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
E-mail: {anderson,anands}@cs.unc.edu
Phone: (919) 962-1757

December 2000

Abstract

There has been much recent interest in multiprocessor Pfair and ERfair scheduling algorithms. Under Pfair scheduling, each task is broken into quantum-length subtasks, each of which must execute within a "window" of time slots. These windows divide each period of a task into potentially overlapping subintervals of approximately equal length. "Early-release" fair (ERfair) scheduling was recently proposed as a work-conserving variant of Pfair scheduling. Under ERfair scheduling, subtasks within the same job are allowed to execute before their Pfair windows.

In this paper, we prove that a simplified variant of the PD Pfair algorithm, called PD^2, is optimal for scheduling any mix of early-release and non-early-release asynchronous tasks on a multiprocessor. This result breaks new ground in two ways. First, we are the first to consider the problem of scheduling both early-release and non-early-release tasks under a common framework. Second, all prior work on optimal multiprocessor Pfair or ERfair scheduling algorithms has been limited to synchronous periodic task systems.

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

1 Introduction

Pfair scheduling was proposed by Baruah et al. as a way of optimally and efficiently scheduling periodic tasks on a multiprocessor system [4, 5]. Pfair scheduling differs from more conventional real-time scheduling disciplines in that tasks are explicitly required to make progress at steady rates. In the classic periodic task model, each task T executes at an implicit rate given by $T.e/T.p$, where $T.e$ is the *execution cost* of each job (i.e, instance) of T, and $T.p$ is the *period* of T. However, this notion of a rate is 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 last of which is its deadline. These windows divide each period of a task into potentially overlapping subintervals of approximately equal length. 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.

Baruah et al. [4, 5] gave two algorithms, called PF and PD, that generate Pfair schedules on multiprocessors for any feasible synchronous periodic task set. (As explained later, any Pfair schedule is also periodic. In a *synchronous* task system, each task releases its first job at time 0; in an *asynchronous* task system, a task may release its first job at any time.) PD is the more efficient of the two algorithms. In recent work [2], we presented an improved version of PD, called PD^2. PD^2 is the most efficient Pfair algorithm proposed to date for optimally scheduling synchronous periodic tasks.

In other recent work, we considered a work-conserving variant of Pfair scheduling called "early-release" fair (ERfair) scheduling, and showed that an early-release version of PD is an optimal ERfair scheduling algorithm for synchronous periodic task systems [1]. ERfair scheduling differs from Pfair scheduling in a rather simple way. 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. 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.

One limitation of most prior work on Pfair and ERfair scheduling is that only synchronous periodic task systems have been considered.[1] One may think that the optimality of PD and related algorithms for asynchronous task systems follows as a simple corollary from previous work on synchronous task systems. However, most correctness proofs published previously for these algorithms hinge critically on the assumption that all tasks

[1] Moir and Ramamurthy have recently given a static-priority Pfair scheduling algorithm for asynchronous task systems [6]. Such static-priority algorithms are not optimal. In other recent work, we proposed a task model that generalizes the sporadic (and hence asynchronous) task model [3]. However, the algorithms given in [3] are only applicable to two-processor systems.

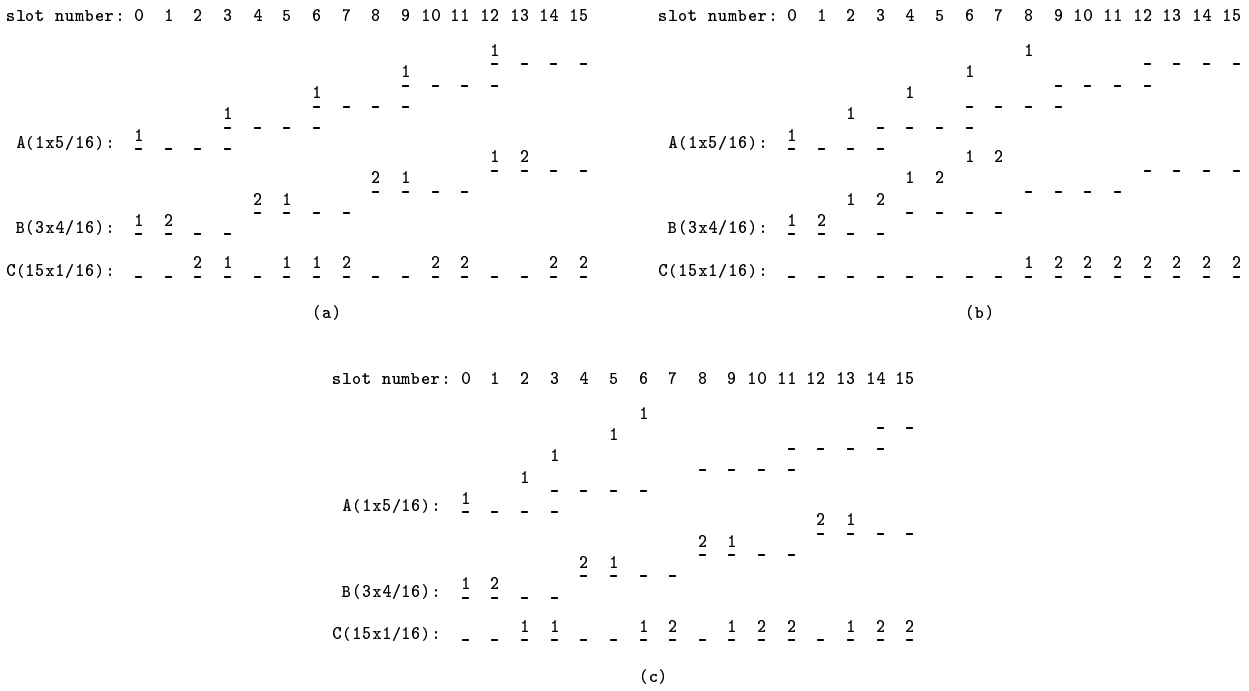


Figure 1: A schedule for a task set under **(a)** Pfair scheduling, **(b)** ERfair scheduling, and **(c)** mixed Pfair and ERfair scheduling. In this figure, tasks of a given weight are shown together. Each Pfair window is shown on a separate line and is depicted by showing the time slots it spans. Each column corresponds to a time slot. In inset (a), an integer value n in slot t of some window means that n of the subtasks that must execute within that window are scheduled in slot t. No integer value means that no such subtask is scheduled in slot t. In insets (b) and (c), some subtasks may be scheduled before their Pfair windows.

are synchronous and periodic. In other words, these proofs are quite brittle and difficult to adapt to other task models.

Contributions of this paper. In this paper, we prove that the PD^2 Pfair algorithm is optimal for scheduling any mix of early-release and non-early-release asynchronous tasks on a multiprocessor. In addition to being the first work on optimal multiprocessor Pfair or ERfair scheduling algorithms for asynchronous systems, this paper is also is the first to consider the problem of scheduling both early-release and non-early-release tasks under a common framework. Such functionality might be useful if a small subset of tasks in a system are subject to stringent average response-time requirements. A task's average response time can be lowered by scheduling it as an early-release task.

Some examples. Before continuing, we consider some example schedules that highlight the differences among the task models considered in this paper. In the Pfair scheduling literature, the ratio of a task 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. 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 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. As we shall see, all of the schedules in Figure 1 are consistent with PD^2.

The rest of this paper is organized as follows. In Section 2, we define Pfair and ERfair scheduling. Then, in Section 3, we define the PD^2 algorithm. In Section 4, we prove that PD^2 correctly schedules any feasible asynchronous periodic task set consisting of a mix of early-release and non-early-release tasks. Concluding remarks appear in Section 5.

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 later.) We assume that processor time in such a system is allocated in discrete time units, or quanta; the time interval $[t, t+1)$, where t is a nonnegative integer, is called *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 $[k \cdot T.p, (k+1) \cdot T.p)$, where $k \geq 0$. 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 $S : \tau \times \mathcal{I} \mapsto \{0, 1\}$, where τ is a set of periodic tasks and $\mathcal{I}$ 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:

$$lag(T, t) = (T.e/T.p)t - allocated(T, t), \tag{1}$$

where $allocated(T, t)$ is the amount of processor time allocated to T in $[0, t)$. A schedule is *Pfair* iff

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

Informally, the allocation error for each task cannot exceed one quantum. The notion of early-release scheduling is obtained by simply dropping the -1 lag constraint. Formally, a schedule is *early-release fair* (*ERfair*) iff

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

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 any ERfair schedule (and hence any Pfair or mixed Pfair/ERfair schedule) is periodic [1, 3, 4].

Feasibility. A synchronous periodic task set τ has a Pfair schedule on M processors iff

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

This result was proved by Baruah et al. [4] by means of a network flow construction. Because any Pfair schedule is ERfair, (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 *predecessor* of T_i and T_{i+1} is called the *successor* of T_i. As in [4], 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 $r(T_i)$ and $d(T_i)$ are as follows.

