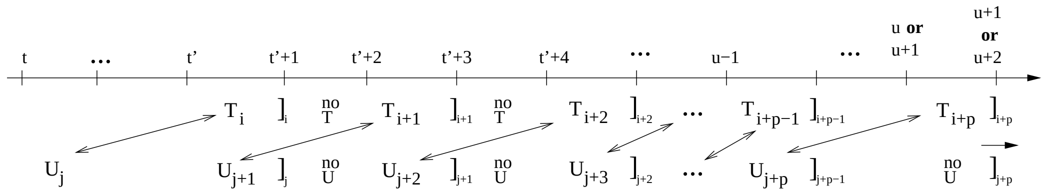

Fig. 11. Case 2. T is heavy, U is light, and $d(T_i) = d(U_j)$.

that $b(T_i) = 1$, since $T_i \prec U_j$. Thus, we have the following.

$$U_{j+1} \in S_{t'} \ \wedge\ d(T_i) = d(U_j) = t' + 1 \ \wedge\ r(T_{i+1}) = r(U_{j+1}) = t' \qquad (15)$$

These conditions are depicted in Figure 10(b). We now consider four cases depending on the weights of T and U.

Case 1: *T* is light and *U* is heavy. By the PD^2 priority definition and the definition of a group deadline, T cannot have higher priority than U at time t.

Case 2: *T* is heavy and *U* is light. In this case, we show that the swapping in Figure 11 is valid. (The argument hinges on the fact that U's windows are at least as long as T's — see Figure 11.) By (P7), all windows of T are of length either two or three. Further, by (P8), $|w(T_k)| = 2$ if $b(T_k) = 0$, and by (B) (refer to Section 3), $b(T_k) = 0$ if T_k is the last subtask of a job. Because $b(T_i) = 1$, it follows that there exists an $r \geq 1$ such that

$$|w(T_{i+r})| = 2 \ \wedge\ (\forall k : 0 < k < r :: |w(T_{i+k})| = 3 \ \wedge\ b(T_{i+k}) = 1).$$

(Note that r could be one, *i.e.*, $w(T_{i+1})$ could be a 2-window.) Because U is light, by (P5), $|w(U_k)| \geq 3$ for all k. This implies that $d(T_{i+r}) < d(U_{j+r})$. Let q denote the smallest value of k that satisfies $d(T_{i+k}) < d(U_{j+k})$. (Note that $q \leq r$.) Then, $d(T_{i+q}) < d(U_{j+q})$, and for all $k \in [1, q-1]$,

$$d(T_{i+k}) = d(U_{j+k}) \ \wedge\ |w(T_{i+k})| = |w(U_{j+k})| = 3 \ \wedge\ b(T_{i+k}) = 1.$$

Because $d(T_{i+q}) < d(U_{j+q})$, by part (a) of Lemma 9, we have $d(T_{i+q}) \leq r(U_{j+q+1})$. Thus, T_{i+q} is scheduled before U_{j+q+1}. Let p be the smallest value for k such that T_{i+k} is scheduled prior to U_{j+k+1}. (Again, note that $p \leq q$). To summarize:

- $(\forall k : 0 < k < p :: d(T_{i+k}) = d(U_{j+k}) \ \wedge\ |w(T_{i+k})| = 3 \ \wedge\ |w(U_{j+k})| = 3 \ \wedge\ b(T_{i+k}) = 1) \ \wedge\ d(T_{i+p}) \leq d(U_{j+p})$,

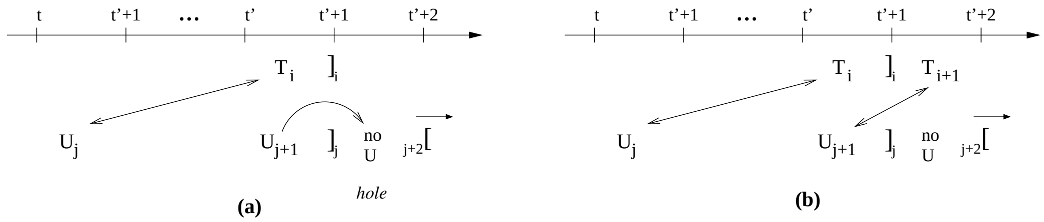

Fig. 12. Case 3. **(a)** Some processor is idle in slot $t' + 1$. **(b)** T_{i+1} is scheduled in slot $t' + 1$.

- T_{i+p} is scheduled before U_{j+p+1}, and
- for each k in the range $0 < k < p$, T_{i+k} is *not* scheduled before U_{j+k+1}.

It is straightforward to see that the relevant subtasks are scheduled as shown in Figure 11 and the depicted swapping is valid.

Case 3: Both *T* and *U* are light. (This case and Case 4 are somewhat lengthy.) Again, the situation under consideration is as depicted in Figure 10(b). Because U is light, by (P5), $|w(U_{j+1})| \geq 3$. Because $r(U_{j+1}) = t'$ (by (15)), this implies that $d(U_{j+1}) > t' + 2$. Therefore, by part (a) of Lemma 9, $r(U_{j+2}) \geq t' + 2$ and hence, U is not scheduled in slot $t' + 1$. If there is a hole in slot $t' + 1$, then the swapping shown in Figure 12(a) gives the required schedule. Otherwise, if T_{i+1} is scheduled in slot $t'+1$, then the swapping shown in Figure 12(b) gives the required schedule. In the rest of Case 3, we assume that there is no hole in slot $t' + 1$ and T_{i+1} is not scheduled there. We now show that one of the swappings shown in Figures 13 and 14 is valid.

Because U is scheduled in slot t' but not in slot $t' + 1$, and because there are no holes in slot $t' + 1$, there exists a task V that is scheduled in slot $t' + 1$ but not in slot t'. Let V_k be the subtask of V scheduled in slot $t'+1$. Because S is a valid schedule, $d(V_k) \geq t'+2$. Therefore, by (15), $d(V_k) > d(T_i)$, which implies that $T_i \prec V_k$. It follows that V_k is not early-released and hence, $r(V_k) \leq t'+1$. If $r(V_k) < t'+1$, then the swapping shown in Figure 13(a) produces the desired schedule. In the rest of the proof for Case 3, we assume

$$r(V_k) = t' + 1, \tag{16}$$

in which case this swapping is not valid. By (15) and (16) we have the following.

$$r(V_k) = r(T_{i+1}) + 1 \tag{17}$$

We first dispense with the case $V_{k-1} \notin \tau$. In this case, by Lemma 11, either $(b(V_{k-1}) = 0 \ \wedge \ |w(V_{k-1})| > t' + 1)$ or $(b(V_{k-1}) = 1 \ \wedge \ |w(V_{k-1})| > t' + 2)$. In the former case, by (P1), $|w(V_k)| = |w(V_{k-1})|$; in the latter case, by (P4), $|w(V_k)| \geq |w(V_{k-1})| - 1$. Thus, in either case, $|w(V_k)| > t' + 1$. Further, since

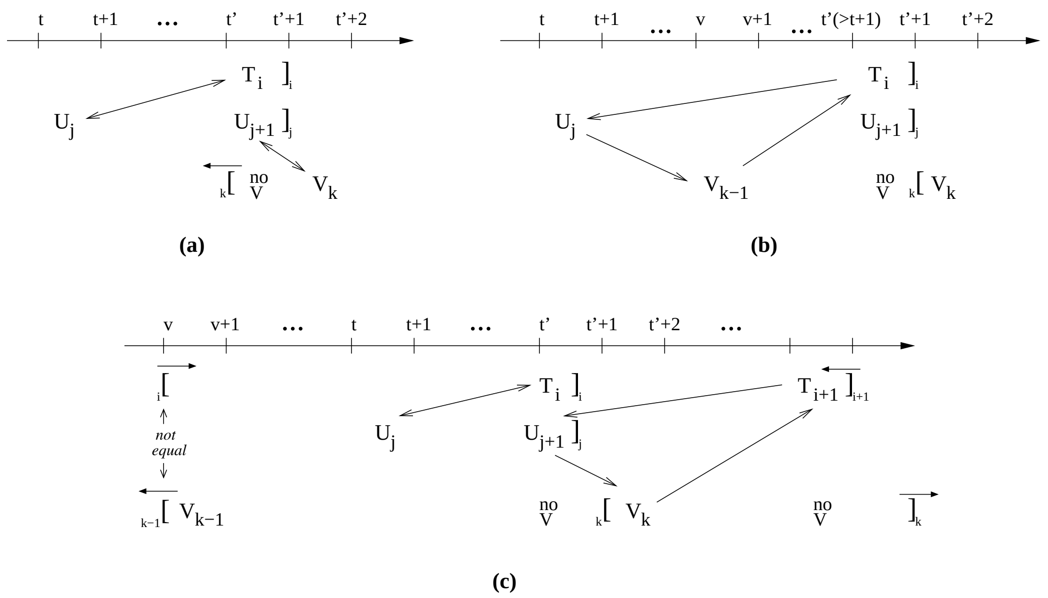

Fig. 13. Case 3 (continued). **(a)** $r(V_k) \le t'$. **(b)** $r(V_k) = t'+1$ and V_{k-1} is scheduled at v, $t < v < t'$. **(c)** $r(V_k) = t'+1$ and $d(V_k) > d(T_{i+1})$.

$T_i \in \tau$ and $d(T_i) = t'+1$ (by 15), $|w(T_i)| \le t'+1$ (recall that slots are numbered from 0). Therefore, $|w(V_k)| > |w(T_i)|$, *i.e.*, $|w(V_k)| \ge |w(T_i)| + 1$. By (P4), $|w(T_i)|+1 \ge |w(T_{i+1})|$. Thus, $|w(V_k)| \ge |w(T_{i+1})|$. Because $r(V_k) = r(T_{i+1})+1$ (by (17)), this implies that $r(V_k)+|w(V_k)| \ge r(T_{i+1})+|w(T_{i+1})|+1$. Therefore, $d(V_k) > d(T_{i+1})$, and hence no subtask of V is scheduled in $[t'+2, d(T_{i+1}))$. Thus, the swapping in Figure 13(c) is valid. (This figure actually depicts $V_{k-1} \in \tau$, but the swapping depicted is applicable nonetheless.) In the rest of the proof for Case 3, we assume that $V_{k-1} \in \tau$.

Note that because $r(V_k) = t'+1$ (by (16)), either $d(V_{k-1}) = t'+2$ or $d(V_{k-1}) = t'+1 \ \wedge \ b(V_{k-1}) = 0$. By (15), $d(T_i) = d(U_j) = t'+1$ and $b(T_i) = b(U_j) = 1$. Therefore, by the PD^2 priority definition, we have the following.

$$T_i \prec V_{k-1} \text{ and } U_j \prec V_{k-1} \tag{18}$$

If V_{k-1} is scheduled in the interval $[t+1, t')$, then the swapping shown in Figure 13(b) is valid. If V_{k-1} is not scheduled in $[t+1, t')$, then it is scheduled at or before t. Because $U_j \prec V_{k-1}$ (by (18)), V_{k-1} is not scheduled in slot t, as this would contradict our choice of U_j as the lowest-priority subtask scheduled at t.

In the rest of Case 3, we assume that V_{k-1} is scheduled at a time $v < t$. Now, it must be the case that T_i was not eligible to be scheduled at time v. To see this, note that if T_i *were* eligible at time v, then it should have been scheduled there because $T_i \prec V_{k-1}$ (by (18)). This contradicts our starting assumption that T_i should be scheduled at t. Thus, either $r(T_i) > v$ or $r(T_i) = v \ \wedge \ T_{i-1} \in S_v$.

(Note that one of these assertions holds even if T_i is an early-release subtask.) Because $T_i \prec V_{k-1}$ (by (18)), V_{k-1} is not early-released. Therefore, $r(V_{k-1}) \leq v$, which implies that either $(r(T_i) > r(V_{k-1}))$ or $(r(T_i) = v \wedge r(V_{k-1}) = v \wedge T_{i-1} \in S_v)$. We consider these two subcases next.

Subcase 3.A: $r(T_i) > r(V_{k-1})$**.** We show that $d(V_k) > d(T_{i+1})$, which implies that the swapping shown in Figure 13(c) is valid. There are two possibilities to consider, depending on the value of $b(V_{k-1})$.

(1) $b(V_{k-1}) = 0$. By (9), we have $d(V_{k-1}) = r(V_k) = t' + 1$ (by (16)). By (15), this implies that $d(V_{k-1}) = d(T_i)$. Since $b(V_{k-1}) = 0$, by (P1), $|w(V_k)| = |w(V_{k-1})|$. Because $r(V_{k-1}) < r(T_i)$ (our assumption for Subcase 3.A) and $d(V_{k-1}) = d(T_i)$, we have $|w(V_{k-1})| \geq |w(T_i)| + 1$. Therefore, $|w(V_k)| \geq |w(T_i)| + 1$. By (P4), $|w(T_i)| + 1 \geq |w(T_{i+1})|$, which implies that $|w(V_k)| \geq |w(T_{i+1})|$. Because $r(V_k) = r(T_{i+1}) + 1$ (by (17)), this implies that $r(V_k) + |w(V_k)| \geq r(T_{i+1}) + 1 + |w(T_{i+1})|$. Therefore, $d(V_k) > d(T_{i+1})$.

