# Fine-Grained Task Reweighting on Multiprocessors *

Technical Report TR06-008

Aaron Block, James H. Anderson, and Gary Bishop

Department of Computer Science, University of North Carolina at Chapel Hill

April 2006

**Abstract**

We consider the problem of *task reweighting* in fair-scheduled multiprocessor systems wherein each task's processor share is specified as a *weight*. When a task is reweighted, a new weight is computed for it, which is then used in future scheduling. Task reweighting can be used as a means for consuming (or making available) spare processing capacity. The responsiveness of a reweighting scheme can be assessed by comparing its allocations to those of an ideal scheduler that can reweight tasks instantaneously. A reweighting scheme is *fine-grained* if any additional per-task "error" (in comparison to an ideal allocation) caused by a reweighting event is constant. In prior work on *uniprocessor* notions of fairness, a number of fine-grained reweighting schemes were proposed. However, in the multiprocessor case, prior work has failed to produce such a scheme. In this paper, we remedy this shortcoming by presenting a multiprocessor reweighting scheme that is fine-grained. We also present an experimental evaluation of this scheme that shows that it is often much more responsive than prior (non-fine-grained) schemes in enacting weight-change requests.

*Work supported by NSF grants CCR 0204312, CCR 0309825, CCR 0408996, and CCR 0541056. The first author was also supported by an NSF fellowship. A preliminary version of this paper appeared in *Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 429-435, August 2005

# 1 Introduction

Two trends are evident in recent work on real-time systems. First, *multiprocessor* designs are becoming common. This is due both to the advent of reasonably-priced multiprocessor platforms and to the prevalence of computationally-intensive applications with real-time requirements that have pushed beyond the capabilities of single-processor systems. Second, many applications now exist that require *fine-grained adaptivity*, *i.e.*, the ability to react to external events within short time scales by adjusting task parameters, particularly *processor shares*. Examples of such applications include human-tracking systems, computer-vision systems, and signal-processing applications such as synthetic aperture imaging.

To better motivate the need for fine-grained adaptivity, we consider in this paper one particular example application in some detail—namely, the Whisper tracking system designed at the University of North Carolina to perform full-body tracking in virtual environments [13]. Like many tracking systems, Whisper uses *predictive techniques* to track objects. The computational cost of making the "next" prediction in tracking an object depends on the accuracy of the previous one, as an inaccurate prediction requires a larger space to be searched. Thus, the processor shares of the tasks that are deployed to implement these tracking functions vary with time. In fact, the variance can be as much as *two orders of magnitude*. Moreover, share changes must be enacted within *time scales as short as 10 ms*.

In this paper, we consider the specific issue of how to support adaptive behavior such as this on (tightly-coupled) multiprocessors when using *fair* global scheduling algorithms, specifically Pfair algorithms [3], as introduced later. In fair scheduling schemes, correctness is defined by comparing to an *ideal* scheduler that can guarantee each task *precisely* its required share over any time interval. Such an ideal scheduler can instantaneously enact share changes, but is impractical to implement, as it requires the ability to preempt and swap tasks at arbitrarily small time scales. In practical schemes, share allocations track the ideal scheduler with only bounded "error." We consider an allocation policy to be *fine-grained* if any additional per-task "error" (in comparison to an ideal allocation) caused by a task share-change request is constant. We use the term *drift* to refer to this source of error, and refer to the process of changing a task's share as *reweighting*.

Srinivasan and Anderson [11] have given sufficient conditions (described in Sec. 2) under which tasks may dynamically join and leave a running Pfair-scheduled system without causing any missed deadlines. These rules can be applied to reweight tasks: such a task simply leaves with its old weight and rejoins with its new weight. However, as discussed later, these rules require that tasks sometimes be delayed when leaving the system. Because of these "leaving delays," any reweighting scheme constructed from these rules is *coarse-grained*, *i.e.*, susceptible to non-constant drift.

After the presentation of the results herein in preliminary form [6], we subsequently considered the use of partitioned [4] and global EDF [7] scheduling algorithms to schedule highly-adaptive multiprocessor workloads. While partitioning and global EDF provide excellent average-case performance and can reduce migration and preemption costs associated with Pfair scheduling, both algorithms have substantial drawbacks. Specifically, under partitioning, fine-grained reweighting is (provably) impossible; under global EDF, fine-grained reweighting is possible only if deadline misses are permissible.

In this paper, we show that fine-grained reweighting (without deadline misses) is possible under Pfair scheduling by presenting reweighting rules that ensure *constant* drift. These rules are introduced in the following way. After first presenting a more careful review of prior work in Sec. 2, we present in Sec. 3 a new task model that allows task weights to vary with time and associated reweighting rules. In Sec. 4, we prove that, under our reweighting rules, no task misses a deadline and drift is constant. (The proof that deadline misses are avoided is rather lengthy; much of this proof is deferred to an appendix.) We also show that *zero drift is not possible*; hence, our rules cannot be substantially improved. In Sec. 5, we assess the efficacy of these rules via an experimental evaluation involving the Whisper system.

# 2 Preliminaries

Under Pfair scheduling, processor time is allocated in discrete time units, called *quanta*; the time interval $[t, t+1)$, where $t$ is a nonnegative integer, is called *slot* $t$. (Hence, time $t$ refers to the beginning of slot $t$.) In this paper, all time values are assumed to indicate an integral number of quanta, unless specified otherwise. Throughout the paper, we use $M$ to denote the number processors in the system.

As mentioned in the introduction, under Pfair scheduling, correctness is defined by comparing to an "ideal" scheduling algorithm that can guarantee each task precisely its required share over any time interval. Thus, as we introduce increasingly flexible task models, we also introduce increasingly more general (and complex) notions of ideal scheduling. The behavior of each scheduling algorithm (ideal or otherwise) presented in this paper is defined by the sequence of its allocation decisions over time. The total time allocated to a task $T$ in an arbitrary schedule $\mathcal{S}$ over the range $[t_1, t_2)$ is denoted as $\mathsf{A}(\mathcal{S}, T, t_1, t_2)$. Similarly, we use $\mathsf{A}(\mathcal{S}, T_j, t_1, t_2)$ and $\mathsf{A}(\mathcal{S}, \tau, t_1, t_2)$ to denote, respectively, the total allocations to the "subtask" $T_j$ (as defined below) and to all tasks in the set $\tau$ over the range $[t_1, t_2)$. As a shorthand, we use $\mathsf{A}(\mathcal{S}, T, t)$ to denote $\mathsf{A}(\mathcal{S}, T, t, t+1)$. In order to compare the difference between the allocations to a task $T$ up to time $t$ in two schedules $\mathcal{S}$ and $\mathcal{I}$, where $\mathcal{S}$ is an "actual" schedule and $\mathcal{I}$ is an "ideal" one, we use the function $\mathsf{lag}(\mathcal{S}, \mathcal{I}, T, t) = \mathsf{A}(\mathcal{I}, T, 0, t) - \mathsf{A}(\mathcal{S}, T, 0, t)$. Additionally, we use the function $\mathsf{LAG}(\mathcal{S}, \mathcal{I}, \tau, t) = \sum_{T \in \tau} \mathsf{lag}(\mathcal{S}, \mathcal{I}, T, t)$ to compare the differences in allocations for *all tasks* in the task set $\tau$ in schedules $\mathcal{S}$ and $\mathcal{I}$. We assume $\mathsf{lag}(\mathcal{S}, \mathcal{I}, T, 0) = 0$. Thus, $\mathsf{LAG}(\mathcal{S}, \mathcal{I}, \tau, t)$ can be rewritten as

$$\mathsf{LAG}(\mathcal{S}, \mathcal{I}, \tau, t) = \mathsf{LAG}(\mathcal{S}, \mathcal{I}, \tau, t-1) + (\mathsf{A}(\mathcal{I}, \tau, t-1) - \mathsf{A}(\mathcal{S}, \tau, t-1)). \quad (1)$$

For brevity, we denote $\mathsf{lag}(\mathcal{S}, \mathcal{I}, T, t)$ as $\mathsf{lag}(T, t)$ and $\mathsf{LAG}(\mathcal{S}, \mathcal{I}, \tau, t)$ as $\mathsf{LAG}(\tau, t)$, when $\mathcal{S}$ and $\mathcal{I}$ are well-defined and obvious. (Examples illustrating the concepts in this paragraph are given shortly.)

**Periodic Pfair scheduling.** In defining notions relevant to Pfair scheduling, we limit attention (for now) to periodic tasks, all of which begin execution at time 0. A periodic task $T$ with an integer *period* $T.p$ and an integer *execution cost* $T.e$ has a *weight* (or *utilization*) $\mathsf{wt}(T) = T.e/T.p$, where $0 < \mathsf{wt}(T) \leq 1$. Due to page limitations, we focus exclusively in this paper on tasks with weight at most $1/2$. Tasks of weight greater than $1/2$, called *heavy tasks*, require additional reasoning, which can be found in the first author's upcoming Ph.D. dissertation. (It is worth noting that the Whisper system used as a test case herein requires task weights of at most 1/3.)