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

$$d(T_i) = \left\lceil \frac{i}{wt(T)} \right\rceil - 1 \tag{6}$$

(Derivations of these expressions can be found in [2].) $r(T_i)$ is the first slot into which T_i potentially could be scheduled, and $d(T_i)$ 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 $d(T_i) - r(T_i) + 1$. A window

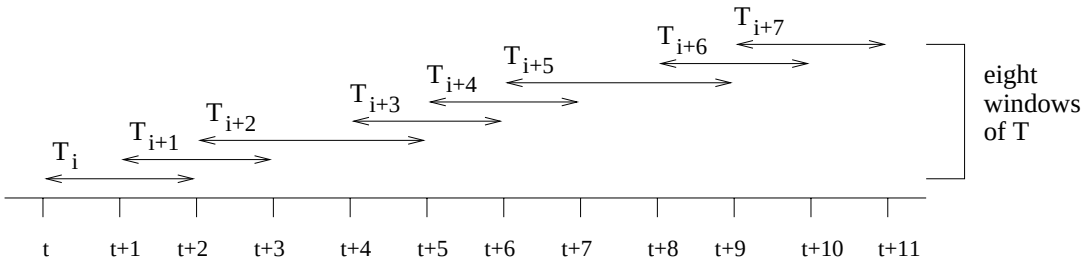


Figure 2: The eight "windows" of a task T with weight $T.e/T.p = 8/11$. Each of T's eight units of computation must be allocated processor time during its window, or else a lag-bound violation will result.

spanning n time slots is called an *n-window*.

Under Pfair scheduling, two tasks T and T' for which $\frac{T.e}{T.p} = \frac{T'.e}{T'.p}$ are 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. We do make this assumption here. Unfortunately, this creates a slight problem, because under ERfair scheduling, two tasks with equal weights but different periods may be scheduled differently, because their job releases occur at different times (under ERfair scheduling, a subtask is eligible as soon as its predecessor executes, unless it is the first subtask of its job). To circumvent this problem, we will henceforth use the term *task instance* to refer to a single release of some task. Each task instance consists of a sequence of one or more jobs.

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. Each job corresponds to one task instance. Using Equations (5) and (6), it is possible to show that the windows within each job of T are as depicted in Figure 2. Under our model, a task T' with weight 16/22 is treated exactly like a task with weight 8/11, except that each task instance of T' consists of two jobs.

The following properties concerning windows, which are proved in [2], are used in our proof.

(P1) The windows of each job of a task T are symmetric, i.e., $|w(T_{ke+i})| = |w(T_{ke+e+1-i})|$, where $e = T.e$, $1 \leq i \leq T.e$, and $k \geq 0$.

(P2) The length of each of task T's windows is either $\left\lceil \frac{T.p}{T.e} \right\rceil$ or $\left\lceil \frac{T.p}{T.e} \right\rceil + 1$. A window of task T with length $\left\lceil \frac{T.p}{T.e} \right\rceil$ (respectively, $\left\lceil \frac{T.p}{T.e} \right\rceil + 1$) is called a *minimal* (respectively, *maximal*) window of T.

(P3) The first window of each job of a task is a minimal window of that task.

(P4) A task has a 2-window iff it is heavy. By (P1) and (P3), the first and last windows of any job a heavy task are both of length two.

Asynchronous Pfair task systems. In the usual definition of an asynchronous periodic task system, each task may release its first task instance at any time. For Pfair- or ERfair-scheduled systems, this means that the first subtask of each task T, namely T_1, may 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 proof. It is straightforward to modify the flow construction in [4] to apply to asynchronous task systems as defined here; thus, (4) is a feasibility condition for such systems. 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. We will not bother to state these formulae here because we do not make explicit use of them in our proof. We instead rely on Properties (P1)-(P4) stated above.

3 PD^2 Algorithm

For synchronous periodic task systems, the most efficient Pfair scheduling algorithm proposed to date is an algorithm called PD^2 [2], which is an improvement of the PD algorithm [5]. PD prioritizes subtasks by pseudo-deadline (hence its name). It is related to an earlier algorithm called PF [4] 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 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-break parameters 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 b-bit. By (5) and (6), $r(T_{i+1})$ 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.

$$b(T_i) = \begin{cases} 1, & \text{if } r(T_{i+1}) = d(T_i) \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

It can be shown that $b(T_i) = 0$ iff $w(T_i)$ is the last window of a job of T. This follows from (2) and the fact that $T.p$ and $T.e$ are relatively prime.

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_i, T_{i+1} or T_{i+2}, T_{i+3}, T_{i+4} or T_{i+5}, T_{i+6} in Figure 2). If any of $T_i, \ldots, T_j$ is scheduled in the last slot of its window, then each subsequent subtask in this sequence must be scheduled in its last slot. In effect, $T_i, \ldots, T_j$ must be considered as a single schedulable entity subject to a "group" deadline. Formally, we define $d(T_j) + 1$ 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 $t+3$, $t+7$, and $t+10$.

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

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

For example, in Figure 2, $D(T_i) = t+3$ and $D(T_{i+5}) = t+10$. The above definition of D is valid only for heavy tasks. If T is light, then $D(T_i) = 0$. Our proof makes use of the following properties concerning group deadlines. These properties are proved in [2].

(P5) If t and t' are successive group deadlines of a task T, then $t' - t$ is either $\left\lfloor \frac{T.p}{T.p - T.e} \right\rfloor$ or $\left\lfloor \frac{T.p}{T.p - T.e} \right\rfloor + 1$.

(P6) Let T be a heavy task with more than one group deadline per job. Let t and t' (respectively, u and u') be successive group deadlines of T, where t' (respectively, u') is the first (respectively, last) group deadline within a complete job J of T (if J is the first job of T, then take t to be one less than J's release time). Then, $t' - t = u' - u + 1$.

Having defined the two tie-break parameters, we can now state the PD^2 priority definition.

PD^2 Priority Definition: Task T's priority at time t is defined to be $(d(T_i), b(T_i), D(T_i))$, where T_i is the currently-enabled subtask of T. Priorities are ordered according to the following relation.

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

At time t, if the current subtasks of T and U are T_i and U_j, respectively, then T's priority is at least U's if $(d(T_i), b(T_i), D(T_i)) \preceq (d(U_j), b(U_j), D(U_j))$. □

All of the components within the PD^2 priority definition can be calculated in an asynchronous system using simple formulae. If a task T is subject to the Pfair lag constraint (2), then each subtask T_i becomes eligible for execution at time $r(T_i)$. 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 task instance, and immediately after the execution of T_{i-1} otherwise.

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 1 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, then their group deadlines are inspected to break the tie. If one is heavy and the other light, then the tie is broken if 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 larger group deadline. Note that the subtask with the larger group deadline can force a longer cascade of scheduling decisions in the future. Thus, choosing to schedule such a subtask early places fewer constraints on the future schedule. If both are heavy and their group deadlines are equal, then the tie can be broken arbitrarily.

4 Correctness Proof

We now prove that PD^2 produces a valid schedule for any feasible asynchronous periodic task system. We begin by assuming, to the contrary, that PD^2 fails to correctly schedule some such task system.

Definition 1 *t_d is defined as the minimal time at which some feasible asynchronous periodic task system misses a deadline under PD^2.*

Let τ be a feasible asynchronous periodic 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.

The following lemma gives an important property of the task set τ (note that the proof of this lemma relies crucially on 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 1 *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 prove that if $b(T_{i-1}) = 0$, then $|w(T_{i-1})| > t$; the proof for $b(T_{i-1}) = 1$ is similar. Suppose 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 $t - |w(T_{i-1})|$. 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 a processor is idle, or it will "push" a lower-priority subtask to a later slot. This "pushed" subtask either misses a deadline or is scheduled correctly, in which case it may "push" yet another subtask to a later slot, and so on. Hence, τ' misses a deadline at t_d or earlier. This contradicts either Definition 1 (the minimality of t_d) or (T2). Therefore, $|w(T_{i-1})| > t$. □

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 priority definition. ($\prec$ is obtained by arbitrarily breaking any ties left by $\preceq$.) Let T_i and U_j be two subtasks in τ. Then, T_i is ordered before U_j, denoted $T_i \lhd U_j$ iff T_i has a higher PD^2 priority than U_j when the issue of eligibility is ignored, i.e., $(d(U_j), b(U_j), D(U_j)) \prec (d(T_i), b(T_i), D(T_i))$. We define a valid schedule S to be *k-compliant* iff **(i)** the first k subtasks according to $\lhd$ are scheduled in accordance with PD^2, and **(ii)** the remaining subtasks are scheduled within their Pfair windows.

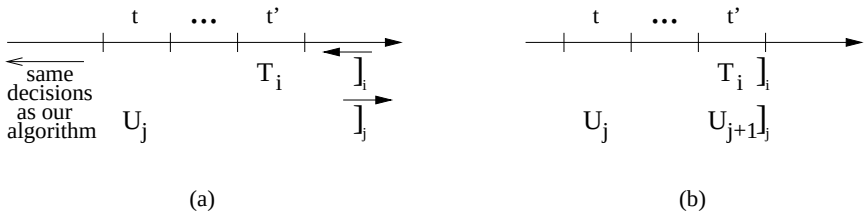


Figure 3: We use the following notation in this and subsequent figures. "[" and "]" indicate the release and deadline of a subtask; subscripts indicate which subtask. Each task is shown on a separate line. An arrow from subtask T_i to subtask U_j indicates that T_i is now scheduled in place of U_j. An 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 don't 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 slot t, then "]" is aligned with the right side of slot t. **(a)** Conditions of Lemma 3. **(b)** The "difficult" case to consider.