(2) $b(V_{k-1}) = 1$. By the definition of $b(V_{k-1})$, we have $d(V_{k-1}) = r(V_k) + 1$. Hence, by (16), $d(V_k) = t' + 2$. By (15), this implies that $d(V_{k-1}) = d(T_i) + 1$. Along with $r(V_{k-1}) < r(T_i)$ (our assumption for Subcase 3.A), this implies that

$$|w(V_{k-1})| \geq |w(T_i)| + 2. \tag{19}$$

By (P4), we have $|w(V_k)| \geq |w(V_{k-1})| - 1$ and $|w(T_i)| \geq |w(T_{i+1})| - 1$. Hence, by (19), $|w(V_k)| + 1 \geq |w(T_{i+1})| - 1 + 2$, *i.e.*, $|w(V_k)| \geq |w(T_{i+1})|$. Because $r(V_k) = r(T_{i+1}) + 1$ (by (17)), this implies that $d(V_k) > d(T_{i+1})$.

Subcase 3.B: $r(T_i) = v \wedge r(V_{k-1}) = v \wedge T_{i-1} \in S_v$**.** Reasoning as in Subcase 3.A, it follows that $d(V_k) \geq d(T_{i+1})$. By part (a) of Lemma 9, we have the following.

$$r(V_{k+1}) + 1 \geq d(V_k) \geq d(T_{i+1}) \tag{20}$$

We now show that a valid swapping exists in all cases. First, note that if T_{i+1} is scheduled before V_{k+1}, then the swapping shown in Figure 13(c) is still valid. This will be the case if $r(V_{k+1}) \geq d(T_{i+1})$. In the rest of the proof for Subcase 3.B, we assume that T_{i+1} is *not* scheduled before V_{k+1}. By (20), this can happen only if there exists a v' that satisfies the following (see Figure 14).

$$r(V_{k+1}) = v' \wedge d(V_k) = v' + 1 \wedge d(T_{i+1}) = v' + 1 \wedge V_{k+1} \in S_{v'} \wedge T_{i+1} \in S_{v'} \tag{21}$$

The following property is used several times in the reasoning that follows.

Claim 15 *$w(V_k)$ is a minimal window of V.*

Proof. By part (a) of Lemma 9, (16) implies that $d(V_{k-1})$ is either $t'+1$ or $t'+2$. If $d(V_{k-1}) = t'+1$, then $b(V_{k-1}) = 0$ and by (P2), $w(V_k)$ is a minimal window. On the other hand, if $d(V_{k-1}) = t'+2$, then we have the following:

- T_i and V_{k-1} are both released at slot v (our assumption for Subcase 3.B),
- $d(T_i) = t'+1$ (by (15)) and $d(V_{k-1}) = t'+2$,
- $r(T_{i+1}) = t'$ (by 15) and $r(V_k) = t'+1$ (by (16)), and
- T_{i+1} and V_k have equal deadlines (by (21)).

Therefore, $|w(T_i)| = |w(V_{k-1})| - 1$ and $|w(T_{i+1})| = |w(V_k)| + 1$. By (P3), $|w(T_{i+1})| \le |w(T_i)| + 1$. Therefore, $|w(V_k)| + 1 \le |w(V_{k-1})|$, *i.e.*, $|w(V_k)| \le |w(V_{k-1})| - 1$. By (P4), this implies that $|w(V_k)| = |w(V_{k-1})| - 1$. By part (d) of Lemma 9, this implies that $w(V_k)$ is a minimal window of V. $\square$

To continue, if there is a hole in slot $v'-1$, then we can left-shift T_{i+1} from v' to $v'-1$ and apply the swapping shown in Figure 13(c). In the rest of Subcase 3.B, we assume that there is no hole in slot $v'-1$.

We now prove that V must be a light task. Because $r(V_k) = t'+1$ (by (16)), $d(V_{k-1})$ is either $t'+1$ or $t'+2$, and if $d(V_{k-1}) = t'+1$, then $b(V_{k-1}) = 0$. Thus, because $r(T_i) = r(V_{k-1}) = v$ (our assumption for Subcase 3.B) and $d(T_i) = t'+1$ (by (15)), either $d(V_{k-1}) = d(T_i)+1$ or $d(V_{k-1}) = d(T_i) \ \wedge \ b(V_{k-1}) = 0$. Therefore, either $|w(V_{k-1})| = |w(T_i)|+1$ or $|w(V_{k-1})| = |w(T_i)| \ \wedge \ b(V_{k-1}) = 0$. Because T is light, by (P5), $|w(T_i)| \ge 3$. Therefore, either $|w(V_{k-1})| \ge 4$ or $|w(V_{k-1})| \ge 3 \ \wedge \ b(V_{k-1}) = 0$. In either case, V cannot be a heavy task: if it were heavy, then by (P7), all windows of V would be of length two or three, and by (P8), $b(V_{k-1}) = 0$ would imply that $|w(V_{k-1})| = 2$. Thus, V is a light task.

By (P5), $|w(V_k)| \ge 3$. By (16) and (21), $w(V_k) = [t'+1, v'+1)$; hence, $v' \ge t'+3$. Because $V_k \in S_{t'+1}$ and $V_{k+1} \in S_{v'}$, this implies that $V \notin S_{v'-1}$. Thus, because there are no holes in slot $v'-1$, there exists a task W that is scheduled in slot $v'-1$ but not in slot v'. Let W_h be the subtask of W scheduled in slot $v'-1$. We now show that at least one of the swappings in Figure 14 is valid.

If $d(W_h) \ge v'+1$, then the swapping in Figure 14(a) is clearly valid. We henceforth assume

$$d(W_h) = v'. \tag{22}$$

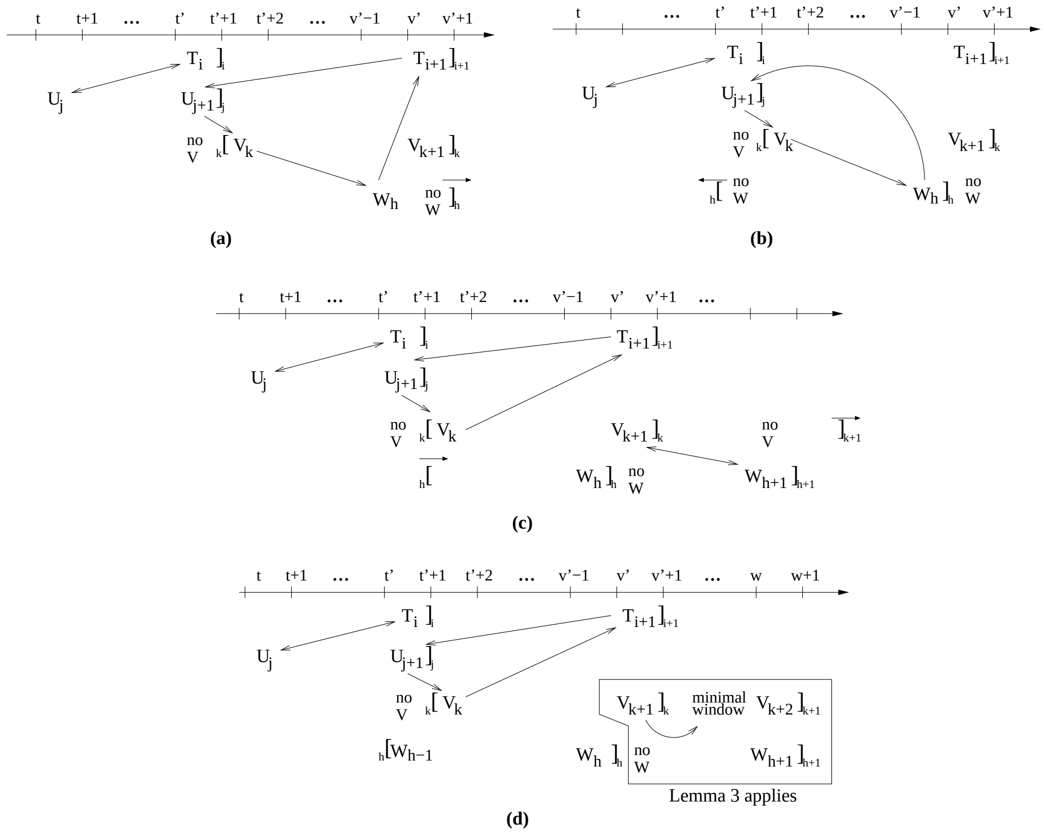

Fig. 14. Subcase 3.B of Case 3. **(a)** $r(V_k) = t' + 1$, $d(V_k) = d(T_{i+1}) = v' + 1$, and $d(W_h) \geq v' + 1$. **(b)** $r(V_k) = t' + 1$, $d(V_k) = d(T_{i+1}) = v' + 1$, $d(W_h) = v'$, $r(W_h) \leq t'$, and W is not scheduled at t'. **(c)** $r(V_k) = t' + 1$, $d(V_k) = d(T_{i+1}) = v' + 1$, $d(W_h) = v'$, and $r(W_h) > t'$ **(d)** $r(V_k) = t' + 1$, $d(V_k) = d(T_{i+1}) = v' + 1$, $d(W_h) = v'$, $r(W_h) = t'$, and W is scheduled at t'.

If $(r(W_h) < t') \ \vee \ (r(W_h) = t' \ \wedge \ W \notin S_{t'})$, then the swapping shown in Figure 14(b) is valid. This leaves the following two possibilities.

(1) $r(W_h) > t'$. In this case, we show that $d(W_{h+1}) < d(V_{k+1})$, which implies that the swapping in Figure 14(c) is valid. Because $r(W_h) > t'$, by (16), $r(W_h) \geq r(V_k)$. By (21) and (22), $d(W_h) < d(V_k)$. Therefore, $|w(W_h)| < |w(V_k)|$ (see Figure 14(c)). Because $w(V_k)$ is a minimal window of V (by Claim 15), $|w(V_k)| \leq |w(V_{k+1})|$. Thus,

$$|w(W_h)| < |w(V_{k+1})|. \tag{23}$$

Now, consider $b(W_h)$.

If $b(W_h) = 0$, then by (22), $r(W_{h+1}) = v'$. Thus, by (21), $r(W_{h+1}) = r(V_{k+1})$. In addition, by (P1), $|w(W_{h+1})| = |w(W_h)|$. Hence, by (23), we have $|w(W_{h+1})| < |w(V_{k+1})|$. Therefore, $d(W_{h+1}) < d(V_{k+1})$.

If $b(W_h) = 1$, then by (22), $r(W_{h+1}) = v' - 1$, which by (21) implies that $r(W_{h+1}) < r(V_{k+1})$. In addition, by (P3), $|w(W_{h+1})| \leq |w(W_h)| + 1$. Hence, by (23), we have $|w(W_{h+1})| \leq |w(V_{k+1})|$. Therefore, $d(W_{h+1}) < d(V_{k+1})$.

(2) $r(W_h) = t' \wedge W \in S_{t'}$. In this case, analysis similar to that above shows that $d(W_{h+1}) \leq d(V_{k+1})$. Let $d(W_{h+1}) = w+1$. If $d(W_{h+1}) < d(V_{k+1})$ or if $d(W_{h+1}) = d(V_{k+1}) \wedge V_{k+2} \notin S_w$, then the swapping shown in Figure 14(c) is valid. (The figure actually shows W_h being released after time t', but the swapping is still valid.) On the other hand, if $d(W_{h+1}) = d(V_{k+1})$ and $V_{k+2} \in S_w$, then we have the following (see Figure 14(d)).

(a) $r(W_h) = t'$ and $r(V_k) = t' + 1$ (by (16)),

(b) $d(W_h) = v'$ (by (22)) and $d(V_k) = v' + 1$ (by (21)),

(c) $r(V_{k+1}) = v'$ (by (21)) and $r(W_{h+1})$ is either v' or $v' - 1$ (by part (a) of Lemma 9 because $d(W_h) = v'$), and

(d) $d(V_{k+1}) = d(W_{h+1}) = w + 1$.

By (a) and (b) above, we have

$$|w(W_h)| = |w(V_k)|. \tag{24}$$

We now show that $|w(V_{k+1})| \leq |w(V_k)|$. If $r(W_{h+1}) = v' - 1$, then $|w(V_{k+1})| = |w(W_{h+1})| - 1$. By (P3), $|w(W_{h+1})| - 1 \leq |w(W_h)|$. Therefore, by (24), $|w(V_{k+1})| \leq |w(V_k)|$. On the other hand, if $r(W_{h+1}) = v'$, then $|w(V_{k+1})| = |w(W_{h+1})|$ and $b(W_h) = 0$ (because $d(W_h) = v'$, by (22)). Therefore, by (P1), $|w(W_{h+1})| = |w(W_h)|$. Thus, $|w(V_{k+1})| = |w(W_h)|$, and by (24), $|w(V_{k+1})| = |w(V_k)|$.

Because $w(V_k)$ is a minimal window of V (by Claim 15), this implies that $w(V_{k+1})$ is a minimal window as well. Thus, by Lemma 13, there exists a schedule in which V_{k+1} is not scheduled at time v'. The swapping shown in Figure 14(d) is therefore valid.

This completes Case 3.