The ideal schedule for a periodic task system allocates $\mathsf{wt}(T)$ processing time to each task in each time slot. Thus, for a periodic system $\tau$, $\mathsf{lag}(T, t)$ is defined as $\mathsf{wt}(T) \cdot t - \mathsf{A}(\mathcal{S}, T, 0, t)$, where $\mathcal{S}$ is some "real" schedule of $\tau$. The schedule

$\mathcal{S}$ is *Pfair* iff $(\forall T \in \tau, t :: -1 < \mathsf{lag}(T, t) < 1)$. Informally, each task's allocation error must always be less than one quantum. These error bounds are ensured by treating each quantum of a task's execution, henceforth called a *subtask*, as a schedulable entity. Scheduling decisions are made only at quantum boundaries. The $i^{th}$ subtask of task $T$, denoted $T_i$, where $i \geq 1$, has an associated *pseudo-release* $\mathsf{r}(T_i) = \lfloor (i-1)/\mathsf{wt}(T) \rfloor$ and *pseudo-deadline* $\mathsf{d}(T_i) = \lceil i/\mathsf{wt}(T) \rceil$. (For brevity, we often drop the prefix "pseudo-.") It can be shown that if each subtask $T_i$ is scheduled in the interval $\mathsf{w}(T_i) = [\mathsf{r}(T_i), \mathsf{d}(T_i))$, termed its *window*, then $(\forall T \in \tau, t :: -1 < \mathsf{lag}(T, t) < 1)$ is maintained [2]. As an example, in Fig. 1(a), $\mathsf{r}(T_2) = 3$, $\mathsf{d}(T_2) = 7$, and $\mathsf{w}(T_2) = [3, 7)$. (This figure also depicts per-slot ideal allocations for each subtask, which are considered below.) Thus, $T_2$ must be scheduled in slots 3–6. (Tasks execute sequentially, so if $T_1$ is scheduled in slot 3, then $T_2$ must be scheduled in slots 4–6.)

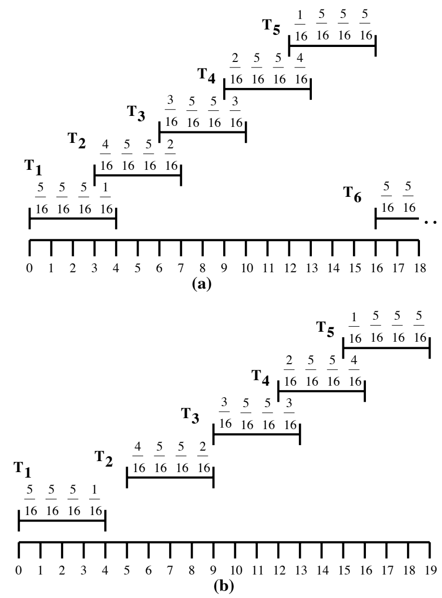


Figure 1: $\mathsf{A}(\mathcal{I}, T_j, t)$ for a **(a)** periodic and **(b)** IS task of weight 5/16. The windows of successive subtasks are as indicated (*e.g.*, $T_1$'s window in both insets is $[0, 4)$).

**IS model.** The *intra-sporadic* (*IS*) *task model* [10] generalizes the well-known sporadic task model [9] by allowing subtasks to be released late. This extra flexibility is useful in many applications where processing steps may be delayed. Fig. 1(b) illustrates the Pfair windows of an IS task of weight 5/16 in which the release of $T_2$ is delayed by two quanta and the release of $T_3$ is delayed by an additional quantum. Each subtask $T_i$ of an IS task has an *offset* $\theta(T_i)$ that gives the amount by which its release has been delayed. For example, in Fig. 1(b), $\theta(T_1) = 0$, $\theta(T_2) = 2$, and for $i \geq 3$, $\theta(T_i) = 3$. The release and deadline of a subtask $T_i$ of an IS task $T$ are defined as $\mathsf{r}(T_i) = \theta(T_i) + \lfloor (i-1)/\mathsf{wt}(T) \rfloor$ and $\mathsf{d}(T_i) = \theta(T_i) + \lceil i/\mathsf{wt}(T) \rceil$, where the offsets satisfy the property $k \geq i \Rightarrow \theta(T_k) \geq \theta(T_i)$. A subtask $T_i$ is *active* at the time $t$ iff $\mathsf{r}(T_i) \leq t < \mathsf{d}(T_i)$, and a task $T$ is *active* at $t$ iff it has an active subtask at $t$. For example, in Fig. 1(b), $T$ is active in every slot except slot 4. If $\theta(T_i) < \theta(T_{i+1})$, then we say that there is an *IS separation* between $T_i$ and $T_{i+1}$. (Note that an extension of the IS model exists in which a subtask $T_i$ can become eligible before $\mathsf{r}(T_i)$ [10]. All the results of this paper can be easily extended to such a model, but for clarity, we do not consider this extension to the IS model.)

**The PD$^2$ algorithm.** The $\mathsf{PD}^2$ Pfair scheduling algorithm [10] is optimal for scheduling IS tasks on an arbitrary number of processors. It prioritizes subtasks on an earliest-pseudo-deadline-first ($\mathsf{EPDF}$) basis, and uses two tie-breaking rules. For the case wherein all task weights are at most 1/2 (our focus here), $\mathsf{PD}^2$ uses one tie-break, $\mathsf{b}(T_i)$, which is defined as $\lceil i/\mathsf{wt}(T) \rceil - \lfloor i/\mathsf{wt}(T) \rfloor$. In a periodic task system, $\mathsf{b}(T_i)$ is 1 if $T_i$'s window overlaps $T_{i+1}$'s, and is 0 otherwise. For example, in all the insets in Fig. 1, $\mathsf{b}(T_i) = 1$ for $1 \leq i \leq 4$ and $\mathsf{b}(T_5) = 0$. If two subtasks have equal deadlines, then a subtask with a b-bit of 1 is favored over one with a b-bit of 0. Further ties are broken arbitrarily. (See [2] for an explanation of this tie-breaking rule.) Notice that, in the absence of IS separations, $\mathsf{r}(T_{i+1}) = \mathsf{d}(T_i) - \mathsf{b}(T_i)$. For example, in Fig. 1(a), $\mathsf{r}(T_2) = \mathsf{d}(T_1) - \mathsf{b}(T_1) = 4 - 1 = 3$, and $\mathsf{r}(T_6) = \mathsf{d}(T_5) - \mathsf{b}(T_5) = 16 - 0 = 16$.

**IS ideal schedule.** Ideal allocations within the IS task model can be defined in much the same way as for periodic tasks [10]; however, we must modify this definition to allow for IS separations. Before continuing, notice that since the total allocation to a task in a given time slot equals the total allocation to all of its subtasks in that slot, $\mathsf{A}(\mathcal{S}, T, t) = \sum_{T_j \in T} \mathsf{A}(\mathcal{S}, T_j, t)$. For example, in Fig. 1(a), $\mathsf{A}(\mathcal{I}, T, 6) = \mathsf{A}(\mathcal{I}, T_1, 6) + \mathsf{A}(\mathcal{I}, T_2, 6) + \mathsf{A}(\mathcal{I}, T_3, 6) + ... = 0 + 2/16 + 3/16 + 0 + ... = 5/16$. Thus, per-task and per-task-set allocations in a schedule $\mathcal{S}$ over an arbitrary interval can be defined by simply defining $\mathsf{A}(\mathcal{S}, T_j, t)$ for an arbitrary subtask $T_j$ and time slot $t$.

For an arbitrary IS task system $\tau$, we let $\mathcal{I}_{\text{IS}}$ denote the ideal schedule of $\tau$. $\mathsf{A}(\mathcal{I}_{\text{IS}}, T_j, u)$ can be defined using an arithmetic expression, but we have opted instead for a more intuitive pseudo-code-based definition in Fig. 2. The ideal IS schedule allocates each subtask $T_j$ some amount of processing time in each slot of its window. For slots other than $\mathsf{r}(T_i)$ and $\mathsf{d}(T_i) - 1$, this allocation is $\mathsf{wt}(T)$. $T_i$'s allocation in slots $\mathsf{r}(T_i)$ and $\mathsf{d}(T_i) - 1$ are adjusted so that **(i)** $T_i$'s entire allocation (across all slots in its window) is one, and **(ii)** $T_i$'s allocation in slot $\mathsf{r}(T_i)$ (resp., $\mathsf{d}(T_i) - 1$) plus $T_{i-1}$'s (resp., $T_{i+1}$'s) allocation in slot $\mathsf{d}(T_{i-1}) - 1$ (resp., $\mathsf{r}(T_{i+1})$) equals $\mathsf{wt}(T)$ (assuming those subtasks exist). Examples of such allocations are given in Fig. 1.

$\mathsf{A}(\mathcal{I}_{\text{IS}}, T_i, t)$
**if** $t < \mathsf{r}(T_i) \vee t \geq \mathsf{d}(T_i))$ **then**
  $\mathsf{A}(\mathcal{I}_{\text{IS}}, T_i, t) := 0$
**else if** $t = \mathsf{r}(T_i)$ **then**
  **if** $i = 1 \vee \mathsf{b}(T_{i-1}) = 0$ **then**
    $\mathsf{A}(\mathcal{I}_{\text{IS}}, T_i, t) := \mathsf{wt}(T)$
  **else**
    $\mathsf{A}(\mathcal{I}_{\text{IS}}, T_i, t) :=$
      $\mathsf{wt}(T) - \mathsf{A}(\mathcal{I}_{\text{IS}}, T_{i-1}, \mathsf{d}(T_{i-1}) - 1)$
  **fi**
**else**
  $\mathsf{A}(\mathcal{I}_{\text{IS}}, T_i, t) :=$
    $\mathsf{min}(\mathsf{wt}(T), 1 - \mathsf{A}(\mathcal{I}_{\text{IS}}, T_i, 0, t))$
**fi**

Figure 2: Pseudo-code defining $\mathsf{A}(\mathcal{I}_{\text{IS}}, T_i, t)$.

**Dynamic task systems.** The leave/join conditions of Srinivasan and Anderson [11] mentioned earlier, and a theorem concerning them, are stated below.

**J:** (*join condition*) A task $T$ can join at time $t$ iff the sum of the weights of all tasks after joining is at most $M$.

**L:** (*leave condition*) Let $T_i$ denote the last-scheduled subtask of $T$. $T$ can leave at time $t$ iff $t \geq \mathsf{d}(T_i) + \mathsf{b}(T_i)$ .

**Theorem 1 ([11]).** $\mathsf{PD}^2$ *correctly schedules any dynamic IS task system satisfying J and L.*

As noted earlier, a task may be reweighted by leaving with its old weight and rejoining with its new weight.