We now present two lemmas. The second of these, Lemma 3, allows us inductively prove that τ does not miss a deadline at t_d as originally assumed. Lemma 2, which is proved in an appendix, deals with a situation arising in one of the cases in Lemma 3. According to Lemma 2, if subtasks T_i, U_j, and U_{j+1} are scheduled as shown in Figure 10(a) in the appendix, then by swapping some subtasks, it is possible to move U_j out of slot t.

Lemma 2 *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, U_{j+1} and T_i are scheduled in slot t', and $r(U_j) = t$, $d(U_j) = t'$, $r(U_{j+1}) = t'$, and $d(T_i) = t'$, where $w(U_j)$ 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 $S_t - \{U\} \subset S'_t$.*

We now prove that a k-compliant schedule exists by induction on k. Note that a 0-compliant schedule is just a Pfair schedule, 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.

Lemma 3 *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 $\lhd$. 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)** some processor is idle at t, or **(ii)** some subtask ordered after T_i by $\lhd$ is scheduled at t. In the former case, we can easily schedule T_i at t, so in the rest of the proof, we assume that (ii) holds. Take t to be the earliest such time slot, and let U_j be the lowest-priority subtask scheduled at t. Let t' be the slot where T_i is scheduled, as depicted in Figure 3(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 prove 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 $\lhd$. Since S is k-compliant, all such subtasks are scheduled within their Pfair windows.

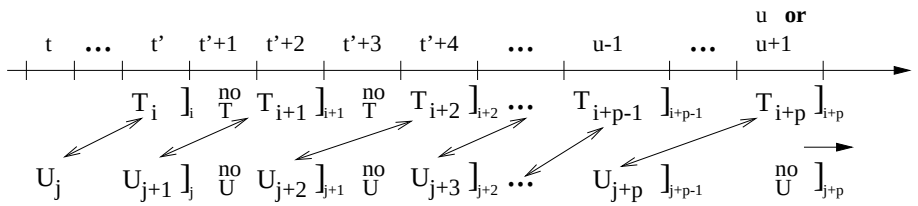


Figure 4: Case 2. T is heavy, U is light, and $d(T_i) = d(U_j)$.

Because S is a valid schedule and T_i's priority is higher than U_j's, we have

$$t < t' \leq d(T_i) \leq d(U_j)$$

If $d(T_i) < d(U_j)$ or if $d(T_i) = d(U_j) \ \wedge \ b(U_j) = 0$, then no subtask of U can be scheduled in the interval $(t, t']$. Hence, T_i and U_j can be directly swapped to get the required schedule. In the rest of the proof, we assume that

$$d(T_i) = d(U_j) \ \wedge \ b(U_j) = 1. \tag{8}$$

Because T has higher priority than U at time t, we have

$$b(T_i) = 1. \tag{9}$$

If U_{j+1} is not scheduled in slot t', then T and U still can be directly swapped to get a valid schedule. Thus, we henceforth assume U_{j+1} is scheduled in slot t', i.e., the same slot as T_i. By (8), this can happen only if $d(T_i)$ and $d(U_j)$ both equal t' (otherwise, $w(U_j)$ and $w(U_{j+1})$ would overlap by more than one slot). Thus, we have

$$U_{j+1} \in S_{t'} \ \wedge \ d(T_i) = t' \ \wedge \ d(U_j) = t'. \tag{10}$$

These conditions are depicted in Figure 3(b). We now consider four cases, which depend on the weights of tasks T and U. We remind the reader that, in all of these cases, (8) through (10) are assumed to hold.

Case 1: T is light and U is heavy. By the definition of D, 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 4 is valid. (The argument hinges on the fact U's windows are at least as long as T's.) By (P2) and (P4), all windows of T are of length two or three, and the last window of each job of T is of length two. By (9), $w(T_i)$ is not the final window of its job. Thus, there exists r such that

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

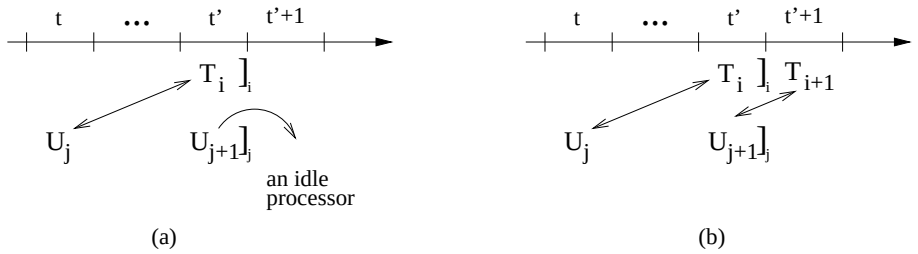


Figure 5: Case 3. **(a)** Some processor is idle in slot $t'+1$. **(b)** T_{i+1} is scheduled in slot $t'+1$.

(Note that r could be one, i.e., T could have no 3-windows between $w(T_i)$ and $w(T_{i+1})$.) Because U is light, by (P4), $|w(U_k)| \geq 3$ 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,

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

Because $d(T_{i+q}) < d(U_{j+q})$, we have $d(T_{i+q}) < r(U_{j+q+1})$. 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:

- $(\forall k : 0 < k < p :: d(T_{i+k}) = d(U_{j+k}) \wedge |w(T_{i+k})| = 3 \wedge |w(U_{j+k})| = 3 \wedge b(T_{i+k}) = 1) \wedge d(T_{i+p}) \leq d(U_{j+p})$,
- 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 4 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 3(b). Because U is light, by (P4), $|w(U_{j+1})| \geq 3$. Therefore, U_{j+2} is released after $t'+1$ and so U is not scheduled in slot $t'+1$. If a processor is idle in slot $t'+1$, then the swapping shown in Figure 5(a) gives the required schedule. If T_{i+1} is scheduled in slot $t'+1$, then the swapping shown in Figure 5(b) gives the required schedule. In the rest of Case 3, we assume that no processor is idle in slot $t'+1$ and T_{i+1} is not scheduled there. We show that one of the swappings in Figure 6 is valid.

Because U is scheduled at t' but not $t'+1$, and because no processor is idle in slot $t'+1$, there exists a task V scheduled in $t'+1$ but not t'. Let V_k be the subtask of V scheduled at $t'+1$. Because V_k is scheduled at time $t'+1$, we have $r(V_k) \leq t'+1$. If $r(V_k) < t'+1$, then the swapping shown in Figure 6(a) produces the desired schedule. In the rest of the proof for Case 3, we assume

$$r(V_k) = t'+1, \tag{11}$$

in which case this swapping is not valid. Now consider subtask V_{k-1}. If $V_{k-1} \notin \tau$, then by Lemma 1, either