Case 4: Both T and U are heavy. In the proof for this case, we refer to successive group deadlines of a task. The following notation will be used. If g is a group deadline of task X, then $pred(X, g)$ (respectively, $succ(X, g)$) denotes the group deadline of task X that occurs immediately before (respectively, after) g. For example, in Figure 2, $pred(T, 8) = 4$ and $succ(T, 8) = 11$.

As before, we are dealing with the situation depicted in Figure 10(b). Because T_i has higher priority than U_j at time t according to PD^2, $D(U_j) \leq D(T_i)$. Recall that $T_i \in S_{t'}$ and $d(T_i) = t' + 1$ (refer to (15) and Figure 10(b)), *i.e.*, T_i is scheduled in the last slot of its window. By the definition of a group deadline, all subsequent subtasks with deadlines at or before $D(T_i)$ have windows of length two that overlap with the window of their predecessor subtask. Therefore, each such subtask is scheduled in the last slot of its window.

Let u be the earliest time after t' such that $U \notin S_u$. Because U_{j+1} is scheduled in the first slot of its window, there exists a time before the group deadline of U_j such that U is not scheduled at that time. Thus, $u < D(U_j)$. Because

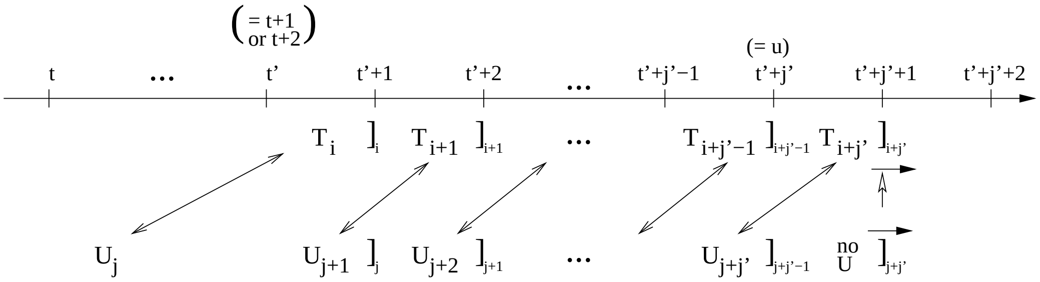

Fig. 15. Case 4. We use the following notation in this figure and Figures 16–21. A group deadline at time t is denoted by an up-arrow that is aligned with time t. A left- or right-pointing arrow over an up-arrow indicates a group deadline that may be anywhere in the direction of the arrow. $D(T_i) > D(U_j)$ or $u+1 = D(T_i) \wedge T \in S_u$.

$D(U_j) \leq D(T_i)$, this implies that $u < D(T_i)$. If $u < D(T_i) - 1$ or $u + 1 = D(T_i) \wedge T \in S_u$ holds then we have the following (refer to Figure 15).

- No subtask of U is scheduled in slot u (*i.e.*, in $[u, u+1)$).
- In all slots in $[t', u+1)$, a subtask of T is scheduled in the last slot of its window.
- In all slots in $[t', u)$, a subtask of U is scheduled in the first slot of its window.

This implies that the swapping in Figure 15 is valid.

The remaining possibility is $u + 1 = D(T_i) \wedge T \notin S_u$. In this case, because $u + 1 \leq D(U_j) \leq D(T_i)$, we have

$$D(T_i) = D(U_j) \ \wedge \ D(U_j) = u + 1 \ \wedge \ T \notin S_u. \tag{25}$$

Let $U_{j+j'}$ be the subtask of U scheduled at $u - 1$. Then, $u = t' + j'$, as shown in Figure 16(a). By the definition of a group deadline, each of the subtasks $T_{i+1}, \ldots, T_{i+j'-1}$ and $U_{j+1}, \ldots, U_{j+j'-1}$ has a window of length two. (If not, T_i's and U_j's group deadlines will be earlier.) Therefore, $d(T_{i+j'-1}) = u$. By part (a) of Lemma 9, this implies that $r(T_{i+j'})$ is either u or $u-1$. If $r(T_{i+j'}) = u$, then $b(T_{i+j'-1}) = 0$, which implies that $D(T_i) = u$. This contradicts (25). Therefore, $r(T_{i+j'}) = u - 1$. Since $T_{i+j'-1}$ is scheduled in slot $u-1$ and $T \notin S_u$ (by (25)), it follows that $d(T_{i+j'})$ must be $u+2$. (By (P7), it cannot be later.) Thus, as shown in Figure 16(a), $w(T_{i+j'})$ is a 3-window starting at slot $u - 1$, and $T_{i+j'}$ is scheduled in slot $u + 1$, *i.e.*, slot $t' + j' + 1$.

If there is a hole in slot u, then shifting subtask $U_{j+j'}$ to slot u produces a situation in which a swapping similar to that in Figure 15 can be applied. We henceforth assume there is no hole in slot u.

Our strategy now is to identify another task to use as an intermediate for swapping. Because T and U are scheduled at $u - 1$ but not at u, and because there are no holes in u, there exists a task V that is scheduled at u but not

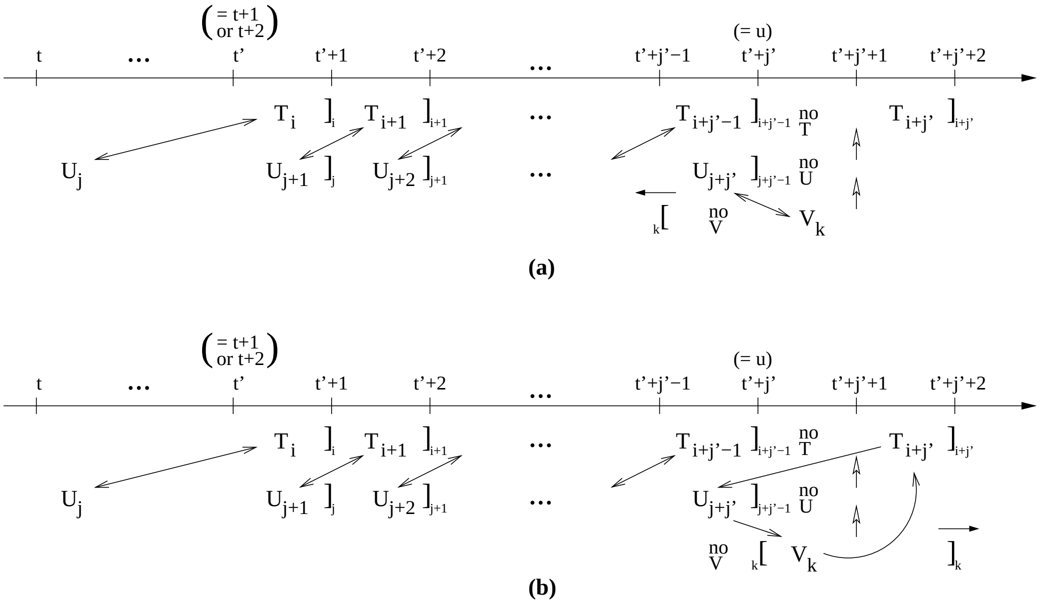

Fig. 16. Case 4 (continued). **(a)** $D(T_i) = D(U_j)$ and $r(V_k) < t' + j'$. **(b)** $D(T_i) = D(U_j)$, $r(V_k) = t' + j'$, and $V \notin S'_{t'+j'+1}$.

at $u - 1$. Let V_k be the subtask of V scheduled at u. If $r(V_k) < u$, then the swapping in Figure 16(a) is valid, and if $r(V_k) = u \ \wedge \ V \notin S_{u+1}$, then the swapping in Figure 16(b) is valid. In the rest of the proof, we assume

$$r(V_k) = u \ \wedge \ V \in S_{u+1}.$$

We now show that V is a heavy task. Note that $V \in S_{u+1}$ implies that $r(V_{k+1}) \leq u + 1$. Therefore, by part (a) of Lemma 9, $d(V_k) \leq u + 2$. Because $r(V_k) = u$, by (P5) and (P7), $d(V_k) \geq u + 2$. Therefore, we have $d(V_k) = u + 2$, which implies that $|w(V_k)| = 2$. Therefore, by (P6), V is heavy.

Consider $D(V_k)$, *i.e.*, the group deadline of V_k. Let v be the earliest slot after u such that $V \notin S_v$. Since V_{k+1} is scheduled in the first slot of its window, we have

$$v + 1 \leq D(V_k). \tag{26}$$

(See Figure 17.) Let $V_{k+i'}$ be the subtask of V that is scheduled in slot $v - 1$. If either $D(T_{i+j'}) > v + 1$ or $D(T_{i+j'}) = v + 1 \ \wedge \ b(T_{i+j'+i'}) = 0$, then $T_{i+j'+i'}$ is scheduled in slot v. To see why, note that $T_{i+j'}$ is scheduled in the last slot of its window and this forces all subtasks of T until its group deadline to be scheduled in the last of their windows. Thus, the swapping shown in Figure 17 is valid. In the rest of the proof, we assume that neither of these conditions holds, *i.e.*, we assume the following.

$$(D(T_{i+j'}) < v + 1) \ \vee \ (D(T_{i+j'}) = v + 1 \ \wedge \ b(T_{i+j'+i'}) = 1) \tag{27}$$

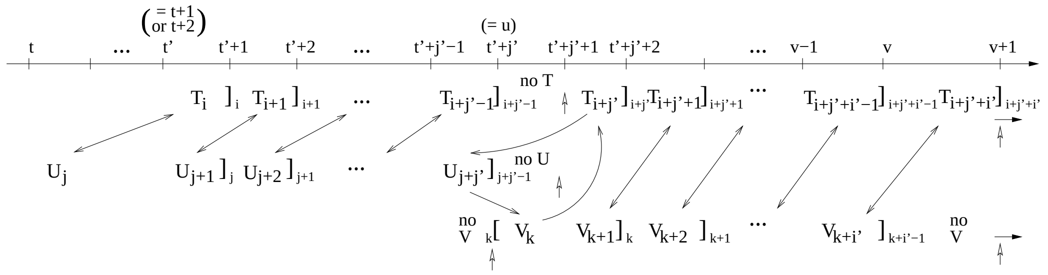

Fig. 17. Case 4 (continued). $D(T_i) = D(U_j)$, $r(V_k) = t' + j'$, $D(T_{i+j'}) = v + 1$ and $T_{i+j'+i'}$ is scheduled in slot v.

We claim that u is a group deadline of V. As seen in Figure 17, V_{k-1} is not scheduled in slot $u - 1$. (Note that $V_{k-1} \in \tau$ because its window fits in the interval $[0, u + 1)$.) Because $r(V_k) = u$, by part (a) of Lemma 9, $d(V_{k-1})$ is either u or $u + 1$. If $d(V_{k-1}) = u$, then $b(V_{k-1}) = 0$ and hence, u is a group deadline. If $d(V_{k-1}) = u + 1$, then we reason as follows. Because V is a heavy task, by (P7), $|w(V_{k-1})| \le 3$, which implies that $r(V_{k-1}) \ge u - 2$. Because V_{k-1} is not scheduled in slots $u - 1$ or u, the following must hold.

$$r(V_{k-1}) = u - 2 \tag{28}$$

This implies that $w(V_{k-1}) = [u - 2, u + 1)$. Therefore, u is a group deadline of V.

Having shown that u is a group deadline of V, we now show that $\mathit{pred}(V, u) \le \mathit{pred}(T, u+1)$. T has consecutive group deadlines at $u+1$ and $\mathit{succ}(T, u+1) = D(T_{i+j'})$. Therefore, by (P9), the difference between $u + 1$ and $\mathit{pred}(T, u + 1)$ is at most one more than $D(T_{i+j'}) - (u + 1)$, *i.e.*, $u + 1 - \mathit{pred}(T, u + 1) \le D(T_{i+j'}) - u$. Therefore,

$$\mathit{pred}(T, u + 1) \ge 2u - D(T_{i+j'}) + 1. \tag{29}$$

V has consecutive group deadlines at u and $\mathit{succ}(V, u) = D(V_k)$. Hence, by (P9), the difference between u and $\mathit{pred}(V, u)$ is at least one less than $D(V_k) - (u)$, *i.e.*, $u - \mathit{pred}(V, u) \ge D(V_k) - u - 1$. Thus,

$$\mathit{pred}(V, u) \le 2u - D(V_k) + 1. \tag{30}$$

By (26) and (27), $D(T_{i+j'}) \le D(V_k)$. Therefore, by (29) and (30), $\mathit{pred}(V, u) \le 2u - D(V_k) + 1 \le 2u - D(T_{i+j'}) + 1 \le \mathit{pred}(T, u + 1)$. Thus, $\mathit{pred}(V, u) \le \mathit{pred}(T, u + 1)$. This sequence of inequalities further implies that $\mathit{pred}(V, u) = \mathit{pred}(T, u+1)$ if and only if $2u - D(V_k) + 1 = 2u - D(T_{i+j'}) + 1$, *i.e.*, $D(T_{i+j'}) = D(V_k)$. By (26) and (27), this can be true only if $D(T_{i+j'}) = D(V_k) = v + 1$.