# 3 Adaptable Task Model and Fine-Grained Reweighting

In this section, we introduce the *adaptable IS* (*AIS*) task model, and define corresponding fine-grained reweighting rules, which allow the $\mathsf{PD}^2$ algorithm to schedule each subtask without missing a deadline and to ensure constant drift per weight change. The AIS task model is an extension of IS task model, where the weight of each task $T$, $\mathsf{wt}(T, t)$, is a function of time $t$.

## 3.1 Adaptable Task Model

A task $T$ *changes weight* or *reweights* at time $t + 1$ if $\mathsf{wt}(T, t) \neq \mathsf{wt}(T, t + 1)$. If a task $T$ changes weight at a time $t_c$ between the release and the deadline of some subtask $T_j$, then the following three actions *may* occur: **(i)** if $T_j$ has not been scheduled by $t_c$, then $T_j$ may be "halted" at $t_c$; **(ii)** $\mathsf{r}(T_{j+1})$ *may* be redefined to be less than $\mathsf{d}(T_j) - \mathsf{b}(T_j)$; and **(iii)** if (ii) holds, then the windows of $T_j$ and $T_{j+1}$ may overlap by more than $\mathsf{b}(T_j)$ time slots. (In the IS model defined earlier, every subtask's deadline is at most $\mathsf{b}(T_j)$ time slots after its successor's release.)

The reweighting rules we present at the end of this section state the conditions under which the above actions may occur and the number of slots before $\mathsf{d}(T_j) - \mathsf{b}(T_j)$ that subtask $T_{j+1}$ can be released. If $T_j$ is *halted* before it is scheduled, then it is never scheduled. (Note that a subtask can only be halted if it has not yet been scheduled in the $\mathsf{PD}^2$ schedule.) We use the function $\mathsf{H}(T_j)$ to denote the time at which $T_j$ is halted; if $T_j$ is never halted, then $\mathsf{H}(T_j) = \infty$. For example, consider Fig. 3(a), which depicts a task $T$ that increases its weight from $3/19$ to $2/5$ at time $8$. (The per-slot allocations and the terms "enacted" and "complete" mentioned in the figure are discussed shortly.) In this inset, for $j \in \{1, 3, 4, 5\}$, $\mathsf{H}(T_j) = \infty$, because none of these subtasks halts; however, because $T_2$ is halted at time 8, $\mathsf{H}(T_2) = 8$. Since a subtask is only halted as a result of a reweighting event, if we do not have *a priori* knowledge of such events, then we cannot determine whether a released subtask will be halted in the future. For example, in Fig. 3(a), we have no knowledge when $T_2$ is released at time 6 that it will be halted at time 8.

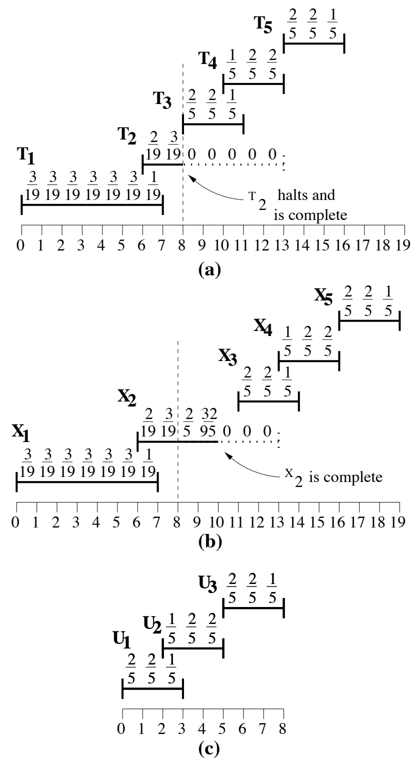


Figure 3: Per-task allocations for a task in an AIS system for **(a)** a task $T$ with an initial weight of $3/19$ that enacts a weight change to $2/5$ and halts at time 8; **(b)** a task $X$ with an initial weight of $3/19$ that enacts a weight change to $2/5$ at time 8 *but does not halt*; **(c)** a periodic task $U$ with weight $2/5$. The dotted window lines indicate that the window that would have existed if the subtask task did not reweight.

**Definition 1 (Initiated and Enacted).** When a task reweights, there can be a difference between when it "initiates" the change and when the change is "enacted." The time at which the change is *initiated* is a user-defined time; the time at which the change is *enacted* is dictated by a set of conditions discussed shortly. We use the *scheduling weight of a task $T$ at time $t$*, denoted $\mathsf{swt}(T, t)$, to represent the "last enacted weight of $T$." Formally, $\mathsf{swt}(T, t)$ equals $\mathsf{wt}(T, u)$, where $u$ is the last time at or before $t$ that a weight change was enacted for $T$. (We assume an initial weight change occurred for $T$ where it initially joined the system.) It is important to note that, *henceforth, we compute subtask deadlines and releases using scheduling weights*. We use $\mathsf{En}(T, t)$ to denote the last time at or before time $t$ that $T$ enacted a weight change, and $\mathsf{Id}(T_j)$ to denote the smallest index $k$ such that $\mathsf{En}(T, \mathsf{r}(T_j)) \leq \mathsf{r}(T_k)$. For example, in Fig. 3(b), $\mathsf{En}(X, t) = 0$, for $0 \leq t < 8$; for $j \in \{1, 2\}$, $\mathsf{Id}(X_j) = 1$; for $t \geq 8$, $\mathsf{En}(X, t) = 8$; and for $j \in \{3, 4, 5\}$, $\mathsf{Id}(X_j) = 3$. Note that if $\mathsf{Id}(T_j) = j$, then $T_j$ is the first subtask of $T$ released after a weight change for $T$ has been enacted.

**Definition 2 (Complete).** If $\mathcal{S}$ is a schedule for the task system $\tau$, then a subtask $T_j$ of $T \in \tau$ is said to have *completed by time $t$ in $\mathcal{S}$* iff $t \geq \mathsf{r}(T_j)$ and one of the following holds: **(i)** $T_j$ has been allocated one quantum by $t$ in $\mathcal{S}$; or **(ii)** $T_j$ is halted by time $t$. As an example, consider the schedule depicted in Fig. 4, which depicts the one-processor $\mathsf{PD}^2$ schedule of two tasks, $T$, with weight 2/5, and $U$, with an initial weight of $2/5$ that increases to $1/2$ at time 3 by halting $U_2$. In this example, $T_1$ completes at time 1 because it is scheduled in slot 0, whereas $U_1$ does not complete until time 2 because it is not scheduled until slot 1. Notice that, since $U_2$ is halted at time 3, it is complete at time 3 even though it is never scheduled. We use the function $\mathcal{D}(\mathcal{S}, T_j)$ to denote the (integral) time at which $T_j$ is complete in $\mathcal{S}$.

For an adaptable task, the *deadline*, *b-bit*, and *release* of a subtask $T_j$, respectively, are defined by Eqns. (2)–(4), where

$z = \mathsf{ld}(T_j) - 1$, $\theta(T_{j+1}) \geq \theta(T_j)$, and $\mathsf{r}(T_1)$ equals the time that $T$ joins the system.

$$\mathsf{d}(T_j) = \mathsf{r}(T_j) + \left\lceil \frac{j-z}{\mathsf{swt}(T, \mathsf{r}(T_j))} \right\rceil - \left\lfloor \frac{j-z-1}{\mathsf{swt}(T, \mathsf{r}(T_j))} \right\rfloor \quad (2)$$

$$\mathsf{b}(T_j) = \left\lceil \frac{j-z}{\mathsf{swt}(T, \mathsf{r}(T_j))} \right\rceil - \left\lfloor \frac{j-z}{\mathsf{swt}(T, \mathsf{r}(T_j))} \right\rfloor \quad (3)$$

$$\mathsf{r}(T_{j+1}) = \mathsf{d}(T_j) - \mathsf{b}(T_j) + (\theta(T_{j+1}) - \theta(T_j)) \quad (4)$$

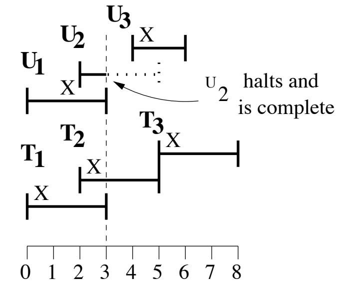


Figure 4: A one-processor $\mathsf{PD}^2$ schedule for two tasks, $T$, with a weight $2/5$, and $U$, which has an initial weight of $2/5$ that increases to $1/2$ at time 3 by halting $U_2$. The $X$'s denote where each subtask is scheduled.

It is important to note that since reweighting events may change a subtask's release time, Eqn. (4) *holds for the subtask $T_{j+1}$ only if a weight change is not enacted for $T$ over the range* $(\mathsf{r}(T_j), \mathsf{d}(T_j)]$. The above equations differ from the earlier definitions of releases, deadlines, and b-bits (given in Sec. 2) in two ways. First, (2) and (3) define the deadline and b-bit of a subtask based on the *scheduling weight* of the task at the time the subtask is *released*. Second, after a task enacts a weight change, its release, deadline, and b-bit are defined as though a new task with the new weight joined the system. (Recall that a subtask $T_j$ is the first released subtask after a weight change is enacted iff $\mathsf{ld}(T_j) = j$.) For example, in Fig. 3(a), after $T$ changes its weight to $2/5$, the subtasks $T_3$–$T_5$ have similar releases, deadlines, and b-bits as the first three subtasks of the task $U$ with weight 2/5 in inset (c).