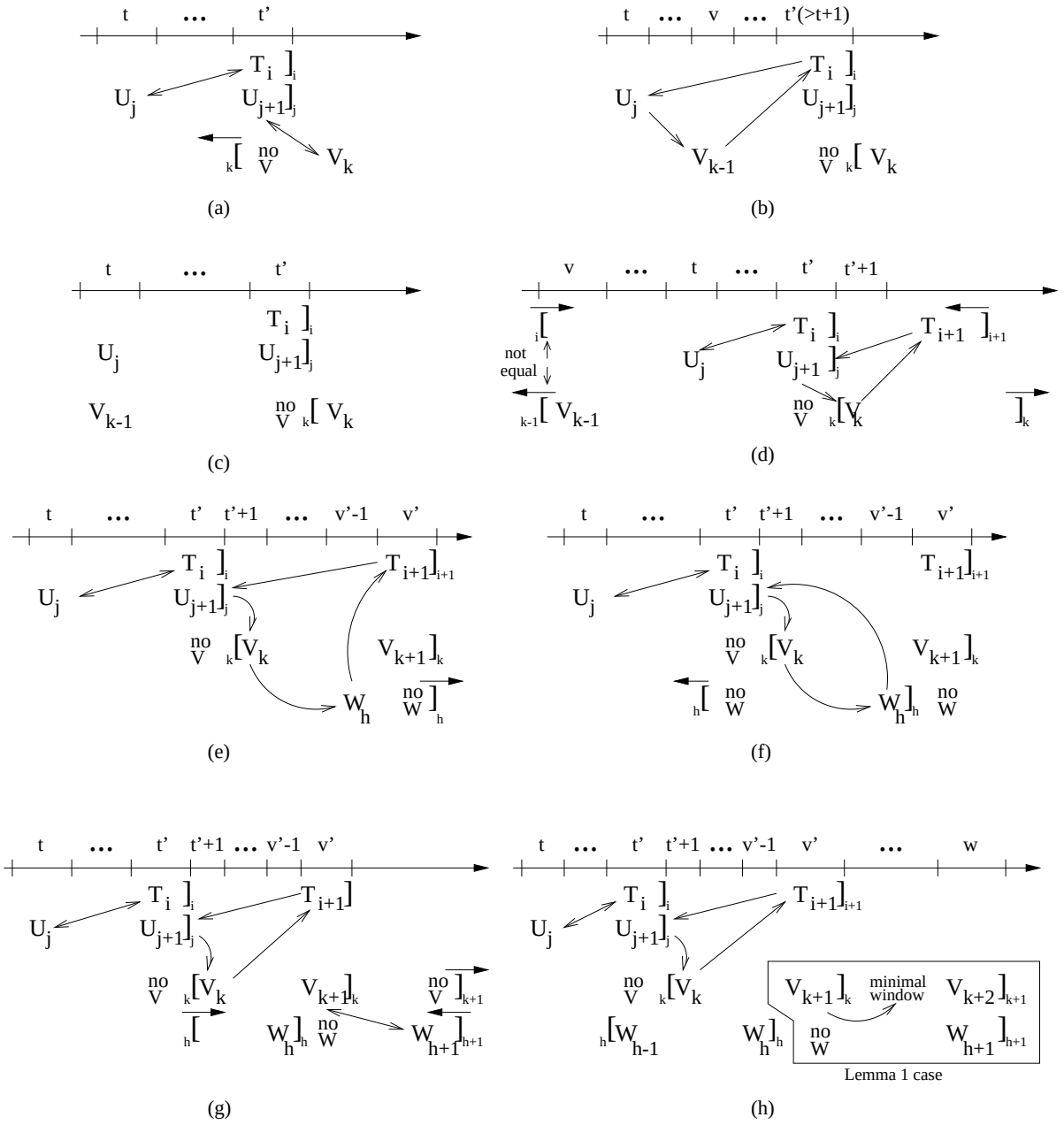


Figure 6: Case 3 (continued). **(a)** $r(V_k) \leq t+1$. **(b)** $r(V_k) = t+2 \wedge r(V_{k-1}) < r(T_i)$. **(c)** $d(W_h) \geq v'$. **(d)** $d(W_h) = v'-1 \wedge r(W_h) \leq t+1 \wedge W \notin S_{t+1}$. **(e)** $d(W_h) = v'-1 \wedge r(W_h) \geq t+2$. **(f)** $d(W_h) = v'-1$, $r(W_h) = t+1$, and $W \in S_{t+1}$.

$b(V_{k-1}) = 0$ and $|w(V_{k-1})| > t'$ or $b(V_{k-1}) = 1$ and $|w(V_{k-1})| > t'+1$. For both possibilities, we have $|w(V_k)| > t' \geq |w(T_i)|$. Therefore, by (P2), $|w(V_k)| \geq |w(T_{i+1})|$. Because $r(V_k) = r(T_{i+1}) + 1$, this implies that $d(V_k) > d(T_{i+1})$. Hence, the swapping in Figure 6(d) 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 $V_{k-1} \in \tau$.

If V_{k-1} is scheduled in the interval (t, t'), then the swapping shown in 6(b) is valid. If V_{k-1} is not scheduled in (t, t'), then it is scheduled at or before t. We now show that it cannot be scheduled in slot t.

Suppose, to the contrary, that V_{k-1} *is* scheduled in slot t, as depicted in Figure 6(c). Because $r(V_k) = t' + 1$, $d(V_{k-1})$ is either $t' + 1$ or t'. If $d(V_{k-1})$ is t', then $b(V_{k-1}) = 0$. In either case, 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.

In the rest of Case 3, we consider the remaining possibility, i.e., V_{k-1} is scheduled at a time $v < t$. Now, it must be 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, as T_i has higher priority than V_{k-1}. 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 task.) Because $r(V_{k-1}) \leq v$, this 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 in Figure 6(d) is valid. There are two possibilities to consider, depending on the value of $b(V_{k-1})$.

$\underline{b(V_{k-1}) = 0}$**.** In this case, by the definition of $b(V_{k-1})$, we have $d(V_{k-1}) = r(V_k) - 1$. By (10) and (11), this implies that $d(V_{k-1}) = d(T_i)$. Since $b(V_{k-1})$ is 0, $w(V_{k-1})$ is the last window of its job, and $w(V_k)$ is the first window of its job. Thus, 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 $|w(T_i)| \leq |w(V_{k-1})| - 1$. Therefore, $|w(T_i)| \leq |w(V_k)| - 1$. By (P2), $|w(T_{i+1})| \leq |w(T_i)| + 1$, and hence, $|w(V_k)| \geq |w(T_{i+1})|$. Because $r(V_k) = r(T_{i+1}) + 1$ (see Figure 6(d)), this implies that $d(V_k) > d(T_{i+1})$.

$\underline{b(V_{k-1}) = 1}$**.** In this case, by the definition of $b(V_{k-1})$, we have $d(V_{k-1}) = r(V_k)$. By (10) and (11), this implies that $d(V_{k-1}) = d(T_i) + 1$. Because $r(V_{k-1}) < r(T_i)$ (our assumption for Subcase 3.A) and $d(V_{k-1}) = d(T_i) + 1$,

$$|w(V_{k-1})| \geq |w(T_i)| + 2. \tag{12}$$

By (P2), $|w(V_k)| \geq |w(V_{k-1})| - 1$ and $|w(T_i)| \geq |w(T_{i+1})| - 1$. Hence, by (12), $|w(V_k)| + 1 \geq |w(T_{i+1})| - 1 + 2$, i.e., $|w(V_k)| \geq |w(T_{i+1})|$. Because $r(V_k) = r(T_{i+1}) + 1$ (again, see Figure 6(d)), 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

$$d(V_k) \geq d(T_{i+1}). \tag{13}$$

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 6(d) is still valid. This will be the case if $d(V_k) > d(T_{i+1})$ or if $d(V_k) = d(T_{i+1}) \ \wedge \ r(V_{k+1}) = d(V_k) + 1$. In the rest of the proof for Subcase 3.B, we assume that T_{i+1} is *not* scheduled before V_{k+1}. By (13), this can happen only if there exists v' satisfying the following (see Figure 6(e)-(h)):

$$d(V_k) = v' \ \wedge \ r(V_{k+1}) = v' \ \wedge \ d(T_{i+1}) = v' \ \wedge \ V_{k+1} \in S_{v'} \ \wedge \ T_{i+1} \in S_{v'}. \tag{14}$$

Before considering other possible swappings, we first show that $w(V_k)$ is a minimal window of V; this fact is used several times in the reasoning that follows. By (11) and the fact that consecutive windows of the same task overlap by at most one slot, $d(V_{k-1})$ is either t' or $t'+1$. If $d(V_{k-1}) = t'$, in which case $w(V_{k-1})$ and $w(V_k)$ do not overlap, then $w(V_k)$ is the first window of its job. Hence, by (P3), $w(V_k)$ is a minimal window. On the other hand, if $d(V_{k-1}) = t'+1$, in which case $w(V_{k-1})$ and $w(V_k)$ do overlap, then we have the following:

- T_i and V_{k-1} are both released at slot v (our assumption for Subcase 3.B),
- T_i has a deadline at t' (see (10)) and V_{k-1} has a deadline at $t'+1$ (by assumption), and
- T_{i+1} and V_k have equal deadlines (by (14)).

By (P1)-(P3), this can happen only if $|w(V_k)| = |w(V_{k-1})| - 1$, which implies that $w(V_k)$ is a minimal window.

To continue, if some processor is idle in slot $v'-1$, then we can left-shift T_{i+1} from v' to $v'-1$ and then apply the swapping in Figure 6(d). In the rest of Subcase 3.B, we assume that no processor is idle in slot $v'-1$.

Now, because T_i and V_{k-1} are both released at v (our assumption for Subcase 3.B), and T_i has a deadline at t' and V_k is released at $t'+1$ (see (10) and (11)), either $|w(V_{k-1})| = |w(T_i)|+1$ or $|w(V_{k-1})| = |w(T_i)| \wedge b(V_{k-1}) = 0$. In either case, because T is light, by (P2)-(P4), V is also light, and hence $|w(V_k)| \geq 3$. By (11) and (14), $w(V_k) = [t'+1, v']$; hence, $v' \geq t'+3$. Because $V_k \in S_{t'+1} \ \wedge \ V_{k+1} \in S_{v'}$, this implies that $V \notin S_{v'-1}$. Thus, because no processor is idle at $v'-1$, there exists a task W that is scheduled in $v'-1$ but not v'. Let W_h be the subtask of W scheduled at $v'-1$. We now show that at least one of the swappings in Figure 6(e)-(h) is valid.

$\underline{d(W_h) \geq v'}$**.** In this case, the swapping in Figure 6(e) is clearly valid. We henceforth assume

$$d(W_h) = v' - 1. \tag{15}$$

$\underline{(r(W_h) < t') \ \vee \ (r(W_h) = t' \ \wedge \ W \notin S_{t'})}$. In this case, the swapping shown in Figure 6(f) is valid.