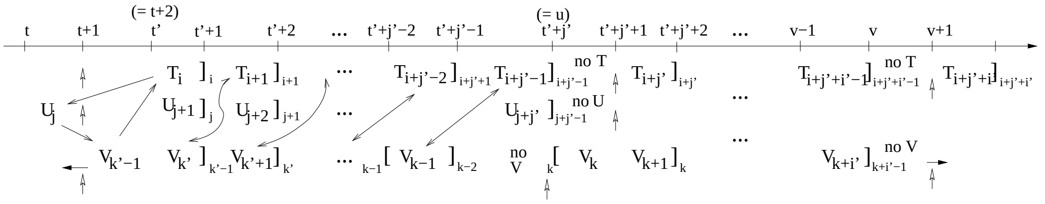

Fig. 18. Subcase 4.A. $t' = t + 2$.

In addition, as seen in Figure 17, T cannot have a group deadline in the interval $(t', u]$. Therefore, we have the following.

$$pred(V, u) \leq pred(T, u + 1) \leq t' \tag{31}$$

$$(pred(V, u) = pred(T, u + 1)) \;\Rightarrow\; (D(T_{i+j'}) = v + 1 \;\wedge\; D(V_k) = v + 1) \tag{32}$$

Because U is a heavy task, by (P7), $|w(U_j)| \leq 3$. Because U_j is scheduled in slot t and $T_i \prec U_j$, U_j is not early-released, *i.e.*, $r(U_j) \leq t$. Further, because $d(U_j) = t' + 1$ (by (15)), it follows that $t + 1 < t' + 1 \leq t + 3$. Hence, t' is either $t + 2$ or $t + 1$. We consider these two subcases next.

Subcase 4.A: $t' = t+2$. In this case, we show that the swapping in Figure 18 is valid. To begin, note that $t' = t + 2$ implies that $T \notin S_{t+1}$ and $U \notin S_{t+1}$. Let $k' = k - j' + 1$. Then, we have the following as depicted in Figure 18.

- $r(V_{k-1}) = u - 2$ (by (28)) and V_{k-1} is scheduled in slot $u - 2$.
- For each l in the range $k' \leq l < k-1$, $w(V_l)$ is a 2-window (since $pred(V, u) \leq t'$ by (31)).
- Each of $V_{k'}$,...,V_{k-1} is scheduled in the first slot of its window (which follows by inducting from right to left, starting with V_{k-1}). In particular, $V_{k'}$ is scheduled at $t' = r(V_{k'})$.

Before continuing, we note that all of the subtasks $V_{k'-1}, \ldots, V_{k-1}$ belong to τ. To see why, note that their windows fit in the interval $[0, t'+j')$ (see Figure 18) and therefore, by Lemma 11, they are in τ.

Because $V_{k'}$ is released at $t' = t+2$, by part (a) of Lemma 9, $d(V_{k'-1})$ is either $t + 2$ or $t + 3$. We now prove that $d(V_{k'-1}) \neq t + 2$.

Claim 16 $d(V_{k'-1}) \neq t + 2$.

Proof. Assume, to the contrary, that $d(V_{k'-1}) = t + 2$. Because $r(V_{k'}) = t + 2$ (refer to Figure 18 and the properties stated above), this implies that $b(V_{k'-1}) = 0$. Thus, by the definition of a group deadline,

$$pred(V, u) = t + 2. \tag{33}$$

Because $pred(V, u)$ corresponds to $d(V_{k'-1})$, and $b(V_{k'-1})$ is 0, it follows that $pred(V, u)$ is the last group deadline within some job of V. Therefore, by (P10), the difference between u and $pred(V, u)$ is at least the difference between any pair of consecutive group deadlines of V. In particular, we have $succ(V, u) - u \leq u - pred(V, u)$. In either case, by (33), $succ(V, u) \leq 2u - t - 2$. Because $succ(V, u) = D(V_k)$, we have

$$D(V_k) \leq 2u - t - 2. \tag{34}$$

By (31), $pred(T, u+1) \leq t'$, *i.e.*, $pred(T, u+1) \leq t+2$. By (29), $D(T_{i+j'}) \geq 2u - pred(T, u+1) + 1$, which implies that $D(T_{i+j'}) \geq 2u - t - 1$. By (27), $v+1 \geq D(T_{i+j'})$, and hence $v+1 \geq 2u-t-1$. By (26), we have $D(V_k) \geq v+1$. Thus, $D(V_k) \geq 2u - t - 1$, which contradicts (34). Therefore, we conclude that $d(V_{k'-1})$ cannot be $t + 2$. $\square$

Thus, we have the following.

$$d(V_{k'-1}) = t + 3$$

By (15), $d(T_i) = t' + 1 = t + 3$. Further, $D(V_{k'-1}) = u < D(T_i)$ (by (25)). Therefore, $T_i \prec V_{k'-1}$.

We now conclude the proof for this subcase by showing that $V_{k'-1}$ is scheduled in slot $t + 1$, which implies that the swapping in Figure 18 is valid. We have established that V is heavy and $d(V_{k'-1}) = t + 3$. Therefore, by (P7), all windows of V are of length at most three, which implies that $r(V_{k'-1}) \geq t$. If $V_{k'-1}$ is not scheduled in slot $t+1$, then it must be scheduled at or before slot t. (Note that $V_{k'}$ is scheduled in slot $t + 2$.) Further, it is not early-released because $T_i \prec V_{k'-1}$ (proved in the preceding paragraph). Therefore, it must be scheduled in slot t and $r(V_{k'-1}) = t$. (In other words, $w(V_{k'-1})$ must be a 3-window.) As seen in Figure 18, $d(V_{k'-1}) = d(U_j)$, $b(V_{k'-1}) = b(U_j) = 1$, and $D(V_{k'-1}) < D(T_i) = D(U_j)$. Thus, $U_j \prec V_{k'-1}$ contradicting our choice of U_j as the lowest-priority subtask scheduled at t. Thus, $V_{k'-1}$ cannot be scheduled in slot t, and must be scheduled in slot $t + 1$.

Subcase 4.B: $t' = t+1$. In this case, we show that one of the swappings in Figures 19–21 is valid. We use the following result.

Claim 17 $pred(V, u) = t + 1$.

Proof. As in Subcase 4.A, we can show the following.

- V_{k-1} has a window of length two or three and is scheduled in the first slot of its window.
- Each of $V_{k'}$, $\ldots$, V_{k-2} has a window of length two and is scheduled in the first slot of its window. (The existence of subtasks $V_{k'}$, $\ldots$, V_{k-1} in τ follows from the same reasoning given earlier in Subcase 4.A.)

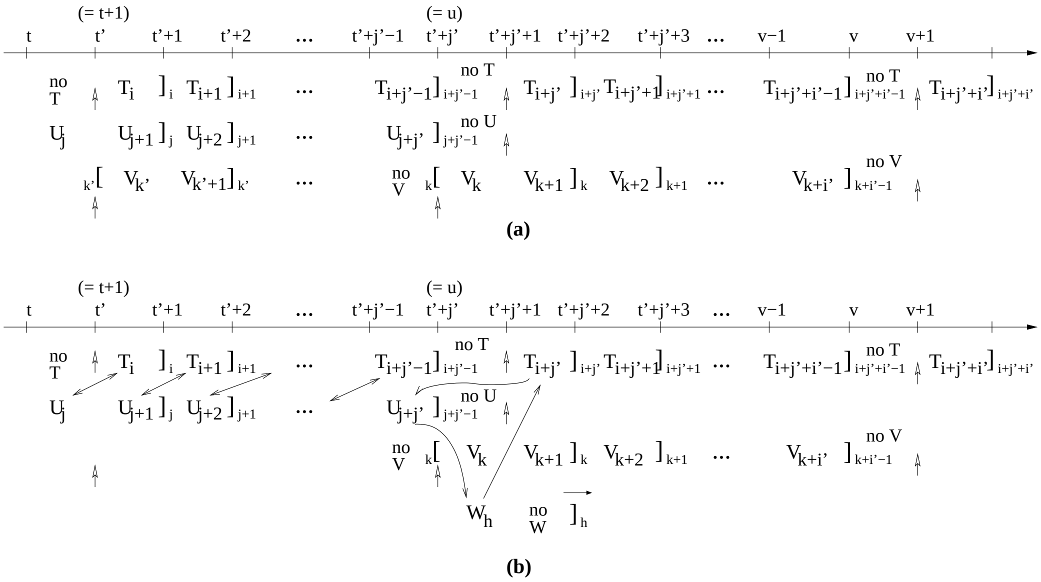

Fig. 19. Subcase 4.B. In each inset of this figure and Figures 20 and 21, $t' = t + 1$, $D(T_i) = D(U_j)$, and $D(V_k) = D(T_{i+j'})$. **(a)** $pred(V, u) = pred(T, u + 1)$. **(b)** $d(W_h) \geq u + 2$.

This is depicted in Figure 19(a). The above facts, along with $t' = t+1$, imply that $r(V_{k'}) = t + 1$. By part (a) of Lemma 9, this implies that $d(V_{k'-1})$ is either $t + 1$ or $t + 2$. If $d(V_{k'-1}) = t + 1$, then $b(V_{k'-1}) = 0$, and hence, $pred(V, u - 1) = t + 1$.

If $d(V_{k'-1}) = t + 2$, then we reason as follows. Because V is heavy, by (P7), $w(V_{k'-1})$ is of length two or three. If $|w(V_{k'-1})| = 3$, then $w(V_{k'-1}) = [t - 1, t + 2)$ and hence, $pred(V, u) = t + 1$. Thus, it suffices to show that $|w(V_{k'-1})| \neq 2$.

Suppose, to the contrary, that $|w(V_{k'-1})| = 2$. (Note that, in this case, $V_{k'-1} \in \tau$ because its window fits in the interval $[0, t+2)$.) Because $d(V_{k'-1}) = t+2$, this implies that $r(V_{k'-1}) = t$. Note that $V_{k'-1}$ cannot have been early-released because $T_i \prec V_{k'-1}$ (this follows from Rule (iii) of the PD^2 priority definition because $D(V_{k'-1}) < D(T_i)$ — see Figure 19(a)). Because $V_{k'}$ is scheduled in slot $t + 1$, $V_{k'-1}$ must be scheduled in slot t. Observe that $d(V_{k'-1}) = t + 2$, $d(U_j) = t' + 1 = t + 2$, $b(V_{k'-1}) = 1$, $b(U_j) = 1$, and $D(V_{k'-1}) < D(U_j)$ (again, refer to Figure 19(a)). Thus, $V_{k'-1}$ has lower priority than U_j at t, which contradicts our choice of U_j as the lowest-priority subtask scheduled at t. This completes the proof of Claim 17. $\square$

Because $t' = t + 1$, by (31) and Claim 17, we have

$$pred(T, u + 1) = pred(V, u).$$

By (32), this implies that $D(T_{i+j'}) = D(V_k) = v + 1$ (see Figure 19(a)).

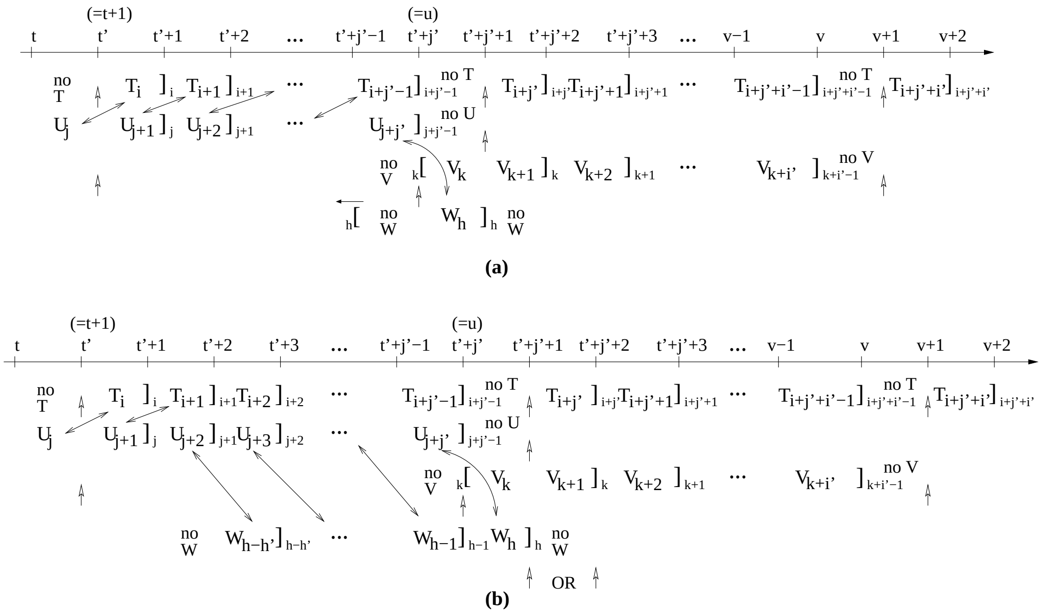

Fig. 20. Subcase 4.B (continued). **(a)** $d(W_h) = u+1$ and $W \notin S_{u-1}$. **(b)** $d(W_h) = u+1$, $W \in S_{u-1}$, and $W \notin S_w$ for some w in $[t', u)$.