**Ideal schedules.** In order to state and prove that the reweighting algorithm that we present at the end of this section does not schedule a subtask after its deadline and that it has constant drift, we introduce three notions of an ideal schedule for an AIS task system. The first ideal schedule, $\mathcal{I}_{\mathrm{SW}}$ (used for stating the reweighting rules), allocates each task a share based on its *scheduling weight* in each time slot. The second ideal schedule $\mathcal{I}_{\mathrm{CSW}}$ (used for proving the reweighting rules and drift bounds), is the same as $\mathcal{I}_{\mathrm{SW}}$ except that $\mathcal{I}_{\mathrm{CSW}}$ is "clairvoyant" so that it does not allocate capacity to tasks that will halt. The third ideal schedule, $\mathcal{I}_{\mathrm{PS}}$ (used for proving drift bounds), allocates each task a share based on its *actual weight* in each time slot. We now formally define the allocations to a subtask in $\mathcal{I}_{\mathrm{SW}}$ and $\mathcal{I}_{\mathrm{CSW}}$; $\mathcal{I}_{\mathrm{PS}}$ is considered in the next section.

As with IS tasks, $\mathsf{A}(\mathcal{I}_{\mathrm{SW}}, T_i, t)$ can be defined mathematically, but we opt instead for a pseudo-code-based definition, shown in Fig. 5. There are three differences between the definitions of $\mathsf{A}(\mathcal{I}_{\mathrm{IS}}, T_i, t)$ and $\mathsf{A}(\mathcal{I}_{\mathrm{SW}}, T_i, t)$: in lines 5, 7, and 10, $\mathsf{swt}(T, t)$ is used instead of $\mathsf{wt}(T)$; and in lines 1 and 7, $\mathcal{D}(\mathcal{I}_{\mathrm{SW}}, T_i)$ is used instead of $\mathsf{d}(T_i)$. These two changes account for $T$'s time-varying weight. The final change is that, in line 4, $i = \mathsf{ld}(T_i)$ is used instead of $i = 1$. This change causes the per-slot allocations to $T_z$, where $z = \mathsf{ld}(T_i)$, to equal that of a task that joins the system at $\mathsf{r}(T_z)$. For example, in Fig. 3(a), since $3 = \mathsf{ld}(T_3)$, by lines 4 and 5, $\mathsf{A}(\mathcal{I}_{\mathrm{SW}}, T_3, \mathsf{r}(T_3)) = \mathsf{swt}(T, \mathsf{r}(T_3)) = 2/5$, which is the same per-slot allocation that $U_1$ in Fig. 3(c) receives at time $\mathsf{r}(U_1)$. Before continuing, there are two important issues to note. First, in the absence of reweighting events, $\mathcal{D}(\mathcal{I}_{\mathrm{SW}}, T_j) = \mathsf{d}(T_j)$. Second, when a task is halted via the reweighting rules given below, it is halted in both the $\mathsf{PD}^2$ sched-

```
A(I_SW, T_i, t)
1:  if t < r(T_i) ∨ t ≥ D(I_SW, T_i) then
2:      A(I_SW, T_i, t) := 0
3:  else if t = r(T_i) then
4:      if i = ld(T_i) ∨ b(T_{i−1}) = 0 then
5:          A(I_SW, T_i, t) := swt(T, t)
6:      else
7:          A(I_SW, T_i, t) := swt(T, t)−
                A(I_SW, T_{i−1}, D(I_SW, T_{i−1}) − 1)
8:      fi
9:  else
10:     A(I_SW, T_i, t) :=
            min(swt(T, t), 1 − A(I_SW, T_i, 0, t))
11: fi
```

Figure 5: Pseudo-code defining the $\mathsf{A}(\mathcal{I}_{\mathrm{SW}}, T_i, t)$.

ule and $\mathcal{I}_{\rm SW}$. Since $\mathcal{I}_{\rm SW}$ is not clairvoyant, it will allocate "normally" to a subtask until that subtask halts, after which the subtask's per-slot allocations are zero, as with $T_2$ in Fig. 3(a). Also note that in Fig. 3(b), $X_2$ is complete at time 10, since $\mathsf{A}(\mathcal{I}_{\rm SW}, X_2, 0, 10) = 1$. Several examples of $\mathcal{I}_{\rm SW}$ allocations are given in Fig. 3. Using the definition of $\mathcal{I}_{\rm SW}$, we can simply define $\mathcal{I}_{\rm CSW}$ as follows: $\mathsf{A}(\mathcal{I}_{\rm CSW}, T_i, t) = \mathsf{A}(\mathcal{I}_{\rm SW}, T_i, t)$, if $\mathsf{H}(T_i) = \infty$, and $\mathsf{A}(\mathcal{I}_{\rm CSW}, T_i, t) = 0$, otherwise. For example, in Fig. 3(a) $\mathsf{A}(\mathcal{I}_{\rm SW}, T_i, t) = \mathsf{A}(\mathcal{I}_{\rm CSW}, T_i, t)$ except for $T_2$, where $\mathsf{A}(\mathcal{I}_{\rm CSW}, T_2, t) = 0$ for all $t$.

## 3.2 Reweighting Rules

We now introduce two new fine-grained reweighting rules that improve upon coarse-grained reweighting by changing future subtask releases. It is important to note that in the following rules, for a given subtask $T_j$, the value $\mathsf{d}(T_j)$ is used to determine the scheduling priority of $T_j$ in the $\mathsf{PD}^2$ algorithm and *does not change* once $T_j$ has been released. Furthermore, $\mathcal{D}(\mathcal{I}_{\rm SW}, T_j)$ is used to determine the *release time* of $T_j$'s successor, $T_{j+1}$. As mentioned earlier, the completion time of a subtask cannot be accurately predicted without *a priori* knowledge of weight changes; however, in the reweighting rules below, the completion time of a subtask in $\mathcal{I}_{\rm SW}$ is only used *after* the subtask has completed, and therefore it is well-defined.

Let $\tau$ be a task system in which some task $T$ initiates a weight change from weight $w$ to weight $v$ at time $t_c$. If there does not exist a subtask $T_j$ of $T$ such that $\mathsf{r}(T_j) \leq t_c$, then the weight change is enacted immediately; otherwise, let $T_j$ denote the last-released subtask of $T$. If $\mathsf{d}(T_j) \leq t_c$, then the weight change is enacted at time $\mathsf{max}(t_c, \mathsf{d}(T_j) + \mathsf{b}(T_j))$. In the following rules, we consider the remaining possibility, *i.e.*, that $T_j$ exists and $\mathsf{r}(T_j) \leq t_c < \mathsf{d}(T_j)$. For simplicity, we assume that the first subtask after a weight change by the corresponding task is released as early as possible. This assumption can be removed at the cost of more complex notation.

The choice of which rule to apply depends on whether $T_j$ has been scheduled by $t_c$. We say that $T$ is *ideal-changeable at time $t_c$ from weight $w$ to $v$* if $T_j$ is scheduled before $t_c$, and otherwise is *omission-changeable at time $t_c$ from $w$ to $v$*. Because $T$ initiates its weight change at $t_c$, $\mathsf{wt}(T, t_c) = v$ holds; however, $T$'s scheduling weight does not change until the weight change has been *enacted*, as specified in the rules below. Note that, if $t_c$ occurs between the initiation and enaction of a previous reweighting event of $T$, then the previous event is skipped, *i.e.*, treated as if it had not occurred. As discussed later, any "error" associated with skipping a reweighting event like this is accounted for when determining drift.

**Rule O:** If $T$ is omission-changeable at time $t_c$ from weight $w$ to $v$ and $j > 1$, then at time $t_c$, subtask $T_j$ is halted and at time $\mathsf{max}(t_c, \mathcal{D}(\mathcal{I}_{\rm SW}, T_{j-1}) + \mathsf{b}(T_{j-1}))$, $T$'s weight change is enacted, and a new subtask is released. If $j = 1$, then at time $t_c$, $T_j$ is halted, $T$'s weight change is enacted, and a new subtask is released.

**Rule I:** If $T$ is ideal-changeable at time $t_c$ from weight $w$ to $v$, then one of two actions is taken: **(i)** if $v > w$, then the weight change is *immediately* enacted, and at time $\mathcal{D}(\mathcal{I}_{\rm SW}, T_j) + \mathsf{b}(T_j)$, a new subtask is released for $T$; **(ii)** otherwise, the weight change is enacted at time $\mathcal{D}(\mathcal{I}_{\rm SW}, T_j) + \mathsf{b}(T_j)$, at which time a new subtask is released.

Both rules are extensions of the leave/join rules L and J given earlier in Sec. 2. However, the rules above exploit the specific circumstances that occur when a task changes its weight to "short circuit" rules L and J, so that reweighting is accomplished faster. By rule L, $T$ can leave at time $\mathsf{d}(T_k) + \mathsf{b}(T_k)$, where $T_k$ is its last-scheduled subtask. We can easily extend rule L to show