$\underline{r(W_h) > t' \ \wedge \ d(W_h) = v'}$**.** In this case, we show that $d(W_{h+1}) < d(V_{k+1})$, which implies that the swapping in Figure 6(g) is valid. $r(W_h) > t'$ implies that $|w(W_h)| < |w(V_k)|$ (see Figure 6(g)). Because $w(V_k)$ is a minimal window of V (as shown above), $|w(V_k)| \leq |w(V_{k+1})|$. Thus,

$$|w(W_h)| < |w(V_{k+1})|. \tag{16}$$

Now, consider $b(W_h)$. If $b(W_h) = 0$, then by (15), $r(W_{h+1}) = v'$. Thus, by (14) $r(W_{h+1}) = r(V_{k+1})$. In addition, by (P1), $|w(W_{h+1})| = |w(W_h)|$. Hence, by (16), 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 (15), $r(W_{h+1}) = v'-1$, which by (14) implies that $r(W_{h+1}) < r(V_{k+1})$. In addition, by (P2), $|w(W_{h+1})| \leq |w(W_h)|+1$. Hence, by (16), we have $|w(W_{h+1})| \leq |w(V_{k+1})|$. Therefore, $d(W_{h+1}) < d(V_{k+1})$.

$\underline{r(W_h) = t' \ \wedge \ W \in S_{t'}}$**.** In this case, analysis similar to that above shows that $d(W_{h+1}) \leq d(V_{k+1})$. Let $d(W_{h+1}) = w$. 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 6(g) 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 6(h)):

- W_h is released at slot t' and V_k is released at slot $t' + 1$,
- V_{k+1} is released at slot v', and
- V_{k+1} and W_{h+1} have deadlines at slot w.

By (P1)-(P3), this can happen only if $|w(V_{k+1})| = |w(V_k)|$. Because $w(V_k)$ is a minimal window of V, this implies that $w(V_{k+1})$ is a minimal window as well. Thus, by Lemma 2, there exists a schedule in which V_{k+1} is not scheduled at time v'. The swapping shown in Figure 6(h) 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, $pred(T, t + 7) = t + 3$ and $succ(T, t + 7) = t + 10$.

As before, we are dealing with the situation depicted in Figure 3(b). Because T_i has higher priority than U_j at time t according to PD^2, $D(U_j) \leq D(T_i)$. Because $T_i \in S_{t'}$ and $t' = d(T_i)$ (refer to Figure 3(b)), each subsequent subtask of T with a deadline at or before $D(T_i)$ is scheduled in the last slot of its window. Note that, because T and U are heavy tasks, t' is either $t + 1$ or $t + 2$. Let u be the earliest time after t' such that $U \notin S_u$. Then, $u \leq D(U_j)$. Because $D(U_j) \leq D(T_i)$, this implies that either $u < D(T_i)$ or $u = D(T_i)$. If $u < D(T_i)$ holds then we have the following (refer to Figure 7(a)):

- in all slots in $[t', u]$, a subtask of T is scheduled in the last slot of its window;
- in all slots in $[t', u - 1]$, a subtask of U is scheduled in the first slot of its window;
- no subtask of U is scheduled in slot u.

This implies that the swapping in Figure 7(a) is valid. If $u = D(T_i) \ \wedge \ T \in S_u$, then a similar swapping is valid (in this case, the subtasks of T to be swapped occupy all slots in the interval $[t', D(T_i)]$).

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

$$D(T_i) = D(U_j) \ \wedge \ D(U_j) = u \ \wedge \ T \notin S_u.$$

Let $U_{j+j'}$ be the subtask of U scheduled at $u - 1$. Then, $u = t' + j'$, as shown in Figure 7(b). As the figure shows, 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. In addition, because $T \notin S_u$, $w(T_{i+j'})$ is either a 2-window starting at slot u or a 3-window starting at slot $u - 1$; however, if $w(T_{i+j'})$ is a 2-window starting at slot u, then T has a group deadline at $u - 1$ rather than u. We conclude that $w(T_{i+j'})$ is a 3-window and $T_{i+j'}$ is scheduled in slot $t' + j' + 1$.

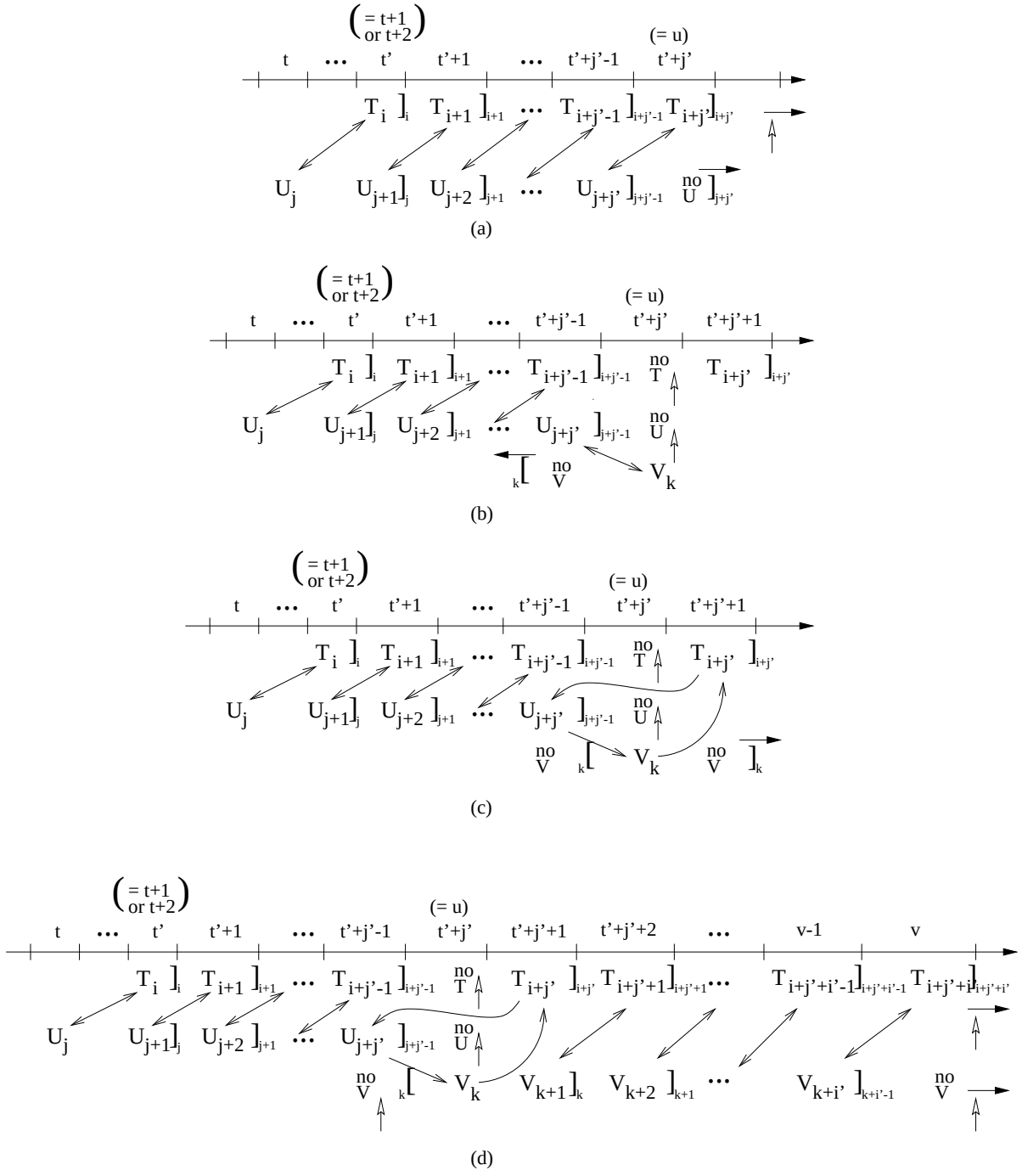


Figure 7: Case 4. We use the following notation in Figures 7-9. A group deadline at slot t is denoted by an up-arrow that is aligned with the right side of slot t. A left- or right-pointing arrow over an up-arrow indicates a group deadline that may be anywhere in the direction of the left- or right-pointing arrow. **(a)** $D(T_i) > D(U_j)$ or $D(T_i) = D(U_j) \wedge T \in S_u$. **(b)** $D(T_i) = D(U_j)$ and $r(V_k) < t' + j'$. **(c)** $D(T_i) = D(U_j)$, $r(V_k) = t' + j'$, and $V \notin S'_{t'+j'+1}$. **(d)** $D(T_i) = D(U_j)$, $r(V_k) = t' + j'$, and $D(T_{i+j'}) > v$.

If some processor is idle in slot u, then shifting subtask $U_{j+j'}$ to slot u produces a situation in which a swapping like that in Figure 7(a) is applicable. We henceforth assume that no processor is idle at 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 u, and because no processor is idle at u, there exists a task V scheduled at u but

not $u-1$. Let V_k be the subtask of V scheduled at u. If $r(V_k) < u$, then the swapping in Figure 7(b) is valid, and if $r(V_k) = u \ \wedge \ V \notin S_{u+1}$, then the swapping in Figure 7(c) is valid. In the rest of the proof, we assume

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

Note that $V \in S_{u+1}$ implies that $d(V_k) = u+1$, i.e., $|w(V_k)| = 2$. Therefore, by (P4), V is heavy. Consider the group deadline of V_k, $D(V_k)$. Let v be the earliest slot after u such that $V \notin S_v$. Note that

$$v \leq D(V_k). \tag{17}$$

(See Figure 7(d).) Let $V_{k+i'}$ be the subtask of V that is scheduled in slot $v-1$. If either $v < D(T_{i+j'})$ or $v = D(T_{i+j'}) \ \wedge \ b(T_{i+j'+i'}) = 0$, then $T_{i+j'+i'}$ is scheduled in slot v, and the swapping shown in Figure 7(d) is valid. In the rest of the proof, we assume that neither of these conditions holds, i.e.,

$$[D(T_{i+j'}) < v] \ \vee \ [v = D(T_{i+j'}) \ \wedge \ b(T_{i+j'+i'}) = 1]. \tag{18}$$

We claim that $u-1$ is a group deadline of V. As seen in Figure 7(d), V_{k-1} is not scheduled in slot $u-1$. Because $r(V_k) = u$, $d(V_{k-1})$ is either $u-1$ or u. If $d(V_{k-1}) = u-1$, then $b(V_{k-1}) = 0$ and $w(V_{k-1})$ is the final window of a job of V, and thus $u-1$ is a group deadline. If $d(V_{k-1}) = u$, then we reason as follows. Because V is a heavy task, by (P2) and (P4), $|w(V_{k-1})| \leq 3$. Because V_{k-1} is not scheduled in slots $u-1$ or u, $r(V_{k-1})$ has to be $u-2$. This implies that $|w(V_{k-1})| = 3$. Therefore, $u-1$ is a group deadline of V.