Because $T \notin S_u$ and $T \in S_{u+1}$, and because there are no holes in u, there exists a task W that is scheduled at u but not at $u+1$. Let W_h be the subtask of W scheduled at u. If $d(W_h) > u+1$, then the swapping shown in Figure 19(b) is valid. In the rest of the proof, we assume that

$$d(W_h) = u+1.$$

In this case, we show that one of the swappings in Figures 20 and 21 are valid. If $W \notin S_{u-1}$, then the swapping shown in Figure 20(a) is valid. In the rest of the proof, we assume

$$W \in S_{u-1}.$$

In this case, we have $d(W_{h-1}) \geq u$. Because $d(W_h) = u+1$, this implies that $d(W_{h-1}) = u$ and $r(W_h) = u-1$. Thus, $|w(W_h)| = 2$. Thus, by (P6), W is heavy.

We now show that W has a group deadline at time $u+1$ or $u+2$ (refer to Figure 20(b)). Because $d(W_h) = u+1$, by by part (a) of Lemma 9, $r(W_{h+1})$ is either u or $u+1$. If it is $u+1$, then $b(W_h) = 0$, *i.e.*, W has a group deadline at $u+1$. If $r(W_{h+1})$ is u, then $d(W_{h+1})$ is either $u+2$ or $u+3$ (follows by (P7) because W is heavy). Because no subtask of W is scheduled in slot $u+1$, W_{h+1} has to be scheduled in slot $u+2$ and $d(W_{h+1}) = u+3$. This implies that $w(W_{h+1})$ is a 3-window and hence W has a group deadline at $u+2$.

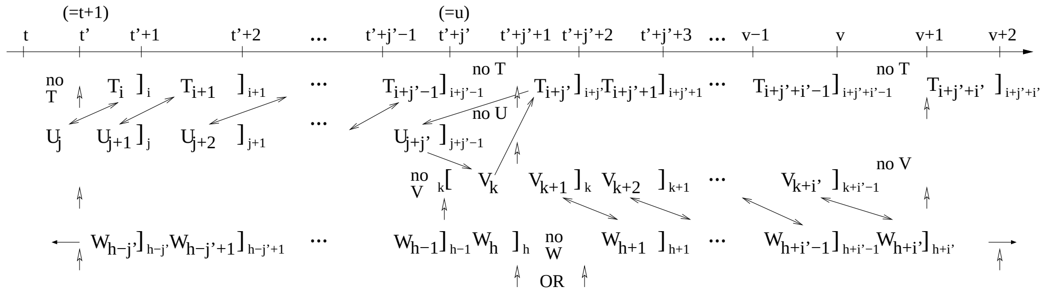

Fig. 21. Subcase 4.B (continued). $d(W_h) = u + 1$, $W \in S_{u-1}$, and W's most recent group deadline before the one at $u + 1$ or $u + 2$ is at or before $t + 1$.

We now look at earlier subtasks of W. If there exists a w such that $t' \leq w \leq u - 1$ and $W \notin S_w$, then a swapping similar to that shown in Figure 20(b) is valid and produces the desired schedule. In the rest of the proof, we assume that for each w in the range $t' \leq w \leq u$, $W \in S_w$. This implies that, at each slot in the interval $[t', u + 1)$, a subtask of W is scheduled in the last slot of its window (recall that W is heavy). This is illustrated in Figure 21. As seen in the figure, each of the subtasks $W_{h-j'+1}, \ldots, W_h$ must have a window of length two. This implies that the most recent group deadline of W before the one at $u + 1$ or $u + 2$ occurs at or before time $t + 1$, *i.e.*,

$$(u + 1 \text{ is a group deadline of } W \Rightarrow \mathit{pred}(W, u + 1) \leq t + 1) \text{ and}$$
$$(u + 2 \text{ is a group deadline of } W \Rightarrow \mathit{pred}(W, u + 2) \leq t + 1). \quad (35)$$

We now show that W's next group deadline after the one at $u+1$ or $u+2$ occurs at or after time $v + 2$, which implies that the swapping shown in Figure 21 is valid.

By (26) and (30), we have $\mathit{pred}(V, u) \leq 2u - v$. Therefore, by Claim 17, we have the following.

$$v \leq 2u - t - 1. \quad (36)$$

There are now two possibilities to consider, depending on whether W has a group deadline at $u + 1$ or $u + 2$.

(1) $u + 1$ is a group deadline of W. In this case, $b(W_h) = 0$. Thus, $u + 1$ *is the last group deadline within some job of* W. Therefore, by (P10), the difference between $u + 1$ and $\mathit{succ}(W, u + 1)$ is at least the difference between any pair of consecutive group deadlines of W. In particular, $\mathit{succ}(W, u+1)-(u+1) \geq u+1-\mathit{pred}(W, u+1)$. Thus, we have $\mathit{succ}(W, u+1) \geq 2u + 2 - \mathit{pred}(W, u + 1)$.

By (35), $\mathit{pred}(W, u + 1) \leq t + 1$. Therefore, $\mathit{succ}(W, u + 1) \geq 2u - t + 1$. By (36), this implies that $\mathit{succ}(W, u + 1) \geq v + 2$.

(2) $u + 2$ is a group deadline of W. In this case, by (P9), the difference between $succ(W, u + 2)$ and $u + 2$ is at least one less than the difference between $u+2$ and $pred(W, u+2)$, *i.e.*, $succ(W, u+2) - (u+2) \geq (u+2) - pred(W, u + 2) - 1$. Therefore, $succ(W, u + 2) \geq 2u - pred(W, u + 2) + 3$.
By (35), $pred(W, u+2) \leq t+1$. Therefore, $succ(W, u+2) \geq 2u - t + 2$. By (36), this implies that $succ(W, u + 2) \geq v + 3$.

This exhausts all the possibilities if T and U are both heavy, and concludes the proof of Lemma 14. $\Box$

By applying Lemma 14 inductively as discussed above, there exists a valid schedule for τ over $[0, t_d)$ consistent with PD^2, contrary to our original assumption. Thus, we have the following theorem.

Theorem 18 PD^2 *generates a valid schedule for any feasible asynchronous task system in which each task's lag is bounded by either* (2) *or* (3).

6 Two-processor Systems

In this section, we prove that the *earliest pseudo-deadline first* (EPDF) algorithm is optimal on two processors. As its name suggests, the EPDF algorithm prioritizes subtask T_i over subtask U_j if it has an earlier deadline, *i.e.*, $d(T_i) \leq d(U_j)$. Any ties are broken arbitrarily.

The basic proof strategy in this section is the same as in the previous section. In particular, we let τ denote a feasible task system that (by assumption) misses a deadline when scheduled on two processors using EPDF. We let t_d be the earliest time at which a deadline is missed in τ (a minimality condition similar to (T2) is not needed here). As before, a 0-compliant schedule exists for τ because it is feasible. By induction, we show that an n-compliant schedule exists for τ, where n is the total number of subtasks with deadlines in $(0, t_d]$. (There is no need here to consider subtasks released in $[0, t_d)$ with deadlines outside of this interval, because (T2) is not being used.) This yields the desired contradiction. The notion of a compliant schedule is slightly different here because it is based on EPDF rather than PD^2. That is, in ranking subtasks, T_i is ordered before U_j, denoted $T_i \trianglelefteq U_j$ if and only if $d(T_i) \leq d(U_j)$ (no tie-breaking information is considered). We define the *rank* of subtask T_i to be its position in the total order $\triangleleft$ (which is obtained from the partial order $\trianglelefteq$).

We now state and prove several lemmas. In the first of these lemmas, an interval of time slots $[t, u)$ is considered, and a set of conditions is stated that is sufficient to conclude that some subtask scheduled in $[t, u)$ can be shifted left or right out of this interval.

Lemma 19 *Let S be a two-processor schedule for $\{T_i \mid T \in \tau \ \wedge \ d(T_i) \leq t_d\}$ that is valid over $[t, u)$, where $t < u - 1$. Suppose that there are no holes in $[t, u)$ and that there exists a subtask U_j scheduled before t such that $d(U_j) \geq t + 1$. Then, there exists a subtask T_i scheduled in $[t, u)$ such that $r(T_i) < t$ or $d(T_i) > u$.*

Proof. Let A be the set of all subtasks scheduled by S in $[t, u)$. Suppose, to the contrary, that

$$(\forall T_i : T_i \in A :: r(T_i) \geq t \ \wedge \ d(T_i) \leq u).$$

We derive a contradiction by showing that total utilization exceeds two, which contradicts the fact that τ is feasible. Let V be a task with subtasks in A. Let $V.n$ denote the number of such subtasks in A, and let V_k (respectively, V_l) be the first (respectively, last) subtask of V scheduled in $[t, u)$. Then, $V.n = l - k + 1$. Because V_k and V_l are in A, $r(V_k) \geq t$ and $d(V_l) \leq u$. Thus,

$$d(V_l) - r(V_k) \leq u - t. \tag{37}$$

By (7) and (8), we have $r(V_k) = \left\lfloor \frac{k-1}{wt(V)} \right\rfloor + \Delta(V)$ and $d(V_l) = \left\lceil \frac{l}{wt(V)} \right\rceil + \Delta(V)$. Substituting these expressions in (37), we get $\left\lceil \frac{l}{wt(V)} \right\rceil + \Delta(V) - \left\lfloor \frac{k-1}{wt(V)} \right\rfloor - \Delta(V) \leq u - t$. Simplifying, we obtain

$$\begin{array}{ll} \left\lceil \frac{l}{wt(V)} \right\rceil - \left\lfloor \frac{k-1}{wt(V)} \right\rfloor \leq u - t & \\ \Rightarrow \frac{l}{wt(V)} - \frac{k-1}{wt(V)} \leq u - t & , \left\lceil \frac{l}{wt(V)} \right\rceil \geq \frac{l}{wt(V)} \text{ and } \left\lfloor \frac{(k-1)}{wt(V)} \right\rfloor \leq \frac{(k-1)}{wt(V)}. \\ \Rightarrow \frac{1}{wt(V)} \leq \frac{u-t}{l-k+1} & , \text{ simplifying.} \end{array}$$

Therefore, we have

$$wt(V) \geq \frac{V.n}{u - t}. \tag{38}$$

We now show that this inequality can be strengthened for U, yielding

$$wt(U) > \frac{U.n}{u - t}. \tag{39}$$

If $U.n = 0$, then the above expression clearly holds, so assume that $U.n \neq 0$. Then, by the statement of the lemma, U_{j+1} must be the first subtask of U scheduled in $[t, u)$. Let U_h be the last such subtask. Then,

$$d(U_h) - r(U_{j+1}) \leq u - t. \tag{40}$$

Because $d(U_j) \geq t+1$, by part (a) of Lemma 9, we have $r(U_{j+1}) \geq t$. If $r(U_{j+1}) > t$, then $d(U_h) - r(U_{j+1}) < u - t$. By reasoning as above (refer to (37) and (38)), this implies that $wt(U) > \frac{U.n}{u-t}$.

On the other hand, if $r(U_{j+1}) = t$, then $d(U_j) = t+1$, which implies that $b(U_j) = 1$. By (7) and (8), we have $d(U_h) = \left\lceil \frac{h}{wt(U)} \right\rceil + \Delta(U)$ and $r(U_{j+1}) = \left\lfloor \frac{(j+1)-1}{wt(U)} \right\rfloor + \Delta(U)$. Substituting these expressions into (40), we obtain

$$\left\lceil \frac{h}{wt(U)} \right\rceil + \Delta(U) - \left\lfloor \frac{j}{wt(U)} \right\rfloor - \Delta(U) \leq u - t.$$

Because $b(U_j) = 1$, $\left\lfloor \frac{j}{wt(U)} \right\rfloor < \frac{j}{wt(U)}$ (see (9)). Because $\left\lceil \frac{h}{wt(U)} \right\rceil \geq \frac{h}{wt(U)}$, it follows that

$$\frac{h}{wt(U)} - \frac{j}{wt(U)} < u - t.$$

Therefore, by reasoning that is similar to that prior to Expression (38), we have $wt(U) > \frac{U.n}{u-t}$.

Now, if there exists a task T with no subtask in A, then $T.n = 0$, implying that $wt(T) > \frac{T.n}{u-t}$. Therefore, from (38) and (39), we conclude that

$$\sum_{V \in \tau} wt(V) \; > \; \sum_{V \in \tau} \frac{V.n}{u-t}.$$

Because there are a total of $(u-t)$ slots in $[t, u)$ and two subtasks are scheduled in each slot, we have $\sum_{V \in \tau} V.n = 2(u-t)$. Therefore,

$$\sum_{V \in \tau} wt(V) > 2.$$

This contradicts the fact that the total utilization of τ is at most 2 (follows from the feasibility of τ). Hence, there exists a subtask T_i scheduled in $[t, u)$ such that either $r(T_i) < t$ or $d(T_i) > u$. $\square$

The above lemma actually holds for any number of processors. In contrast, Lemma 20, given next, is valid only for two-processor systems. This lemma gives a set of conditions under which a subtask U_j can be right-shifted to a later slot. (Recall that finding a valid way to right-shift U_j in Figure 10(b) was the key problem to address in Section 5.) The conditions of the lemma are illustrated in Figure 22(a).