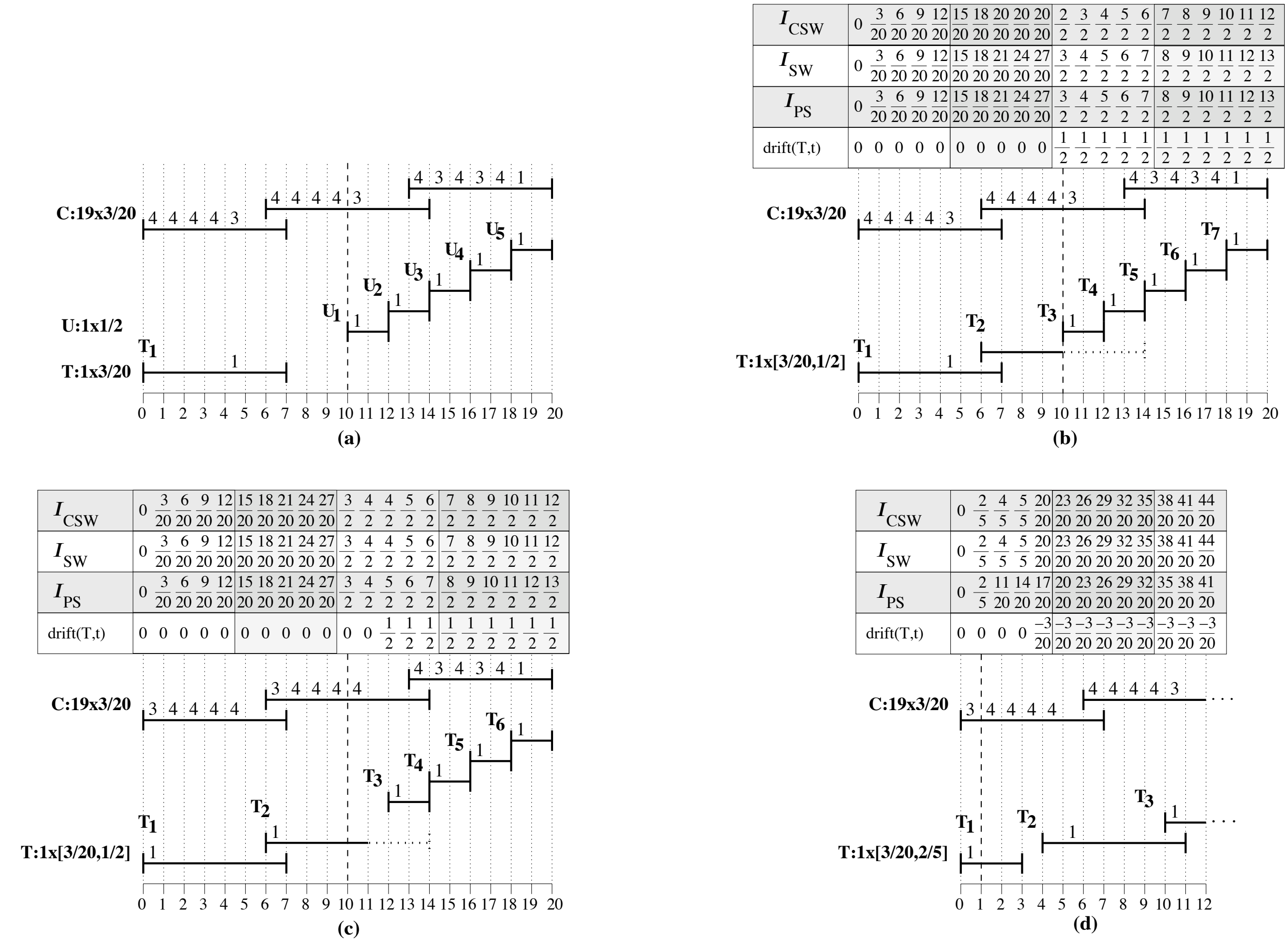


Figure 6: A four-processor system consisting of a set $C$ of 19 tasks of weight 3/20 each, and a task $T$ of weight 3/20, and in (a), a task $U$ of weight 1/2. In (a) and (b), all ties are broken in favor of tasks from $C$, and in (c), all ties are broken in favor of task $T$. The notation $T$:$[u, v]$ denotes that $T$'s weight ranges over $[u, v]$. The windows of the various task groups are shown together. The numbers in each slot in a window denote the number of tasks from each set scheduled in that slot. (Similar notation is used in later figures.) In insets (b)–(d), $T$'s allocation up to time $t$ in $\mathcal{I}_{\rm SW}$, $\mathcal{I}_{\rm CSW}$, and $\mathcal{I}_{\rm PS}$ as well as $T$'s drift are labeled at the top. **(a)** $T$ leaves at time 8 and $U$ joins at time 10. **(b)** $T$ reweights to 1/2 via rule O at slot 10. (Notice that $T_2$ is halted at $t_c$ and is never scheduled.) **(c)** $T$ reweights to 1/2 via rule I at slot 10. **(d)** $T$ has an initial weight of $2/5$ that decreases to $3/20$ via rule I at time 1. Rule O or I is applied depending on whether $T_2$ (in (b) and (c)) or $T_1$ (in (d)) is scheduled prior to slot 10 (in (b) and (c)) or 1 (in (d)).

that $T$ can leave at time $\mathcal{D}(\mathcal{I}_{\rm SW}, T_k) + \mathsf{b}(T_k)$. If task $T$ (as defined above) is omission-changeable, then its subtask $T_j$ has not been scheduled by time $t_c$. Such a task can be viewed as having "left" the system at time $\max(t_c, \mathcal{D}(\mathcal{I}_{\rm SW}, T_{j-1}) + \mathsf{b}(T_{j-1}))$, in which case, it can rejoin the system immediately. For example, in Fig. 6(a), task $T$ of weight 3/20 leaves at time 8 and task $U$ of weight 1/2 joins at time 10. In Fig. 6(b), task $T$ increases its weight from 3/20 to 1/2 via rule O. Note that, in Fig. 6(b), $T$ behaves as if it leaves at time 8 and rejoins at time 10 with its new weight.

If $T$ is ideal-changeable, then by rule L, it may "leave and rejoin" with a new weight at time $\mathsf{d}(T_j) + \mathsf{b}(T_j)$ (*i.e.*, its weight change can be enacted at $\mathsf{d}(T_j) + \mathsf{b}(T_j)$). However, if $\mathcal{D}(\mathcal{I}_{\rm SW}, T_j) < \mathsf{d}(T_j)$, then $T$ may "leave and rejoin" with a new weight at $\mathcal{D}(\mathcal{I}_{\rm SW}, T_j) + \mathsf{b}(T_j)$. ("Ideal-changeable" refers to the fact that the time at which the subtask can leave and rejoin is based on that subtask's allocations in an ideal schedule.) For example, in Fig. 6(c), task $T$ increases its weight from 3/20 to 1/2 at time 10 via rule I. Since at time 11 the total allocation to $T_2$ in $\mathcal{I}_{\rm SW}$ is one, $\mathcal{D}(\mathcal{I}_{\rm SW}, T_2) = 11$. Hence, by rule I, $T$ can "leave" at time $\mathcal{D}(\mathcal{I}_{\mathrm{SW}}, T_2) + \mathsf{b}(T_2) = 12$, which is two time units earlier than its deadline. In Fig. 6(d), task $T$ decreases its weight from $2/5$ to $3/20$ at time one. Since $T$ decreases its weight, by rule I, this weight change is not enacted until time $\mathcal{D}(\mathcal{I}_{\mathrm{SW}}, T_1) + \mathsf{b}(T_1)$. Since no weight change is enacted before $\mathsf{d}(T_1)$, $\mathsf{d}(T_1) = \mathcal{D}(\mathcal{I}_{\mathrm{SW}}, T_1) = 3$. Thus, by rule I, $T$ "leaves" at time $\mathcal{D}(\mathcal{I}_{\mathrm{SW}}, T_1) + \mathsf{b}(T_1) = 4$. Notice that the difference in rule I between cases (i) and (ii) is that, when a task increases its weight, the weight change is immediately enacted, whereas when a task decreases its weight, its weight change is not enacted until time $\mathcal{D}(\mathcal{I}_{\mathrm{SW}}, T_j) + \mathsf{b}(T_j)$. Thus, $T$'s scheduling weight is redefined at different times.

Throughout this paper we use $\mathsf{PD}^2$-$\mathsf{OI}$ (respectively, $\mathsf{PD}^2$-$\mathsf{LJ}$) to refer to reweighting via rules O and I (resp., the leave/join rules L and J) under $\mathsf{PD}^2$. Since these rules change the ordering of a task in the priority queues that determine scheduling, the time complexity for reweighting one task is $O(logN)$, where $N$ is the number of tasks in the system.

# 4 Scheduling Correctness and Drift Bounds

In the prior section, we used $\mathcal{I}_{\mathrm{SW}}$ to determine the release times of future subtasks. Notice that $\mathcal{I}_{\mathrm{SW}}$ and the $\mathsf{PD}^2$-$\mathsf{OI}$ schedule treat halted subtasks differently. Specifically, $\mathcal{I}_{\mathrm{SW}}$ will "partially" allocate a halted subtask, whereas $\mathsf{PD}^2$-$\mathsf{OI}$ will never schedule a halted subtask. Because of this difference, it is convenient, when proving correctness and drift bounds, to slightly alter $\mathcal{I}_{\mathrm{SW}}$ by eliminating halted subtasks. Therefore, we use $\mathcal{I}_{\mathrm{CSW}}$ instead, since $\mathcal{I}_{\mathrm{CSW}}$ does not allocate any capacity to any halted subtask. In this section, we discuss the scheduling correctness of $\mathsf{PD}^2$-$\mathsf{OI}$ (the full proof can be found in an appendix), formally define drift, and discuss the drift bounds of $\mathsf{PD}^2$-$\mathsf{LJ}$, $\mathsf{PD}^2$-$\mathsf{OI}$, and any $\mathsf{EPDF}$ reweighting algorithm.

We first show that initiating multiple reweighting events without enacting them does not increase the time of the next weight-change enactment. We show this by proving the following.

**(C)** If $T$ initiates two weight-change events at $t_c$ and $t'_c$, where $t_c < t'_c$ and $t'_c < t_e$, and $t_e$ and $t'_e$ denote the time the changes initiated at $t_c$ and $t'_c$, respectively, would have been enacted in the absence of other reweighting events, then $t'_e \leq t_e$.