Having shown that $u-1$ is a group deadline of V, we now show that $pred(V, u-1) \leq pred(T, u)$. T has consecutive group deadlines at u and $succ(T, u) = D(T_{i+j'})$. Therefore, by (P5), the difference between u and $pred(T, u)$ is at most one more than $D(T_{i+j'}) - u$, i.e., $u - pred(T, u) \leq D(T_{i+j'}) - u + 1$. Therefore,

$$pred(T, u) \geq 2u - D(T_{i+j'}) - 1. \tag{19}$$

V has consecutive group deadlines at $u-1$ and $succ(V, u-1) = D(V_k)$. Hence, by (P5), the difference between $u-1$ and $pred(V, u-1)$ is at least one less than $D(V_k) - (u-1)$, i.e., $u - 1 - pred(V, u-1) \geq D(V_k) - u$. Thus,

$$pred(V, u-1) \leq 2u - D(V_k) - 1. \tag{20}$$

By (17), (18), (19), and (20), $pred(V, u-1) \leq 2u - D(V_k) - 1 \leq 2u - D(T_{i+j'}) - 1 \leq pred(T, u)$. Thus, $pred(V, u-1) \leq pred(T, u)$. Note also that $pred(V, u-1) = pred(T, u)$ iff $D(T_{i+j'}) = D(V_k) = v$. In addition, as seen in Figure 7(d), T cannot have a group deadline in the interval $[t', u-1]$. Therefore, we have the following.

$$pred(V, u-1) \leq pred(T, u) \leq t' - 1 \tag{21}$$

$$(pred(V, u-1) = pred(T, u)) \ \Rightarrow \ (D(T_{i+j'}) = v \ \wedge \ D(V_k) = v) \tag{22}$$

Recall that t' is either $t+2$ or $t+1$. We consider these two subcases next.

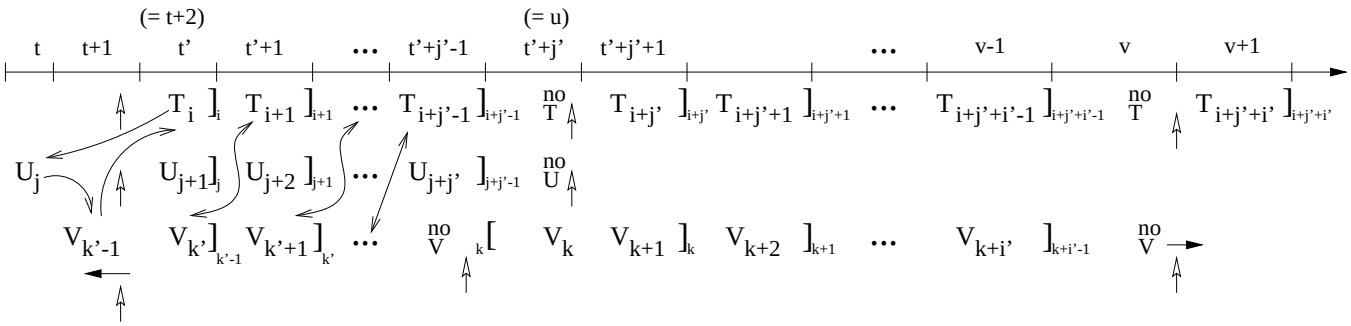


Figure 8: Subcase 4.A. $t' = t + 2$.

Subcase 4.A: $t' = t + 2$. In this case, we show that the swapping in Figure 8 is valid. To begin, note that $t' = t + 2$ implies that $T \notin S_{t+1}$ and $U \notin S_{t+1}$. Let $k' = k - j' + 1$. As Figure 8 shows,

- $w(V_{k-1})$ is a 2- or 3-window starting in slot $u - 2 = t' + j' - 2$ (since $u - 1$ is a group deadline of V);
- for each l in the range $k' \leq l < k - 1$, $w(V_l)$ is a 2-window (this is because, by (21), $pred(V, u - 1) < t'$);
- each of $V_{k'}, \ldots, V_{k-1}$ is scheduled in the first slot of its window (this can be seen by inducting from right to left, starting with V_{k-1}).

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 $[0, t' + j' - 1]$ (see Figure 8) and therefore, by Lemma 1, they are in τ.

Because $V_{k'}$ is released at $t' = t + 2$, $V_{k'-1}$ has a deadline at either $t + 1$ or $t + 2$. We now prove that $d(V_{k'-1}) \neq t + 1$. Assume, to the contrary, that $d(V_{k'-1}) = t + 1$. Because $w(V_{k'})$ is a 2-window starting in slot $t' = t + 2$ (see Figure 8), this implies that $b(V_{k'-1}) = 0$, i.e., $w(V_{k'-1})$ is the final window of its job. Thus,

$$pred(V, u - 1) = t + 1. \tag{23}$$

If V has multiple group deadlines per job, then by (P5) and (P6), the difference between $pred(V, u - 1)$ and $u - 1$ is at least $succ(V, u - 1) - (u - 1)$, i.e., $succ(V, u - 1) - (u - 1) \leq (u - 1) - pred(V, u - 1)$. If V has one group deadline per job, then clearly $succ(V, u - 1) - (u - 1) = (u - 1) - pred(V, u - 1)$. In either case, by (23),

$$succ(V, u - 1) \leq 2u - t - 3. \tag{24}$$

Because $t' = t + 2$, $w(T_i)$ is a 3-window. Thus, $pred(T, u) = t' - 1 = t + 1$. By (P5), the difference between $succ(T, u)$ and u is at least one less than $u - pred(T, u)$, i.e., $succ(T, u) - u \geq u - (t + 1) - 1$. Because $succ(T, u) = D(T_{i+j'})$, this implies that $D(T_{i+j'}) \geq 2u - t - 2$. Thus, by (18), $v \geq 2u - t - 2$. Because $succ(V, u-1) = D(V_k)$, by (17), we have $succ(V, u-1) \geq v$. Thus, $succ(V, u-1) \geq 2u-t-2$, which contradicts (24). Therefore, we conclude that $d(V_{k'-1})$ cannot be $t + 1$. Thus, we have the following.

$$d(V_{k'-1}) = t + 2$$

We now show that $V_{k'-1}$ is scheduled in slot $t+1$, which implies that the swapping in Figure 8 is valid. Assume, to the contrary, that $V_{k'-1}$ is *not* scheduled at $t+1$. We have established that V is heavy and $d(V_{k'-1}) = t+2$. Moreover, $V_{k'-1}$ is not scheduled in slot $t+1$ (by assumption) or in slot $t+2$ (because $V_{k'}$ is scheduled there). By (P2)-(P4), this implies that $w(V_{k'-1})$ is a 3-window, $r(V_{k'-1}) = t$, and $V_{k'-1}$ is scheduled in slot t. As seen in Figure 8, $d(V_{k'-1}) = d(U_j)$, $b(V_{k'-1}) = b(U_j)$, and $D(V_{k'-1}) < D(U_j)$. Thus, $V_{k'-1}$ has lower priority than U_j at time t, contradicting our choice of U_j as the lowest-priority subtask scheduled at t.

Subcase 4.B: $t' = t+1$. In this case, we show that one of the swappings in Figure 9 is valid. We first prove

$$pred(V, u-1) = t. \tag{25}$$

As in Subcase 4.A, it is possible to show that

- each of $V_{k'}, \ldots, V_{k-2}$ has a window of length two and is scheduled in the first slot of its window, and
- V_{k-1} has a window of length two or three 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.) This is depicted in Figure 9(a). Because $r(V_{k'}) = t+1$, $d(V_{k'-1})$ is either t or $t+1$. If $d(V_{k'-1}) = t$, then $w(V_{k'-1})$ is the final window of its job. Therefore, $pred(V, u-1) = t$.