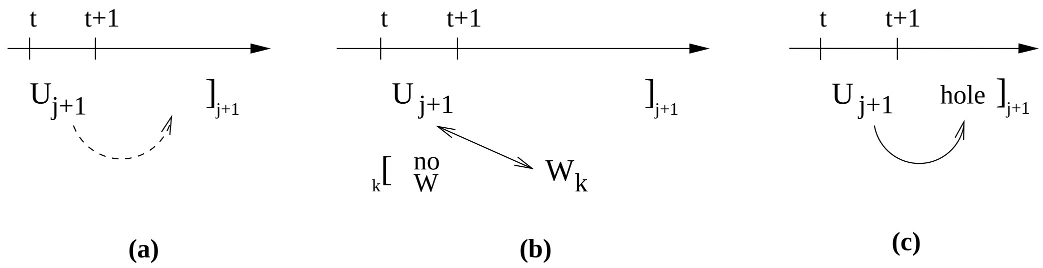

Fig. 22. Lemma 20. **(a)** Conditions of the lemma. **(b)** Base case: there exists W_k scheduled after t such that $r(W_k) \leq t$. **(c)** Base case: no such W_k exists.

Lemma 20 *Let $\tau' = \{T_i \mid T \in \tau \ \wedge \ d(T_i) \leq t_d\}$ and let S be a two-processor schedule for τ'. Let $t \leq t_d$. Assume the following:*

- *subtask U_j is scheduled at slot t, and $d(U_j) > t + 1$;*
- *either no other subtask is scheduled at slot t or U_{j-1} is scheduled there (in which case S is not valid at t);*
- *each subtask scheduled at or after t is scheduled in its Pfair window and S is valid for all $v > t$.*

Then, there exists a schedule S' for τ' such that

- *U_j is scheduled at some slot after t;*
- *$S_v = S'_v$ for all $v < t$;*
- *each subtask scheduled at or after t is scheduled in its Pfair window and S is valid for all $v \geq t$.*

Proof. We prove the lemma by inducting over the rank of U_j. From the statement of the lemma, we have

$$d(U_j) \geq t + 2. \tag{41}$$

Base case: *U_j is the lowest-ranked subtask scheduled in S.* If no subtasks are scheduled after slot t in S, then by (41), we can clearly shift U_j to slot $t + 1$. In the rest of the proof for the base case, we assume that there exist subtasks that are scheduled after slot t.

Suppose that there exists a subtask W_k scheduled after t such that $r(W_k) \leq t$. By the statement of the lemma, W_{k-1} is not scheduled in slot t. Thus, we can swap W_k with U_j to get the desired schedule, as shown in Figure 22(b). (Note that U_{j+1} does not exist in S because U_j is of lowest rank. Thus, swapping W_k with U_j will not create a schedule in which two subtasks of U are scheduled in the same slot.)

The remaining possibility is that, for each subtask W_k scheduled after t, $r(W_k) > t$ and $d(W_k) \leq d(U_j)$ (the latter follows because U_j is of lowest rank). If there are no holes in any slot in $[t+1, d(U_j))$, then we have a contradiction of Lemma 19. Therefore, there exists a slot in $[t+1, d(U_j))$ such that there is a hole in that slot. This implies that we can schedule U_j in that slot (see Figure 22(c)). Thus, a valid schedule exists in which U_j is scheduled at a slot later than t.

Induction step: *U_j is not the lowest-ranked subtask scheduled in S.* Assume that the lemma holds for all subtasks with lower rank than U_j. We consider two cases.

Case 1: *For each subtask W_k scheduled after t, $r(W_k) > t$.* Let $[t+1, u)$ be the smallest interval such that for each subtask V_h scheduled in $[t+1, u)$, $r(V_h) \geq t+1$ and $d(V_h) \leq u$ (see Figure 23(a)). Note that such a u exists, because S includes only a finite collection of subtasks. If there are no holes in this interval, then we have a contradiction of Lemma 19. Therefore, there exists a slot u' in $[t+1, u)$ at which there is a hole. Without loss of generality, let u' be the earliest such slot in $[t+1, u)$. Either $u' = t+1$ or $u' > t+1$.

Subcase 1.A: $u' = t+1$. By (41), U_j can be shifted to slot u'. Unfortunately, if U_{j+1} is scheduled at u', then this might result in a schedule in which U_j and U_{j+1} are scheduled in the same slot. Recall from the statement of the lemma that the schedule is allowed to be invalid at t. Thus, we can first shift U_j to u' and then apply the the induction hypothesis to move U_{j+1} to a later slot (refer to Figure 23(b)). This will result in a schedule that is valid for all $v \geq t$.

Subcase 1.B: $u' > t+1$. If for all subtasks V_h scheduled in $[t+1, u')$, $d(V_h) \leq u'$, then we have a contradiction of the fact that $[t+1, u)$ was the smallest such interval. Therefore, there exists a subtask V_h such that $d(V_h) \geq u'+1$ (refer to Figure 23(c)). Observe that we can move V_h to slot u' to get a schedule in which a slot with a hole occurs earlier. Unfortunately, this movement of V_h is not valid if V_{h+1} is scheduled at u'. However, as before, we can apply the induction hypothesis to move V_{h+1} to a later slot.

By the reasoning above, if Subcase 1.B applies, then it is always possible to get a schedule in which either U_j is shifted to a later slot, or in which the first slot in $[t+1, u)$ that has a hole occurs earlier. If we repeatedly apply Subcase 1.B without shifting U_j to a later slot, then Subcase 1.A will eventually apply, in which case U_j can be shifted as desired.

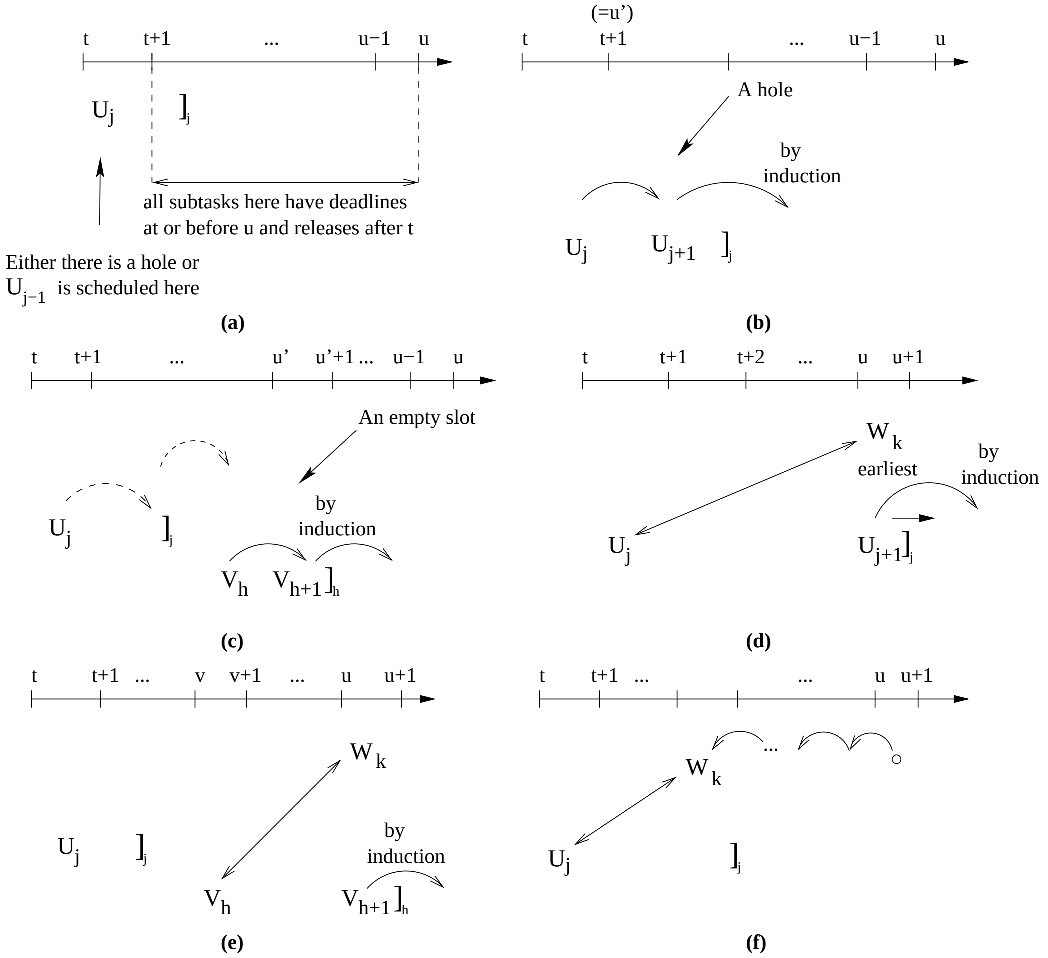

Fig. 23. Cases of Lemma 20. In each inset, $d(U_j) > t+1$. **(a)** Case 1: Smallest interval $[t+1, u)$ such that all subtasks have deadlines and releases inside the interval. **(b)** Subcase 1.A: $u' = t+1$. **(c)** Subcase 1.B: $u' > t+1$. **(d)** Subcase 2.A: $d(U_j) \geq u+1$, and W_k is the earliest subtask with $r(W_k) \leq t$. **(e)** Subcase 2.B: $d(U_j) < u+1$. **(f)** Repeated application of Subcase 2.B to finally apply Subcase 2.A.

Case 2: *There exists a subtask W_k scheduled after t such that $r(W_k) \leq t$.* Without loss of generality, assume that W_k is the earliest-scheduled subtask among all such subtasks. Let W_k be scheduled at slot u. If there are any holes in $[t+1, u)$, then W_k could be moved to the first such slot u', and the reasoning below applies with the smaller interval $[t+1, u')$. Thus, we can assume without loss of generality that there are no holes in $[t+1, u)$.

Subcase 2.A: *$d(U_j) \geq u+1$.* Refer to Figure 23(d). Because $d(U_j) \geq u+1$, we can swap U_j and W_k directly. Unfortunately, U_{j+1} might be scheduled at slot u, in which case the resulting schedule is not valid. However, as before (refer to Case 1), we can apply the induction hypothesis to move U_{j+1} to a later slot. The resulting schedule will be valid for all $v \geq t$.

Subcase 2.B: $d(U_j) < u + 1$. In this subcase, we try to identify subtasks that can be used as an intermediate for swapping. Because W_k is the earliest scheduled subtask after t such that $r(W_k) \leq t$, for each subtask V_h scheduled in $[t + 1, u)$, we have $r(V_h) \geq t + 1$. Hence, because there are no holes in $[t + 1, u)$, by Lemma 19, there exists a subtask V_h scheduled in $[t + 1, u)$ for which $d(V_h) \geq u + 1$. Let V_h be scheduled at time $v \in \{t + 1, \ldots, u - 1\}$ (see Figure 23(e)). We can swap V_h and W_k to get a schedule in which W_k is scheduled earlier (*i.e.*, nearer to slot t). This swapping is valid only if W_{k-1} is not scheduled at v and V_{h+1} is not scheduled at u. Because $r(W_k) < t+1 \leq v$, $d(W_{k-1}) \leq v$ (by part (a) of Lemma 9). Therefore, W_{k-1} cannot be scheduled at slot v. On the other hand, V_{h+1} *can* be scheduled at time u. However, by the induction hypothesis, V_{h+1} can be moved to a later slot.

By repeatedly applying Subcase 2.b, we obtain either the required schedule or a schedule in which Subcase 2.A can be applied. This is illustrated in Figure 23(f). □

The following lemma gives the inductive step of our proof.

Lemma 21 *Let S be a k-compliant two-processor schedule for $\{T_i \mid T \in \tau \ \wedge\ d(T_i) \leq t_d\}$. Then, there exists a $(k + 1)$-compliant two-processor schedule S' for $\{T_i \mid T \in \tau \ \wedge\ d(T_i) \leq t_d\}$.*

Proof. Let T_i be the $(k + 1)^{st}$ subtask according to $\lhd$. If T_i is scheduled in accordance with EPDF, then take S' to be S. Otherwise, we have the following: there exists a time slot t such that T_i is eligible at t, some subtask ranked lower than T_i according to $\lhd$ is scheduled at t, and T_i is scheduled at a slot later than t. Without loss of generality, let t be the earliest such slot and U_j be the lowest-ranked subtask scheduled at t. Let t' be the slot where T_i is scheduled.

As in the proof of Theorem 18, a $(k + 1)$-compliant schedule can be obtained by swapping T_i and U_j, and also as before, the difficult case to consider is as depicted in Figure 10(b). In this case, Lemma 20 can be applied to right-shift U_{j+1} out of slot t'. Once U_{j+1} has been shifted, T_i and U_j can be safely swapped. □

Theorem 22 *EPDF optimally schedules asynchronous, periodic tasks on systems of one or two processors.*

Proof. Establishing the optimality of EPDF for one-processor systems is straightforward, so we consider only two-processor systems. Suppose to the contrary, that EPDF is not optimal for such systems. Let τ and t_d be as defined at the beginning of this section. Then, the set of subtasks $\{T_i \mid T \in \tau \ \wedge\ d(T_i) \leq t_d\}$ misses a deadline at t_d using EPDF. Because τ is feasible, there exists a valid schedule for this set of subtasks such that each subtask is scheduled in its Pfair window. Starting with this schedule, we can apply Lemma 21 inductively (as discussed earlier) to get a valid schedule for $\{T_i \mid T \in \tau \ \wedge \ d(T_i) \leq t_d\}$ under EPDF. Contradiction. $\Box$

7 Concluding Remarks

We have introduced a "work-conserving" variant of Pfair scheduling called ERfair scheduling, and have presented a new algorithm called PD^2, which is the most efficient Pfair/ERfair scheduling algorithm known to date. We have shown that PD^2 is optimal for scheduling any mix of early-release and non-early-release asynchronous, periodic tasks on a multiprocessor. This is the first work known to us on the problem of scheduling both early-release and non-early-release tasks under a common framework. This is also the first paper to show that a variant of the PD Pfair algorithm is optimal for scheduling asynchronous task systems on a multiprocessor. Our counterexamples show that, in general, it is highly unlikely that an optimal Pfair or ERfair scheduling algorithm that is more efficient than PD^2 can be obtained. However, for the special case of a two-processor system, we have shown that a simpler algorithm, namely EPDF, is optimal.