**Proof of (C).** Assume that $t_c$, $t_e$, $t'_c$, and $t'_e$ are as defined in (C). Notice that all types of reweighting events initiated at $t_c$ except for ideal-changeable decreasing-weight events and (some) omission-changeable events are enacted within one quantum. Thus, we assume that $T$ is either omission-changeable (and not immediately enacted) or decreasing-weight ideal-changeable at $t_c$. Before continuing, notice that if $\mathsf{d}(T_j) \leq t'_c$, where $T_j$ is as defined in rules O and I, then the change initiated at $t'_c$ is enacted by $t'_c + 1$. Since $t'_c < t_e$, this implies that $t'_e \leq t_e$. Thus, we assume in the rest of the proof that $t'_c < \mathsf{d}(T_j)$. We first consider the case wherein $T$ is omission-changeable at $t_c$ and this change is not immediately enacted. In this case, the change initiated at $t_c$ is enacted at time $t_e = \mathcal{D}(\mathcal{I}_{\mathrm{SW}}, T_{j-1}) + \mathsf{b}(T_{j-1})$. Since $T$ is omission-changeable at $t_c$, it is halted at $t_c$ and no successor subtask can be released until the change initiated at $t_c$ (or a future change) has been enacted. Hence, since $t'_c < t_e$, $T_{j+1}$ is not released until after $t'_c$. Since $\mathsf{r}(T_j) < t_c < t'_c < \mathsf{d}(T_j)$, $T_j$ is the last-released subtask of $T$ at or before $t'_c$. Because $T_j$ was halted at $t_c$, $T$ is therefore omission-changeable at $t'_c$. Thus, by rule O, the change initiated at $t'_c$ is enacted at time $t'_e = \mathsf{min}(t'_c, \mathcal{D}(\mathcal{I}_{\mathrm{SW}}, T_{j-1}) + \mathsf{b}(T_{j-1}))$. Thus, $t'_e = t_e$. We next consider the case wherein $T$ is decreasing-weight ideal-changeable at $t_c$. By rule I, such a change initiated at $t_c$ will be enacted at time $t_e = \mathcal{D}(\mathcal{I}_{\mathrm{SW}}, T_j) + \mathsf{b}(T_j)$. Since $T$

is ideal-changeable at $t_c$, no subtask can be released until the change which was initiated at $t_c$ (or a future change) has been enacted. Hence, since $t'_c < t_e$, $T_{j+1}$ is not released until after the change at $t'_c$ is initiated. Since $\mathsf{r}(T_j) < t_c < t'_c < \mathsf{d}(T_j)$, $T_j$ is the last-released subtask of $T$ at or before $t'_c$. Because $T_j$ is scheduled before $t'_c$, $T$ is ideal-changeable at $t'_c$. If the event at $t'_c$ is a decreasing-weight event, then by rule I, it is enacted at time $t'_e = \mathcal{D}(\mathcal{I}_{\mathrm{SW}}, T_j) + \mathsf{b}(T_j) = t_e$; if it is an increasing-weight event, then $t'_e = t'_c < t_e$. □

When Srinivasan and Anderson [11] proved the scheduling correctness for $\mathsf{PD}^2$-$\mathsf{LJ}$ for an IS task system, they were able to utilize the fact that the windows for any subtask $T_j$ and its successor $T_{j+1}$ do not "overlap" by more than $\mathsf{b}(T_j)$ quanta, *i.e.*, $\mathsf{d}(T_j) - \mathsf{b}(T_j) \leq \mathsf{r}(T_{j+1})$. However, this property can be weakened without affecting most of their proof, so that their proof can be applied to an AIS task system. Specifically, their proof can be used to establish the scheduling correctness of $\mathsf{PD}^2$-$\mathsf{OI}$ for any AIS task system $\tau$, if the following properties hold. (In these properties, we denote the $\mathsf{PD}^2$-$\mathsf{OI}$ schedule of $\tau$ as $\mathcal{S}$.)

**(W)** For any time $t$, $\sum_{T \in \tau} \mathsf{swt}(T, t) \leq M$, where $M$ is the number of processors.

**(V)** For the subtasks $T_i$ and $T_{i+1}$, if $\mathsf{d}(T_i) - \mathsf{b}(T_i) > \mathsf{r}(T_{i+1})$, then $\mathcal{D}(\mathcal{I}_{\mathrm{CSW}}, T_i) \leq \mathsf{r}(T_{i+1})$ and $\mathcal{D}(\mathcal{S}, T_i) \leq \mathsf{r}(T_{i+1})$.

Since (W) can be satisfied by policing weight-change requests, we focus our attention on showing that $\mathcal{S}$ and $\mathcal{I}_{\mathrm{CSW}}$ satisfy (V).

**Proof of (V).** Before we begin, notice that by the rules O and I, when $T$ initiates a weight change at time $t_c$, it is enacted no later than time $\mathsf{r}(T_k)$, where $T_k$ is the next-released subtask of $T$. (By (C), no sequence of reweighting events can delay the next weight-change enactment after a weight change has been initiated, and by the rules I and O, a subtask is released within one quantum of a weight change enactment. Thus, $T_k$ is eventually released.) Hence, if a weight change is enacted over the range $(\mathsf{r}(T_\ell), \mathsf{d}(T_\ell)]$, then that change must have been initiated over the range $[\mathsf{r}(T_\ell), \mathsf{d}(T_\ell)]$.

Let $T_i$ be some subtask such that $\mathsf{d}(T_i) - \mathsf{b}(T_i) > \mathsf{r}(T_{i+1})$. By the definition of a subtask release, if $\mathsf{d}(T_i) - \mathsf{b}(T_i) > \mathsf{r}(T_{i+1})$, then $T$ enacted a weight change at $t_e \in (\mathsf{r}(T_i), \mathsf{r}(T_{i+1})]$; otherwise, we would have $\mathsf{d}(T_i) - \mathsf{b}(T_i) \leq \mathsf{r}(T_{i+1})$, by Eqn. (4). Since a weight change is enacted in the range $(\mathsf{r}(T_i), \mathsf{d}(T_i)]$, (as established above) a change must have been initiated in the range $[\mathsf{r}(T_i), \mathsf{d}(T_i)]$. Thus, since a weight change is initiated before it is enacted (and by assumption $t_e \leq \mathsf{r}(T_{i+1}) < \mathsf{d}(T_i) - \mathsf{b}(T_i) \leq \mathsf{d}(T_i)$) a change must have been initiated in $[\mathsf{r}(T_i), \mathsf{r}(T_{i+1})]$. Without loss of generality, let $t_c$ be the earliest time in this range that $T$ initiates a weight change.

Since $\mathsf{r}(T_i) \leq t_c \leq \mathsf{r}(T_{i+1}) < \mathsf{d}(T_i)$, $T$ is either omission- or ideal-changeable at $t_c$. We now consider these two cases. If at $t_c$, $T$ is omission-changeable, then by rule O, $T_i$ is halted at $t_c$. In this case, $T_i$ is complete by $t_c \leq t_e \leq \mathsf{r}(T_{i+1})$ in both the $\mathcal{S}$ and $\mathcal{I}_{\mathrm{CSW}}$. If $T$ is ideal-changeable at $t_c$, then $T_i$ has been scheduled in $\mathcal{S}$ before $t_c$, and hence, $T_i$ is complete by $t_c$ in $\mathcal{S}$. Furthermore, in this case $T_{i+1}$ is not released until time $\mathcal{D}(\mathcal{I}_{\mathrm{CSW}}, T_i) + \mathsf{b}(T_i)$. □

By (V), it is possible to use Srinivasan and Anderson's correctness proof for $\mathsf{PD}^2$-$\mathsf{LJ}$ [11] to prove the following theorem.

**Theorem 2.** *Under* $\mathsf{PD}^2$-$\mathsf{OI}$*, no subtask is scheduled after its scheduling deadline, provided that* (*W*) *holds.*

## 4.1 Drift

We now turn our attention to the issue of measuring drift under $\mathsf{PD}^2$-$\mathsf{OI}$. In order to measure the drift of a task system

$\tau$, we compare $\mathcal{I}_{\rm CSW}$ to an ideal schedule in which weight changes are enacted instantaneously. Under the *ideal processor sharing* ($\mathcal{I}_{\rm PS}$) schedule, at each instant $t$, each task $T$ in $\tau$ is allocated a share equal to its weight $\mathsf{wt}(T, t)$. Hence, over the interval $[t_1, t_2)$, the task $T$ is allocated $\mathsf{A}(\mathcal{I}_{\rm PS}, T, t_1, t_2) = \int_{t_1}^{t_2} \mathsf{wt}(T, u)du$ time. (For the remainder of this section, we assume that every subtask in $T$ is released as early as possible. This assumption can be removed at the cost of more complex notation. If we did not make this assumption, then the allocation function for $\mathcal{I}_{\rm PS}$ would equal zero between active subtasks.) $\mathcal{I}_{\rm PS}$ is similar to $\mathcal{I}_{\rm SW}$ and $\mathcal{I}_{\rm CSW}$, with three major exceptions: **(i)** tasks in $\mathcal{I}_{\rm PS}$ continually receive allocations, whereas tasks in $\mathcal{I}_{\rm SW}$ and $\mathcal{I}_{\rm CSW}$ receive allocations only at quantum boundaries; **(ii)** under $\mathcal{I}_{\rm PS}$, each task receives an allocation equal to its *weight*, whereas under $\mathcal{I}_{\rm SW}$ and $\mathcal{I}_{\rm CSW}$, each task receives allocations according to its *scheduling weight*; and **(iii)** the total allocation each task receives in $\mathcal{I}_{\rm SW}$ and $\mathcal{I}_{\rm CSW}$ is calculated based on the releases and completion times of its active subtasks, whereas allocations in $\mathcal{I}_{\rm PS}$ are independent of subtask releases and completion times. Hence, even if all active subtasks of a given task are halted, $\mathcal{I}_{\rm PS}$ still allocates capacity to that task. For example, consider Fig. 7, which depicts the allocations in the schedules $\mathcal{I}_{\rm CSW}$ and $\mathcal{I}_{\rm PS}$ (insets (a) and (b), resp.) to a task $X$ that has an initial weight of $3/19$ that increases to $2/5$ (via rule I) at time 8. Notice that in $\mathcal{I}_{\rm PS}$ over the range $[9, 11)$, $X$ receives an allocation equal to its weight at every instant (for a total allocation of $4/5$ over $[9, 11)$). Compare this to $\mathcal{I}_{\rm CSW}$, in which $X$ receives only an allocation of $32/95$ over the same range.