If $d(V_{k'-1}) = t+1$, then we reason as follows. Because V is heavy, by (P2) and (P4), $w(V_{k'-1})$ is of length two or three. If $|w(V_{k'-1})| = 3$, then clearly, $pred(V, u-1) = t$. Thus, it suffices to show that $|w(V_{k'-1}) \neq 2$. Suppose, to the contrary, that $|w(V_{k'-1})| = 2$. Because $d(V_{k'-1}) = t+1$, and because $V_{k'}$ is scheduled in slot $t+1$, $V_{k'-1}$ is scheduled in slot t. Observe that $d(V_{k'-1}) = t+1$, $d(U_j) = t+1$, $b(V_{k'-1}) = 1$, $b(U_j) = 1$, and $D(V_{k'-1}) < D(U_j)$. 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 our proof of (25). By (21) and (25), we have

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

By (22), this implies that $D(T_{i+j'}) = D(V_k) = v$ (see Figure 9(a)).

Because $T \notin S_u$ and $T \in S_{u+1}$, and because no processor is idle at u, there exists a task W that is scheduled at u but not $u+1$. Let W_h be the subtask of W scheduled at u. If $d(W_h) > u$, then the swapping shown in Figure 9(b) is valid. In the rest of the proof, we assume that

$$d(W_h) = u.$$

In this case, we show that one of the swappings in Figure 9(c)-(e) are valid. If $W \notin S_{u-1}$, then the swapping shown in Figure 9(c) is valid. In the rest of the proof, we assume

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

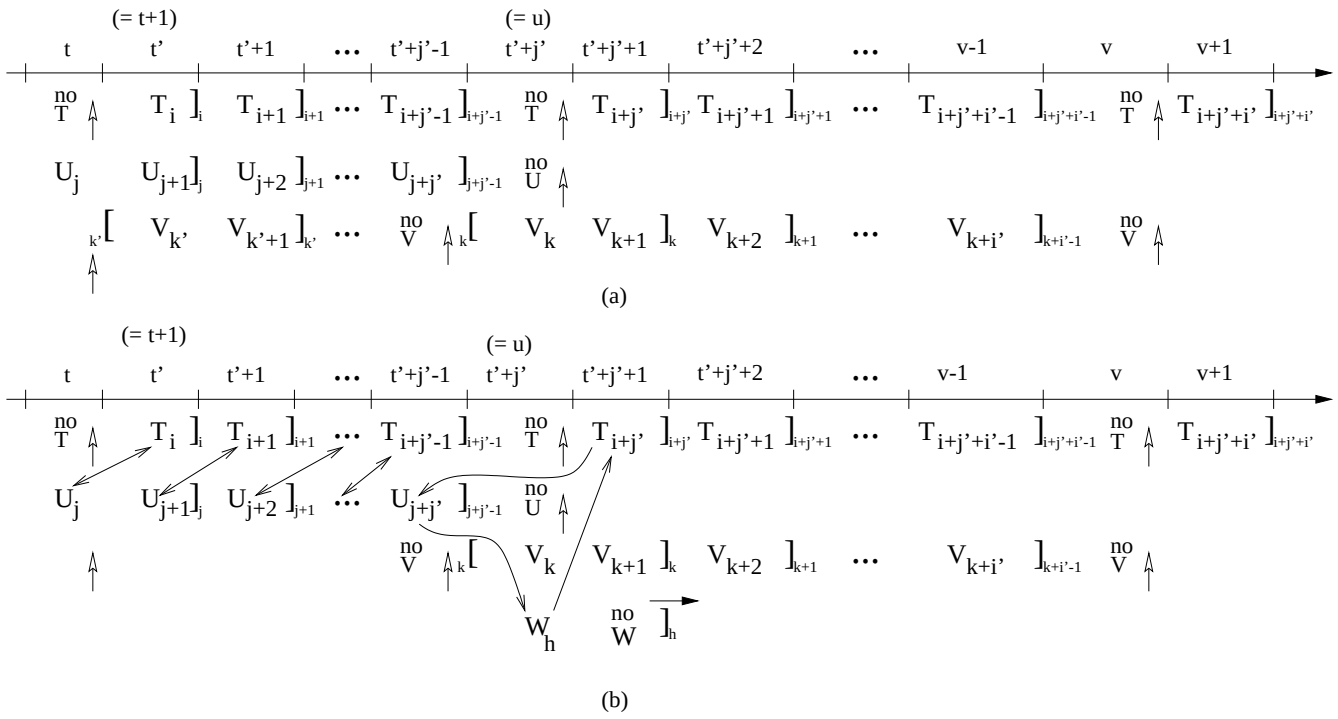


Figure 9: Subcase 4.B. In each inset, $t' = t+1$, $D(T_i) = D(U_j)$, and $D(V_k) = D(T_{i+j'})$. **(a)** $pred(V, u-1) = pred(T, u)$. **(b)** $d(W_h) > u$. (Continued on the following page.)

In this case, we have $r(W_h) = u - 1$, i.e., $|w(W_{h-1})| = 2$. Thus, by (P4), W is heavy.

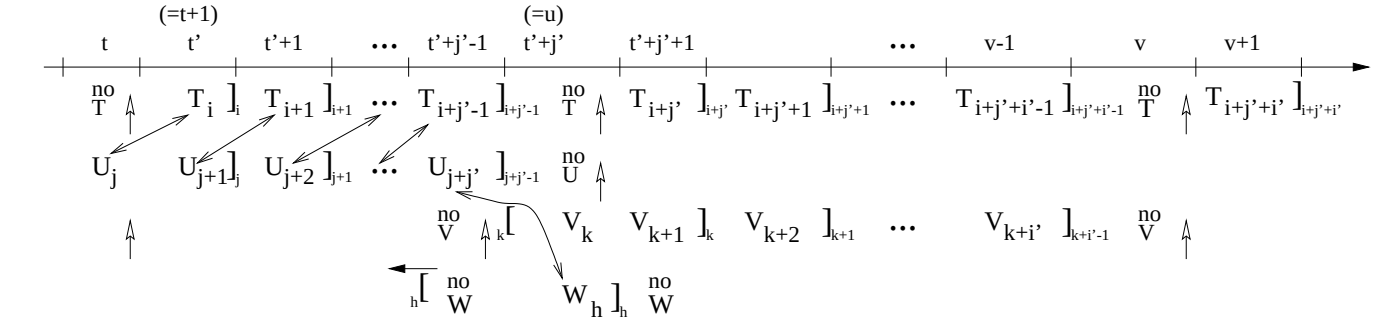
We now show that W has a group deadline at time u or $u+1$ (refer to Figure 9(d)). Because $W_{h+1} \notin S_{u+1}$ and W is heavy, by (P2) and (P4), W_{h+1} has to be scheduled at time $u+2$, and $w(W_{h+1})$ is either a 3-window beginning at slot u or a 2-window beginning at slot $u+1$. In the former case, $u+1$ is the middle slot of a 3-window, and in the latter case, u is the final slot of a job of W, so either u or $u+1$ is a group deadline of W.

We now look at earlier subtasks of W. If there exists w such that $t' \leq w \leq u-1$ and $W \notin S_w$, then a swapping similar to that shown in Figure 9(d) 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 $[t', u]$, a subtask of W is scheduled in the last slot of its window (recall that W is heavy). This is illustrated in Figure 9(e). As seen in the figure, each of the subtasks $W_{h-j'+1}, \ldots, W_h$ has a window of length two. This implies that the most recent group deadline of W before the one at u or $u+1$ occurs at or before time t, i.e.,

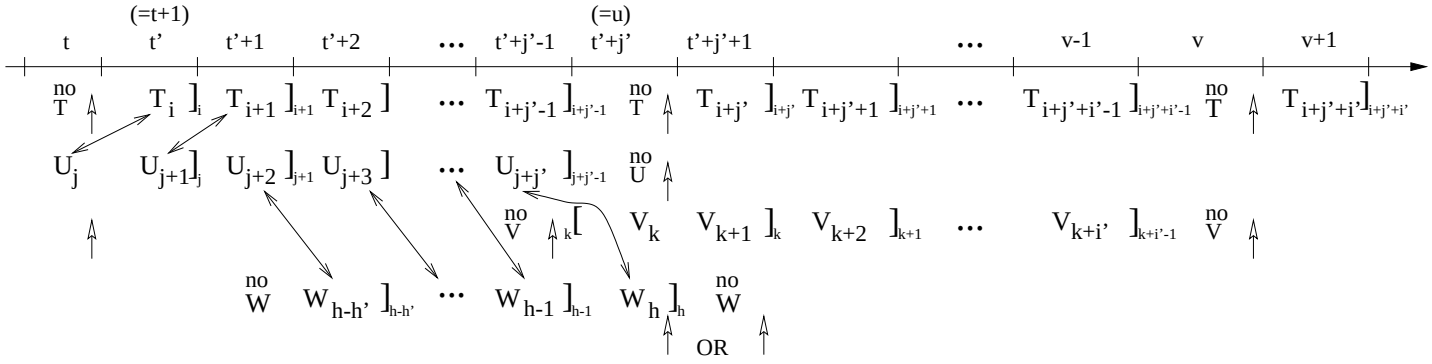
$$\begin{aligned} (u \text{ is a group deadline of } W \quad &\Rightarrow \quad pred(W, u) \leq t) \ \wedge \\ (u+1 \text{ is a group deadline of } W \quad &\Rightarrow \quad pred(W, u+1) \leq t). \end{aligned} \tag{26}$$