A Proof of Properties (P1) through (P10)

In this appendix, we prove properties (P1)-(P10), all of which pertain to just a single task. As in Section 5.1, for brevity, we let T denote this task, and abbreviate $T.e$ and $T.p$ as e and p, respectively. Thus, $wt(T) = \frac{e}{p}$, and by (7) and (8), we have the following.

$$|w(T_i)| = \left\lceil \frac{ip}{e} \right\rceil - \left\lfloor \frac{(i-1)p}{e} \right\rfloor. \tag{A.1}$$

As noted in Section 5, we assume that e and p are relatively prime, *i.e.*, $gcd(e,p) = 1$.

Lemma 23 *The following properties hold for any task T.*

(a) *$r(T_{i+1})$ is either $d(T_i)$ or $d(T_i)-1$, which implies that $r(T_{i+1}) \geq d(T_i)-1$.*
(b) *The sequence of windows within any two jobs are identical,* i.e., *$|w(T_{ke+i})| = |w(T_i)|$, where $1 \leq i \leq e$ and $k \geq 0$.*
(c) *The windows are symmetric within each job,* i.e., *$|w(T_{ke+i})| = |w(T_{ke+e+1-i})|$, where $1 \leq i \leq e$, and $k \geq 0$.*
(d) *The length of each window is either $\left\lceil \frac{p}{e} \right\rceil$ or $\left\lceil \frac{p}{e} \right\rceil + 1$.*

(e) $|w(T_i)| = \left\lceil \frac{p}{e} \right\rceil$ *if* $(i-1)$ *is a multiple of* e.

Proof. Below, we prove each property separately.

Proof of (a): The required result follows because $r(T_{i+1}) = \Delta(T) + \left\lfloor \frac{i}{wt(T)} \right\rfloor$ (by (7)) and $d(T_i) = \Delta(T) + \left\lceil \frac{i}{wt(T)} \right\rceil$ (by (8)).

Proof of (b): By (A.1), $|w(T_{ke+i})| = \left\lceil \frac{(ke+i)p}{e} \right\rceil - \left\lfloor \frac{(ke+i-1)p}{e} \right\rfloor$. Therefore, $|w(T_{e+i})| = kp + \left\lceil \frac{ip}{e} \right\rceil - kp - \left\lfloor \frac{(i-1)p}{e} \right\rfloor = |w(T_i)|$.

Proof of (c): By part (b), we need to prove this only for the first job of T, *i.e.*, for $k = 0$. By (A.1),

$$\begin{aligned}
|w(T_{e+1-i})| &= \left\lceil \frac{(e+1-i)p}{e} \right\rceil - \left\lfloor \frac{(e-i)p}{e} \right\rfloor \\
&= \left(p + \left\lceil \frac{(1-i)p}{e} \right\rceil\right) - \left(p + \left\lfloor \frac{-ip}{e} \right\rfloor\right) \\
&= \left\lceil \frac{(1-i)p}{e} \right\rceil - \left\lfloor \frac{-ip}{e} \right\rfloor \\
&= \left\lceil \frac{-(i-1)p}{e} \right\rceil + \left\lceil \frac{ip}{e} \right\rceil \\
&= -\left\lfloor \frac{(i-1)p}{e} \right\rfloor + \left\lceil \frac{ip}{e} \right\rceil
\end{aligned}$$

Thus, $|w(T_i)| = |w(T_{e+1-i})|$.

Proof of (d): By (A.1), we have

$$\begin{aligned}
|w(T_i)| &= \left\lceil \frac{ip}{e} \right\rceil - \left\lfloor \frac{(i-1)p}{e} \right\rfloor \\
&= \left\lceil \frac{ip}{e} \right\rceil - \left\lfloor \frac{ip}{e} - \frac{p}{e} \right\rfloor \\
&= \left\lceil \frac{ip}{e} \right\rceil + \left\lceil \frac{p}{e} - \frac{ip}{e} \right\rceil.
\end{aligned}$$

It is easy to see that this last expression equals either $\left\lceil \frac{p}{e} \right\rceil$ or $\left\lceil \frac{p}{e} \right\rceil + 1$.

Proof of (e): By (A.1), $|w(T_1)| = \left\lceil \frac{p}{e} \right\rceil$. By part (b), $|w(T_{ke+1})| = |w(T_1)|$. Thus, the required result follows. □

The next property that we prove is used to prove property (P2) below. It refers to a "minimal" window of a task. As noted earlier, by part (d) of Lemma 9, the windows of any task are of at most two different lengths. We refer to a window of task T with length $\left\lceil \frac{T.p}{T.e} \right\rceil$ as a *minimal* window of T. The following property follows by part (e) of Lemma 9.

(P0) The first window of each job of T is a minimal window of T.

By (B), the b-bit of the last subtask of a job is zero. Therefore, the following property implies that the last window of each job of T is a minimal window of T.

(P1) If $b(T_i) = 0$, then $|w(T_i)| = |w(T_{i+1})|$.

Proof. By (9), $b(T_i) = 0$ implies that $\frac{ip}{e}$ is an integer. Thus, because $gcd(e, p) = 1$, i is a multiple of e. In other words, $i = (k + 1)e$ for some $k \geq 0$, and $|w(T_i)| = |w(T_{ke+e})|$. Therefore, by part (c) of Lemma 9, we have $|w(T_i)| = |w(T_{ke+1})|$. By part (b) of Lemma 9, $|w(T_{ke+1})| = |w(T_{ke+e+1})|$. Therefore, $|w(T_i)| = |w(T_{i+1})|$, as required. □

(P2) If $b(T_i) = 0$, then $w(T_i)$ is a minimal window of T.

Proof. As in the proof of (P1), we can show that i is a multiple of e. Therefore, by (P0), $w(T_{i+1})$ is a minimal window. The required result then follows from (P1). □

(P3) and (P4) below follow directly from part (d) of Lemma 9.

(P3) For all i and j, $|w(T_j)| \leq |w(T_i)| + 1$.

(P4) For all i and j, $|w(T_j)| \geq |w(T_i)| - 1$.

(P5) If T is light, then all of its windows are of length at least three.

Proof. If T is light, then $\frac{e}{p} < \frac{1}{2}$. Therefore, $\frac{p}{e} > 2$, and $\left\lceil \frac{p}{e} \right\rceil \geq 3$. The required result follows because $|w(T_i)| \geq \left\lceil \frac{p}{e} \right\rceil$ (by part (d) of Lemma 9). □

(P6) T has a 2-window if and only if it is heavy.

Proof. By part (e) of Lemma 9, $|w(T_1)| = \left\lceil \frac{p}{e} \right\rceil$. Note that $\left\lceil \frac{p}{e} \right\rceil$ is 2 if and only if $\frac{1}{2} \leq \frac{e}{p} < 1$, which implies that T is heavy. $\Box$

(P7) below follows directly from (P6) and part (d) of Lemma 9.

(P7) If T is heavy, then all its windows are of length two or three.

The following property shows that the last subtask of each job of a heavy task has a 2-window.

(P8) If T is heavy and $b(T_i) = 0$, then $|w(T_i)| = 2$.

Proof. By (P2), $|w(T_i)| = \left\lceil \frac{p}{e} \right\rceil$. Reasoning as in the proof of (P6), it follows that $|w(T_i)| = 2$. $\Box$

(P9) If t and t' are successive group deadlines of a heavy task T, then $t' - t$ is either $\left\lceil \frac{1}{1-wt(T)} \right\rceil$ or $\left\lceil \frac{1}{1-wt(T)} \right\rceil - 1$.

Proof. As shown by Baruah *et al.* [7], the group deadlines of T correspond to subtask deadlines of a task U such that $wt(U) = 1 - wt(T)$. Therefore, by (8), $t' - t = \left\lceil \frac{j+1}{wt(U)} \right\rceil - \left\lceil \frac{j}{wt(U)} \right\rceil$ for some j. Thus, $t' - t = \left\lceil \frac{j}{wt(U)} + \frac{1}{wt(U)} \right\rceil - \left\lceil \frac{j}{wt(U)} \right\rceil$. From this, the required result follows. $\Box$

The following claim is used to property (P10) below. (Refer to Figure A.1.)

Claim 24 *Let T be a heavy task with more than one group deadline per job. Let t and t' be consecutive group deadlines of T, where t' is the first group deadline within some job of T (for the first job of T, take t to be 0). Similarly, let u and u' be consecutive group deadlines of T, where u' is the last group deadline within some job of T. Then, $t' - t = u' - u + 1$.*

Proof. Recall that the group deadlines of T correspond to subtask deadlines of a task U such that $U.e = p - e$ and $U.p = p$. Note that since t' is the first group deadline within some job of T, t is the last group deadline within the previous job, *i.e.*, t corresponds to the deadline of a subtask U_i such that $b(U_i) = 0$. By (9), $b(U_i) = 0$ implies that $\dfrac{ip}{p-e}$ is an integer. Note that because $gcd(e, p) = 1$, we have $gcd(p, p - e) = 1$. Therefore, i is a multiple of $p - e$. In other words, $i = j(p - e)$ for some $j \geq 0$. (This also takes care of the case when $t = 0$.) Therefore, $d(U_i) = jp$, *i.e.*, $t = jp$.

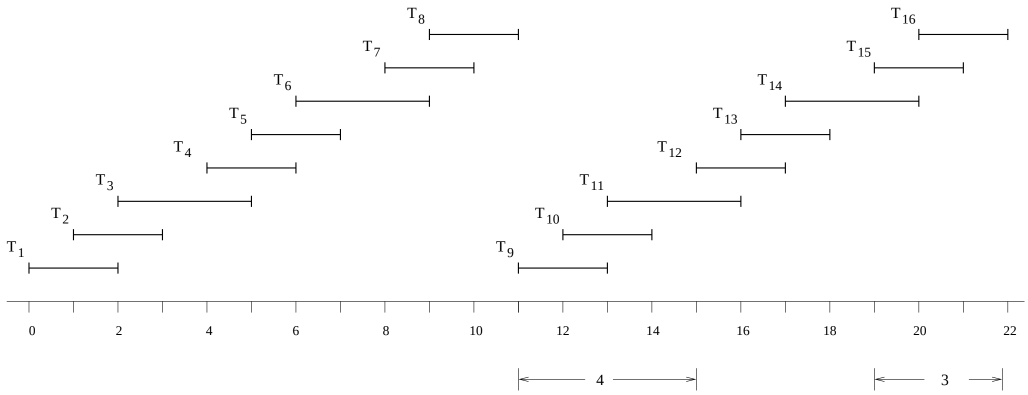

Fig. A.1. Windows of a task T of weight 8/11 are shown. Sample values of t', t, u, and u' (from Claim 24) are 11, 15, 19, 22.

Further, because T has more than one group deadline per job, the number of subtask deadlines in each job of U is at least 2, *i.e.*, $U.e \geq 2$. Therefore, $i+1$ $(= j(p-e)+1)$ is not a multiple of $p-e$. Hence, $\left\lceil \frac{(j(p-e)+1)p}{p-e} \right\rceil = 1 + \left\lfloor \frac{(j(p-e)+1)p}{p-e} \right\rfloor$, which implies the following.

$$\left\lceil \frac{p}{p-e} \right\rceil = 1 + \left\lfloor \frac{p}{p-e} \right\rfloor \tag{A.2}$$

Also, we have $t' - t = \left\lceil \frac{(j(p-e)+1)p}{p-e} \right\rceil - jp = \left\lceil \frac{p}{p-e} \right\rceil$.

Similarly, we can show that $u' = kp$ for some k and $u = \left\lceil \frac{(k(p-e)-1)p}{p-e} \right\rceil = kp + \left\lceil \frac{-p}{p-e} \right\rceil$. Therefore, $u' - u = -\left\lceil \frac{-p}{p-e} \right\rceil = \left\lfloor \frac{p}{p-e} \right\rfloor$.

Thus, $(t'-t) - (u'-u) = \left\lceil \frac{p}{p-e} \right\rceil - \left\lfloor \frac{p}{p-e} \right\rfloor$, which is 1 (by (A.2)). $\square$

(P10) Let T be a heavy task. Let t and t' be consecutive group deadlines of T, where t is the last group deadline within some job of T (for the first job of T, take t to be 0). Then $t' - t$ is at least the difference between any pair of consecutive group deadlines of T.