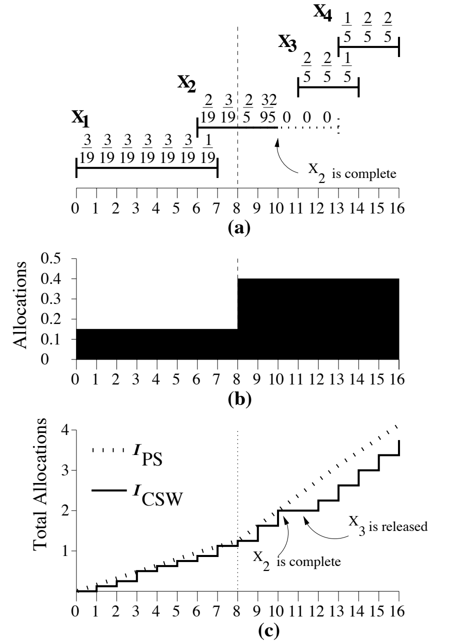


Figure 7: Allocations for a task $X$ with an initial weight of $3/19$ that changes to $2/5$. **(a)** The value of $\mathsf{A}(\mathcal{I}_{\rm CSW}, X_j, u)$ for each slot and subtask. **(b)** The allocations to $X$ in $\mathcal{I}_{\rm PS}$ at each instant. **(c)** The total allocations to $X$ in $\mathcal{I}_{\rm CSW}$ and $\mathcal{I}_{\rm PS}$.

For most real-time scheduling algorithms, the difference between the ideal and actual allocations a task receives lies within some bounded range centered at zero (that is, *lag bounds* are maintained). For example, under $\mathsf{PD}^2$ (*i.e.*, $\mathsf{PD}^2$-$\mathsf{OI}$ without weight changes), the difference between the ideal and actual allocations for a task lies within $(-1, 1)$. When a weight change occurs, the same range is maintained except that it may be centered at a different value. This lost allocation is called *drift*. In general, a task's drift per reweighting event will be non-negative (non-positive) if it increases (decreases) its weight. Let $\mathcal{S}$ denote a $\mathsf{PD}^2$-$\mathsf{OI}$ schedule of some task system $\tau$. Since Thm. 2 established that no subtask misses its deadline, and neither $\mathcal{S}$ nor $\mathcal{I}_{\rm CSW}$ schedules any halted subtasks, $\mathsf{A}(\mathcal{S}, T, 0, t)$ differs from $\mathsf{A}(\mathcal{I}_{\rm CSW}, T, 0, t)$ by $\pm 1$. Hence, we can bound the drift that a task $T$ incurs under $\mathsf{PD}^2$-$\mathsf{OI}$ up to time $t$ by comparing the total allocations to $T$ in $\mathcal{I}_{\rm PS}$ to that in $\mathcal{I}_{\rm CSW}$ up to time $t$. Formally, under $\mathsf{PD}^2$-$\mathsf{OI}$, the *drift of a task $T$ is defined*[1] as

$$\mathsf{drift}(T, t) = \mathsf{A}(\mathcal{I}_{\rm PS}, T, 0, u) - \mathsf{A}(\mathcal{I}_{\rm CSW}, T, 0, u), \tag{5}$$

where $u$ is defined as follows: if $t < \mathsf{r}(T_1)$, then $u = t$; otherwise, $u = \mathsf{r}(T_i)$, where $T_i$ is the last-released subtask of $T$ at or before $t$ such that $\mathsf{Id}(T_i) = i$. For example, in Fig. 6(b), the drift of task $T$ at time $t = 9$ is $\mathsf{A}(\mathcal{I}_{\rm PS}, T, 0, \mathsf{r}(T_1)) -$

[1]The definition of drift presented in (5) is designed specifically for $\mathsf{EPDF}$ systems. This concept can be more generally defined to pertain to other systems like global $\mathsf{EDF}$ and partitioning schemes.

$\mathsf{A}(\mathcal{I}_{\mathrm{CSW}}, T, 0, \mathsf{r}(T_1)) = 0 - 0 = 0$, whereas at time $t = 10$, the drift of $T$ is $\mathsf{A}(\mathcal{I}_{\mathrm{PS}}, T, 0, \mathsf{r}(T_3)) - \mathsf{A}(\mathcal{I}_{\mathrm{CSW}}, T, 0, \mathsf{r}(T_3)) = 3/2 - 1 = 1/2$. Notice that since $T_2$ is halted at time 10, $\mathsf{A}(\mathcal{I}_{\mathrm{CSW}}, T_2, 0, 10) = 0$. We say that a reweighting algorithm is *fine-grained* iff there exists some constant value $c$ such that the drift per weight change is less than $c$. We say that a reweighting algorithm is *coarse-grained* otherwise.

We now prove that $\mathsf{PD}^2$-$\mathsf{LJ}$ is not fine-grained. Consider the four-processor system depicted in Fig. 8. This system consists of a set $A$ of 35 tasks with weight 1/10 and a task $T$ with weight 1/10 that increases to 1/2 at time 4. By rule L, $T$ cannot "leave" until time 10. Hence, the change is not enacted until time 10. Thus, over the range $[4, 10)$, $T$ receives a $1/10$ per-slot allocation in $\mathcal{I}_{\mathrm{CSW}}$ and $1/2$ in $\mathcal{I}_{\mathrm{PS}}$. Hence, $T$'s drift reaches a value of $24/10$ at time 10. This example can be generalized to generate any value of drift for $T$, by decreasing its initial weight. Under $\mathsf{PD}^2$-$\mathsf{LJ}$, such a task cannot change its weight until the end of the first window generated by its initial weight. Hence, by decreasing the weight of $T$ to $1/(2(c+1))$, we have $\mathsf{drift}(T, \mathsf{d}(T_1)) \geq c$. The theorem below follows.

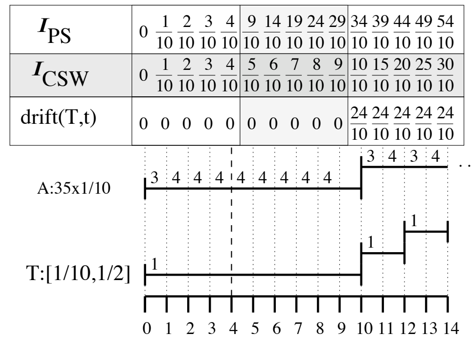


Figure 8: The $\mathsf{PD}^2$-$\mathsf{LJ}$ schedule of a four-processor system with a set $A$ of 35 tasks with weight 1/10 and a task $T$ with weight 1/10 that increases to 1/2 at time 4. The allocations of $T$ in $\mathcal{I}_{\mathrm{CSW}}$ and $\mathcal{I}_{\mathrm{PS}}$ and its drift are labeled.

**Theorem 3.** $\mathsf{PD}^2$-$\mathsf{LJ}$ *is not fine-grained.*

Next, we show that any $\mathsf{EPDF}$ scheduling algorithm incurs some drift. This follows from the two-processor counterexample depicted in Fig. 9. This system consists of a set $A$ of 10 tasks with weight 1/7 that leave at time 7, a set $B$ of two tasks with weight 1/6 that leave at time 6, a set $C$ of two tasks with weight 1/14 that join at time 6, and a set $D$ of five tasks with a weight of 1/21 that increases to 1/3 at time 7. With subtask deadlines defined by $\mathcal{I}_{\mathrm{PS}}$, the deadline for each task in set $D$ changes at time 7 from 21 to 9. The tasks in $D$ have an original deadline of 21 because that is the projected time at which their $\mathcal{I}_{\mathrm{PS}}$ allocations will equal one if their weights do not change. These tasks change their deadlines to 9 at time 7 because the new weight, 1/3, changes the projected time by which their $\mathcal{I}_{\mathrm{PS}}$ allocations will equal one to time 9. Hence, any $\mathsf{EPDF}$ algorithm will not schedule the tasks in $D$ until time 7. As a result, a deadline is missed. Notice that any $\mathsf{EPDF}$ algorithm would need to use projections for determining subtask deadlines if we assume no prior knowledge of weight changes. To prevent a deadline miss, the lag-bound range must be shifted, thus incurring drift. The theorem below follows.

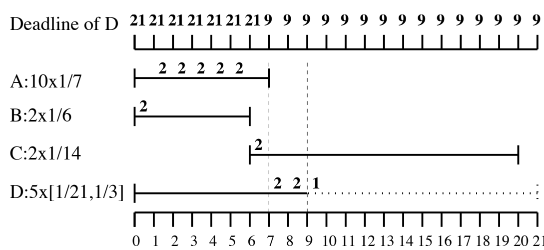


Figure 9: A two-processor system consisting of a set $A$ of 10 tasks with weight 1/7 that leave at time 7, a set $B$ of two tasks with weight 1/6 that leave at time 6, a set $C$ of two tasks with weight 1/14 that join at time 6, and a set $D$ of five tasks with a weight of 1/21 that increases to 1/3 at time 7. The projected deadlines of tasks in $D$ based on their true ideal allocations are labeled above. Notice that a task in $D$ misses its deadline at time 9 since its "true" deadline is unknown until time 7.