We now show that W's next group deadline after the one at u or $u+1$ occurs after time v, which implies that the swapping shown in Figure 9(e) is valid. There are two possibilities to consider, depending on whether W has a group deadline at u or $u+1$. In both cases, by (17), (20), and (25), we have

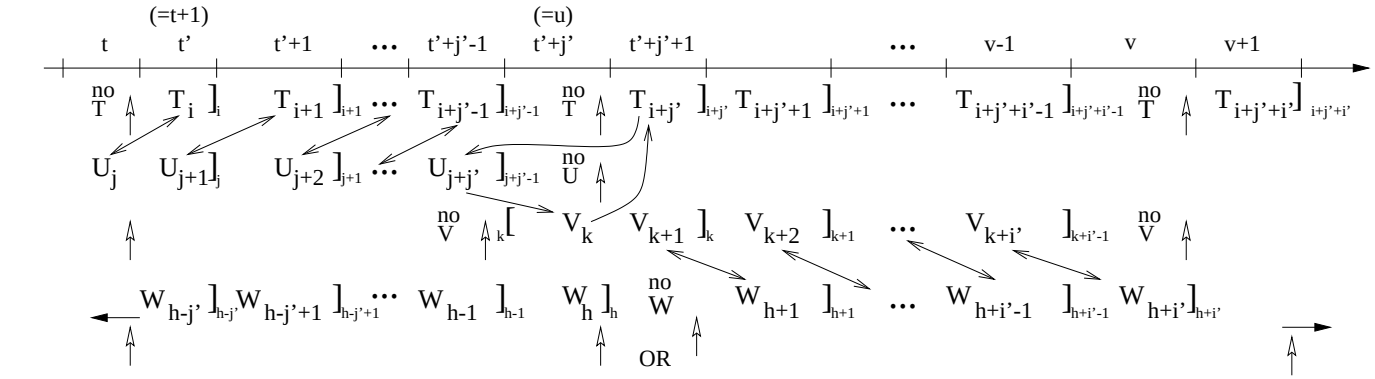
$$v \leq 2u - t - 1. \tag{27}$$



(c)



(d)



(e)

Figure 9: (Continued) **(c)** $d(W_h) = u$ and $W \notin S_{u-1}$. **(d)** $d(W_h) = u$, $W \in S_{u-1}$, and $W \notin S_w$ for some w in $[t', u-1]$. **(e)** $d(W_h) = u$, $W \in S_{u-1}$, and W's most recent group deadline before the one at u or $u+1$ is at or before t.

u is a group deadline of W. In this case, W_h is the last subtask of its job. By (P5) and (P6), the difference between $succ(W, u)$ and u is at least the difference between u and $pred(W, u)$, i.e., $succ(W, u) - u \geq u - pred(W, u)$. Furthermore, by (26), $pred(W, u) \leq t$. Therefore, $succ(W, u) \geq 2u - t$. By (27), this implies that $succ(W, u) > v$.

$u+1$ is a group deadline of W. In this case, by (P5), the difference between $succ(W, u+1)$ and $u+1$ is at least one less than the difference between $u+1$ and $pred(W, u+1)$, i.e., $succ(W, u+1) - (u+1) \geq$

$(u+1) - pred(W, u+1) - 1$. Therefore, $succ(W, u+1) \geq 2u - pred(W, u+1) + 1$. Furthermore, by (26), $pred(W, u+1) \leq t$. This implies that $succ(W, u+1) \geq 2u - t + 1$. Therefore, by (27), $succ(W, u+1) > v$.

This exhausts all the possibilities if T and U are both heavy, and concludes the proof of Lemma 3. □

By applying Lemma 3 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 1 *PD^2 generates a valid schedule for any feasible asynchronous periodic task system in which each task's lag is bounded by either (2) or (3).*

5 Concluding Remarks

We have shown that the PD^2 algorithm 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. In addition, this is the first paper to show that any variant of the PD Pfair algorithm is optimal for scheduling asynchronous task systems on a multiprocessor. We are currently trying to extend the results of this paper to show that PD^2 is also optimal for scheduling sporadic task systems.

Acknowledgements: We are grateful to Sanjoy Baruah, Mark Moir, and Srikanth Ramamurthy for many helpful discussions on the subject of this paper.

References

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

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

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

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

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

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

Appendix: Proof of Lemma 2

In this appendix, we prove Lemma 2, which is restated below.

Lemma 2 *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, U_{j+1} and T_i are scheduled in slot t', and $r(U_j) = t$, $d(U_j) = t'$, $r(U_{j+1}) = t'$, and $d(T_i) = t'$, where $w(U_j)$ 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: Note that 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 (P2), all windows of each span at least three slots. Thus, $t' > t + 1$ and $\{T, U\} \not\subseteq S_{t'-1}$.

If some processor is idle at slot $t' - 1$, then the swapping shown in Figure 10(b) gives the required schedule.

We henceforth assume that no processor is idle at slot $t' - 1$. In this case, because $\{T, U\} \not\subseteq S_{t'-1}$ and $\{T, U\} \subseteq S_{t'}$, there exists a task V that is scheduled at $t' - 1$ but not t'. Let V_k be the subtask of V scheduled at $t' - 1$. If $d(V_k) > t' - 1$, then the swapping shown in Figure 10(c) gives the desired schedule. In the rest of the proof, we assume

$$d(V_k) = t' - 1. \tag{28}$$

If $r(V_k) < t$ or if $r(V_k) = t \ \wedge \ V \notin S_t$, then the swapping shown in Figure 10(d) produces the desired schedule. The remaining possibility to consider is

$$(r(V_k) > t) \ \vee \ (r(V_k) = t \ \wedge \ V \in S_t). \tag{29}$$

In this case, we show that the swapping in Figure 10(e) is valid (this figure actually depicts the case $r(V_k) = t \wedge V \in S_t$). From (28), (29), and the statement of the lemma, we have $d(V_k) = d(U_j) - 1$ and $r(V_k) \geq r(U_j)$. This implies that

$$|w(V_k)| < |w(U_j)|. \tag{30}$$

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

$$d(U_{j+1}) \geq t' - 1 + |w(U_j)|. \tag{31}$$

Now, by (28) and the fact that consecutive windows of a task overlap by at most one slot, V_{k+1} is released at either $t' - 1$ or t'. We now show that in either case, $d(V_{k+1}) \leq t' - 1 + |w(V_k)|$. If $r(V_{k+1}) = t'$, then by (P1), $|w(V_{k+1})| = |w(V_k)|$. By definition, $|w(V_{k+1})| = d(V_{k+1}) - r(V_{k+1}) + 1$. Therefore, $d(V_{k+1}) = t' - 1 + |w(V_k)|$.

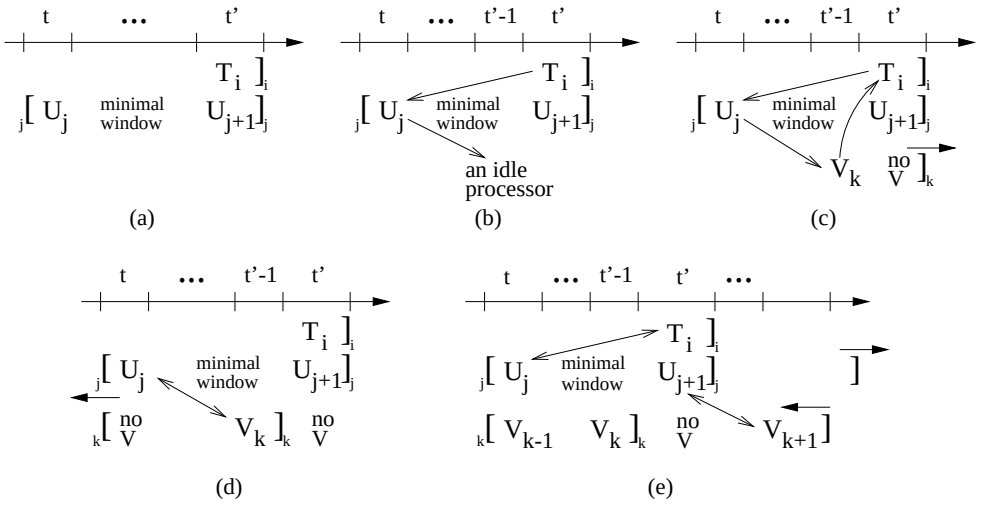


Figure 10: **(a)** Conditions of Lemma 2. **(b)** $d(V_k) > t' - 1$. **(c)** $d(V_k) = t' - 1$ and $r(V_k) \leq t$. **(d)** $d(V_k) = t' - 1$ and $r(V_k) \geq t$.

On the other hand, if $r(V_{k+1}) = t' - 1$, then we reason as follows. By (P2), $|w(V_{k+1})| \leq |w(V_k)| + 1$. Because $|w(V_{k+1})| = d(V_{k+1}) - r(V_{k+1}) + 1$, it follows that $d(V_{k+1}) \leq t' - 1 + |w(V_k)|$. Hence, in both cases, $d(V_{k+1}) \leq t' - 1 + |w(V_k)|$. By (30) and (31), this implies that $d(U_{j+1}) > d(V_{k+1})$. Thus, the swapping shown in Figure 10(e) is valid, and produces the required schedule. □