Proof. If T has just one group deadline per job, then the difference between any two consecutive group deadlines exactly equals the period of T. On the other hand, if T has multiple group deadlines within a job, then by Claim 24, $t' - t = u' - u + 1$, where u' and u are two consecutive deadlines of T. The required result then follows by (P9), because the difference between consecutive group deadlines can have at most two distinct values. $\square$

B Feasibility Proof

We now prove that the following expression is a feasibility condition for an asynchronous task system τ on M processors.

$$\sum_{T \in \tau} wt(T) \leq M \tag{B.1}$$

Let t_l be an arbitrary time slot. We show that τ has a valid schedule over the time interval $[0, t_l)$ by considering flows in a certain graph $G(\tau, t_l)$. By examining the windows of all the subtasks that have deadlines in the interval $(0, t_l]$, we construct a flow graph $G(\tau, t_l)$ such that a maximal flow f in $G(\tau, t_l)$ corresponds to a valid schedule for these subtasks. As mentioned in Section 5, we construct a valid schedule in which each subtask is scheduled in its Pfair window.

Definition of $G(\tau, t_l)$. Let $ns(T, t_l)$ denote the number of subtasks of T that have deadlines in the interval $(0, t_l]$, and let $et(T, t_l)$ denote the union of the Pfair windows of the first $ns(T, t_l)$ subtasks of T.

The vertex set V of $G(\tau, t_l)$ is the union of six disjoint sets of vertices $V_0, \ldots, V_5$ and the edge set E is the union of five disjoint sets of weighted edges $E_0, \ldots, E_4$, where E_i is a subset of $V_i \times V_{i+1} \times \mathbf{N}^+$, $0 \leq i \leq 4$. Thus, G is a six-layered graph, with all the edges connecting vertices in adjacent layers. The vertex sets $V_0, \ldots, V_5$ are defined as follows.

$V_0 = \{\text{source}\}$.
$V_1 = \{\langle 1, T\rangle \mid T \in \tau\}$, corresponding to tasks.
$V_2 = \{\langle 2, T, i\rangle \mid T \in \tau,\ 1 \leq i \leq ns(T, t_l)\}$, corresponding to subtasks.
$V_3 = \{\langle 3, T, t\rangle \mid T \in \tau,\ t \in et(T, t_l)\}$, corresponding to subtask windows.
$V_4 = \{\langle 4, t\rangle \mid 0 \leq t \leq t_l\}$, corresponding to time.
$V_5 = \{\text{sink}\}$

The edge sets $E_0, \ldots, E_4$ are defined as follows.

$E_0 = \{(\text{source}, \langle 1, T\rangle, ns(T, t_l)) \mid T \in \tau\}$
$E_1 = \{(\langle 1, T\rangle, \langle 2, T, i\rangle, 1) \mid T \in \tau,\ 1 \leq i \leq ns(T, t_l)\}$
$E_2 = \{(\langle 2, T, i\rangle, \langle 3, T, t\rangle, 1) \mid T \in \tau,\ 1 \leq i \leq ns(T, t_l),\ t \in w(T_i)\}$
$E_3 = \{(\langle 3, T, t\rangle, \langle 4, t\rangle, 1) \mid T \in \tau,\ t \in et(T, t_l)\}$
$E_4 = \{(\langle 4, t\rangle, \text{sink}, M) \mid 0 \leq t \leq t_l\}$

The weight of an edge in E_0 corresponds to the total processing time required by a task in the interval $[0, t_l)$. The edges in E_1 are used to add the restriction that tasks are allocated in terms of subtasks. The edges in E_2, E_3, and E_4 are used to ensure the validity of the resulting schedule.

A flow is called *integral* if and only if flow across each edge is integral. We use the following theorem about integral flows in graphs with integral edge capacities.

Theorem 25 (Ford and Fulkerson [8]) *A graph in which all edge capacities are integral has a integral maximal flow.*

Feasibility proof. The existence of a schedule for an asynchronous task system τ that satisfies Expression (B.1) follows from Lemmas 26 and 27 below.

Lemma 26 *If there exists an integral flow of size $\sum_{T \in \tau} ns(T, t_l)$ in $G(\tau, t_l)$, then there exists a valid schedule for τ over the interval $[0, t_l]$.*

Proof. An integral flow of size $\sum_{T \in \tau} ns(T, t_l)$ implies that the flow out of the source is $\sum_{T \in \tau} ns(T, t_l)$. By definition of E_0, the sum of the capacities of all the outgoing edges from the source is $\sum_{T \in \tau} ns(T, t_l)$. Therefore, all the edges in E_0 carry a flow equal to their capacity. Hence, an edge from the source to $\langle 1, T \rangle \in V_1$ carries a flow equal to $ns(T, t_l)$. Because there are $ns(T, t_l)$ outgoing edges from $\langle 1, T \rangle$, and each edge of E_1 has a capacity of 1, each such edge carries a flow of 1. Therefore, the flow into each vertex in V_2 is 1.

We obtain a schedule for τ by scheduling T_i in slot t if and only if there is a flow of 1 from vertex $\langle 2, T, i \rangle$ to $\langle 3, T, t \rangle$. From the following discussion, it follows that this schedule is valid.

(i) Since each outgoing edge from V_2 has a capacity of 1, and because the flow is integral, this implies that at most one edge, starting from any vertex in V_2, has a non-zero flow. Thus, for each subtask T_i, only one of $\langle 3, T, t \rangle$ has an incoming flow of 1. In other words, a subtask is scheduled at most once. Because there is an edge from $\langle 2, T, i \rangle$ to $\langle 3, T, t \rangle$ only if t lies in T_i's Pfair window, T_i is scheduled in its window.

(ii) Note that, because successive Pfair windows of the same task may overlap by one slot, vertex $\langle 3, T, t \rangle$ can have more than one incoming edge. However, because the edge from $\langle 3, T, t \rangle$ to $\langle 4, t \rangle$ has a capacity of 1, at most one such incoming edge can have a flow of 1. This ensures that multiple subtasks of the same task are not scheduled in the same slot.

(iii) Because each edge in E_4 has a capacity of M, there can be at most M edges in E_3 with a flow of 1 that are incident on the same vertex in V_4. In other words, at most M subtasks are scheduled in a single slot. $\square$

Note that the maximum flow of $G(\tau, t_l)$ is at most $\sum_{T \in \tau} ns(T, t_l)$, because this is the sum of the capacities of all edges coming from the source. We now show that a real-valued flow of such a size exists.

Lemma 27 *$G(\tau, t_l)$ has a real-valued flow of size $\sum_{T \in \tau} ns(T, t_l)$.*

Proof. We use the following flow assignments. These assignments are similar to those given by Baruah *et al.* [6] to establish that Expression (4) is a feasibility condition for synchronous, periodic tasks.

- Each edge (source, $\langle 1, T \rangle, ns(T, t_l)) \in E_0$ carries a flow of size $ns(T, t_l)$.
- Each edge $(\langle 1, T \rangle, \langle 2, T, i \rangle, 1) \in E_1$ carries a flow of 1. Because there are $ns(T, t_l)$ outgoing edges from each $\langle 1, T \rangle$, flow is conserved at all vertices in V_1.
- The flow through the edges in E_2 is defined as follows. Let $f(T_i, t)$ define the flow from $\langle 2, T, i \rangle$ to $\langle 3, T, t \rangle$. Then,

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

  We now show that these assignments ensure that the flow is conserved at every vertex in V_2, *i.e.*, the flow out of each vertex $\langle 2, T, i \rangle \in V_2$ is 1. By (B.2), the total flow out of $\langle 2, T, i \rangle$ is $(d(T_i) - r(T_i) - 2) \times wt(T) + \left(\left\lfloor \frac{i-1}{wt(T)} \right\rfloor + 1\right) \times wt(T) - (i-1) + i - \left(\left\lceil \frac{i}{wt(T)} \right\rceil - 1\right) \times wt(T)$, which simplifies to $1 + wt(T) \times (d(T_i) - r(T_i)) + wt(T) \times \left(\left\lfloor \frac{i-1}{wt(T)} \right\rfloor - \left\lceil \frac{i}{wt(T)} \right\rceil\right)$. By (5) and (6), $d(T_i) - r(T_i) = \left\lceil \frac{i}{wt(T)} \right\rceil - \left\lfloor \frac{i-1}{wt(T)} \right\rfloor$. Thus, the total flow is 1.
- Each edge $(\langle 3, T, t \rangle, \langle 4, t \rangle, 1) \in E_3$ carries a flow equal to the sum of all incoming flows at $\langle 3, T, t \rangle$. We now show that this flow is at most $wt(T)$ (which is at most 1). We first show that $f(T_i, t) \leq wt(T)$. This follows directly from (B.2) if $t \notin \{r(T_i), d(T_i - 1)\}$. If $t = r(T_i)$, then $f(T_i)$ is

$$\begin{aligned} &\left(\left\lfloor \frac{i-1}{wt(T)} \right\rfloor + 1\right) \times wt(T) - (i-1) && \text{, by (B.2)} \\ &\leq \left(\frac{i-1}{wt(T)} + 1\right) \times wt(T) - (i-1) && \text{, } \lfloor x \rfloor \leq x \\ &= wt(T) && \text{, by simplification} \end{aligned}$$

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

$$
\begin{aligned}
& i - \left(\left\lceil \frac{i}{wt(T)} \right\rceil - 1\right) \times wt(T) && \text{, by (B.2)} \\
& \leq i - \left(\frac{i}{wt(T)} - 1\right) \times wt(T) && \text{, } \lceil x \rceil \geq x \Rightarrow -\lceil x \rceil \leq -x \\
& = wt(T) && \text{, by simplification}
\end{aligned}
$$

We now only need to consider the time slot in which two consecutive Pfair windows overlap. That will be the case when $d(T_i) - 1 = r(T_{i+1})$ for some i. In this case, the total flow will be $f(T_i, d(T_i) - 1) + f(T_{i+1}, r(T_{i+1}))$. Thus, the flow 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, $d(T_i) - 1 = r(T_{i+1})$, it follows that $\left\lceil \frac{i}{wt(T)} \right\rceil - 1 = \left\lfloor \frac{i}{wt(T)} \right\rfloor$. Therefore, $\left\lfloor \frac{i}{wt(T)} \right\rfloor - \left\lceil \frac{i}{wt(T)} \right\rceil = -1$. Thus, the total flow is $wt(T)$. Thus, in all cases, the sum of all incoming flows at $\langle 3, T, t \rangle$ is at most $wt(T)$.

- Each edge $(\langle 4, t \rangle, \text{sink}, M) \in E_4$ carries a flow equal to the sum of all incoming flows at $\langle 4, t \rangle$. Thus, the incoming flow into $\langle 4, t \rangle$ from $\langle 3, T, t \rangle$ can be at most $\sum_{T \in \tau} wt(T)$. Because $\sum_{T \in \tau} wt(T) \leq M$, the incoming flow and hence the outgoing flow at $\langle 4, t \rangle$ is at most M.

This proves that the flow along each edge is at most its capacity and that the flow is conserved at all vertices. Hence, the flow defined above is a valid flow. □

We are now in a position to state the following lemma and theorem.

Lemma 28 *An asynchronous task system τ has a valid schedule on M processors in which each subtask is scheduled in its Pfair window if and only if $\sum_{T \in \tau} wt(T) \leq M$.*

Proof. The necessity of the condition follows from the necessity of condition for periodic task systems (proved by Baruah *et al.* [6]) since any periodic task system is also an asynchronous task system. To prove sufficiency, we construct a valid schedule for τ in $[0, t)$ for any given t. Because t is arbitrary, it follows that τ has a valid schedule. By Lemma 27, it follows that $G(\tau, t)$ has a real-valued flow of size $\sum_{T \in \tau} ns(T, t)$. This is a maximal flow, since the sum of the capacities of all the outgoing edges from the source is the same. Because all edge capacities in $G(\tau, t)$ are integers, by Theorem 25, it follows that $G(\tau, t)$ also has a integral flow of size $\sum_{T \in \tau} ns(T, t)$. Therefore, by Lemma 26, τ has a valid schedule over $[0, t)$. By the definition of $G(\tau, t)$, it follows that in this schedule each subtask will be scheduled in its Pfair window. □

Theorem 29 *An asynchronous periodic task system τ has a valid schedule on M processors if and only if $\sum_{T \in \tau} wt(T) \leq M$.*

Proof. Follows directly from Lemma 28. □

Acknowledgements

We are grateful to Sanjoy Baruah, Mark Moir, and Srikanth Ramamurthy for many helpful discussions on the subject of this paper. We also thank the anonymous reviewers for their suggestions.

References

[1] 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.

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

[3] J. Anderson and A. Srinivasan. Mixed Pfair/ERfair scheduling of asynchronous periodic tasks. In *Proceedings of the 13th Euromicro Conference on Real-Time Systems*, pages 76–85, June 2001.

[4] S. Baruah. Fairness in periodic real-time scheduling. In *Proceedings of the 16th IEEE Real-time Systems Symposium*, pages 200–209, December 1995.

[5] S. Baruah, private communication.

[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] L. Ford and D. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[9] M. Moir and S. Ramamurthy. Pfair scheduling of fixed and migrating periodic tasks on multiple resources. In *Proceedings of the 20th IEEE Real-time Systems Symposium*, pages 294–303, December 1999.

[10] S. Ramamurthy and M. Moir. Static-priority periodic scheduling of multiprocessors. In *Proceedings of the 21st IEEE Real-Time Systems Symposium*, pages 69–78, December 2000.

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

[12] 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.