**Theorem 4.** *All* $\mathsf{EPDF}$ *algorithms can incur non-zero drift per reweighting event.*

Finally, we show that $\mathsf{PD}^2$-$\mathsf{OI}$ is fine grained. By the definition of drift, in order to prove that $\mathsf{PD}^2$-$\mathsf{OI}$ is fine-grained, we merely need to consider the window placement of a task after it is reweighted. Suppose that a task $T$ initiates a weight change at $t_c$. Let $t_e$ be the next time at which $T$ enacts a change at or after $t_c$, and let $T_j$ be the last-released subtask of $T$ at or before

$t_c$, if $T_j$ exists. (If $T_j$ does not exist, then $T$'s drift does not change.) We now show that if the change initiated at $t_c$ is enacted at $t_e$, then the added drift is bounded by showing that the maximal absolute difference between $\mathcal{I}_{\text{CSW}}$ and $\mathcal{I}_{\text{PS}}$ in allocations to $T$ over the interval $[t_c, \mathsf{r}(T_{j+1}))$ is at most two. Notice that such a result implies that if $T$ were to initiate a change at time $t'_c$ such that $t_c < t'_c \leq t_e$, then the absolute difference in allocation between $\mathcal{I}_{\text{CSW}}$ and $\mathcal{I}_{\text{PS}}$ to $T$ over the interval $[t_c, t'_c)$ is at most two. This implies that the absolute value of the added drift per reweighting event is at most two, even for those events that are "cancelled" by future reweighting events that occur before any change is enacted.

If $\mathsf{d}(T_j) \leq t_c$, then the weight change is enacted within one quantum. Since the maximal weight of a task is $1/2$, the maximal increase in the absolute value of drift in such a case is $1/2$. If $T_j$ exists and $\mathsf{r}(T_j) \leq t_c < \mathsf{d}(T_j)$, then $T$ is either omission- or flow-changeable. If $T$ changes its weight via rule O, then the resulting allocation error is at most two quanta. One quantum of the error can be incurred because $T_j$ is halted at $t_c$, resulting in an allocation of up to one subtask is "lost." For example, in Fig. 6(b), $\mathsf{A}(\mathcal{I}_{\text{CSW}}, T_2, 0, 10) = 1/2$ quanta is "lost" when $T$ initiates and enacts a weight change at time 10. The second quantum of error can be incurred because the change $T$ initiated at $t_c$ may not be enacted until time $\mathsf{max}(t_c, \mathcal{D}(\mathcal{I}_{\text{CSW}}, T_{j-1}) + \mathsf{b}(T_{j-1}))$. By (V), if a change is enacted in the range $[\mathsf{r}(T_{j-1}), \mathsf{d}(T_{j-1})]$, then $\mathcal{D}(\mathcal{I}_{\text{CSW}}, T_{j-1}) - \mathsf{r}(T_j) \leq \mathsf{b}(T_{j-1})$. Further, by Eqn. (4), if no change is enacted in the range $[\mathsf{r}(T_{j-1}), \mathsf{d}(T_{j-1})]$, then $\mathcal{D}(\mathcal{I}_{\text{CSW}}, T_{j-1}) - \mathsf{r}(T_j) \leq \mathsf{b}(T_{j-1})$. Thus, since $\mathsf{r}(T_j) \leq t_c$, the range $[t_c, \mathsf{max}(t_c, \mathcal{D}(\mathcal{I}_{\text{CSW}}, T_{j-1}) + \mathsf{b}(T_{j-1})))$ has a length of at most two. Hence, the change initiated at $t_c$ may not be enacted for two quanta, and since the maximal weight for any task is $1/2$ the $\mathcal{I}_{\text{PS}}$ allocations may "get ahead" (if $T$ increases its weight at $t_c$) or "fall behind" (if $T$ decreases its weight at $t_c$) by $2 \cdot 1/2 = 1$ quantum.

If $T$ increases its weight at $t_c$ via rule I, then the weight change is immediately enacted and $T_{j+1}$ is released at time $\mathcal{D}(\mathcal{I}_{\text{CSW}}, T_j) + \mathsf{b}(T_j)$. Since the weight change is immediately enacted, the only period of time during which $T$ receives less allocation in $\mathcal{I}_{\text{CSW}}$ than in $\mathcal{I}_{\text{PS}}$ is between $\mathcal{D}(\mathcal{I}_{\text{CSW}}, T_j) - 1$ and $\mathsf{r}(T_{j+1})$. Since $\mathsf{r}(T_{j+1}) = \mathcal{D}(\mathcal{I}_{\text{CSW}}, T_j) + \mathsf{b}(T_j)$, the length of this interval is at most two. Since the maximal weight of a task is $1/2$, the maximal increase in drift is $2 \cdot 1/2$. For example, in Fig. 6(c), $\mathsf{A}(\mathcal{I}_{\text{CSW}}, T, 10, 12) = \mathsf{A}(\mathcal{I}_{\text{CSW}}, T, 0, 12) - \mathsf{A}(\mathcal{I}_{\text{CSW}}, T, 0, 10) = 4/2 - 3/2 = 1/2 < 1 = 2 - 1 = \mathsf{A}(\mathcal{I}_{\text{PS}}, T, 0, 12) - \mathsf{A}(\mathcal{I}_{\text{PS}}, T, 0, 10) = \mathsf{A}(\mathcal{I}_{\text{PS}}, T, 10, 12)$. If $T$ decreases its weight at $t_c$ via rule I, then $T_{j+1}$ is released at time $\mathcal{D}(\mathcal{I}_{\text{CSW}}, T_j) + \mathsf{b}(T_j)$. Since $T$ decreases its weight, over the range $[t_c, \mathcal{D}(\mathcal{I}_{\text{CSW}}, T_j))$, $T$ is allocated at most one quantum more in $\mathcal{I}_{\text{CSW}}$ than in $\mathcal{I}_{\text{PS}}$. Furthermore, over the range $[\mathcal{D}(\mathcal{I}_{\text{CSW}}, T_j), \mathcal{D}(\mathcal{I}_{\text{CSW}}, T_j) + \mathsf{b}(T_j))$, $T$ is allocated at most $1/2$ quanta more in $\mathcal{I}_{\text{PS}}$ than in $\mathcal{I}_{\text{CSW}}$, since the length of this range is one and the maximal weight of a task is $1/2$. Thus, the maximal possible decrease in drift is one and the maximal possible increase in drift is $1/2$. For example, in Fig. 6(d), the drift incurred by changing the weight of $T$ from $2/5$ to $3/20$ is $-3/20$, *i.e.*, $\mathsf{drift}(T, t) = -3/20$, where $t \geq 4$.

**Theorem 5.** *The absolute value of the per-event drift under* $\mathsf{PD}^2$-$\mathsf{OI}$ *for each task is at most two.*

# 5 Experimental Results

The results of this paper are part of a longer-term project on adaptive real-time allocation in which Whisper, described earlier, will be used as a test application. In this section, we provide an extensive simulation of Whisper. Unfortunately, at this point in time, it is not feasible to produce experiments involving a real implementation of Whisper for several reasons. First, the existing Whisper system is single threaded (and non-adaptive) and consists of several thousand lines of code. All of this code has to be re-implemented as a multi-threaded system, which is a nontrivial task. Indeed, because of this, it is *essential* that we first understand the algorithmic tradeoffs involved in adapting tasks on multiprocessor real-time systems. The purpose of this paper, as well as the two related ones we have written on adaption under partitioning [4] and global EDF [7], is to explore these tradeoffs. Additionally, this paper is only concerned with scheduling methods that facilitate adaption—we have *not* addressed the issue of devising mechanisms for determining *how* and *when* the system should adapt. Such mechanisms will be based on issues involving virtual-reality systems that are well beyond the scope of this paper. For these reasons, we have chose to evaluate PD$^2$-OI via simulation.

Specifically, we present a simulated implementation of Whisper on a four-processor system, with 2.7 GHz processors and a 1-ms quantum. The system was simulated for 10 s, with a sampling frequency of 1,000 Hz for each tracked object. Whisper tracks users through a system of speakers attached to users and microphones attached to the ceiling. Each speaker emits a unique "white noise" signal that is received by the microphones. As depicted in Fig. 10, we simulated three speakers (one per object) revolving around a 5-cm pole in a 1m $\times$ 1m room with a microphone in each corner. The pole creates potential occlusions. Whisper is able to compute the time-shift between the transmitted and received versions of the sound by performing a *correlation* calculation on the most recent set of samples. As the distance between the speaker and microphone changes, so do the number of correlation computations necessary to correctly track the speaker. This distance is (obviously) impacted by a speaker's movement, but is also lengthened when an occlusion is caused by the pole. The range of weights of each task was determined (as a function of a tracked object's position) by implementing and timing the basic computation of the correlation algorithm (an accumulate-and-multiply operation) on a testbed system that is the same as that assumed in the simulations.

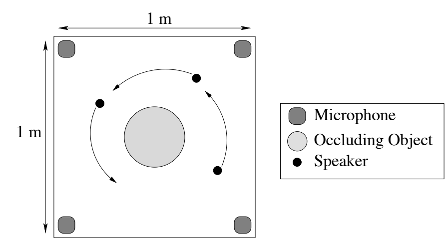

Figure 10: The simulated Whisper system.

Each simulation was run 61 times with the speakers placed randomly around the pole, at an equal distance from the pole, and each rotating around the pole at the same speed. As mentioned above, as the distance between a speaker and microphone changes, so does the amount of computation necessary to correctly track the speaker. This distance is (obviously) impacted by a speaker's movement, but is also lengthened by an occlusion.

In our simulations, we made several simplifying assumptions. First, all objects are moving in only two dimensions. Second, there is no ambient noise in the room. Third, no speaker can interfere with any other speaker. Fourth, all objects move at a constant rate. Fifth, for each speaker/microphone pair, there is only one task. Sixth, the weight of each task changes only once for every 5 cm of distance between its associated speaker and microphone. Finally, all speakers and microphones are omnidirectional. Because there are three speakers in this simulation, there is not sufficient capacity on the assumed system to statically allocate each task the capacity it needs to perform all calculations in the worst case. Even with theses assumptions, frequent share adaptations are required, since the share required by each task changes with the distance between its associated speaker/microphone pair. (In the absence of these assumptions, we expect PD$^2$-LJ to be completely inadequate, since required adaptations would be even more pronounced and frequent than those occurring here.)

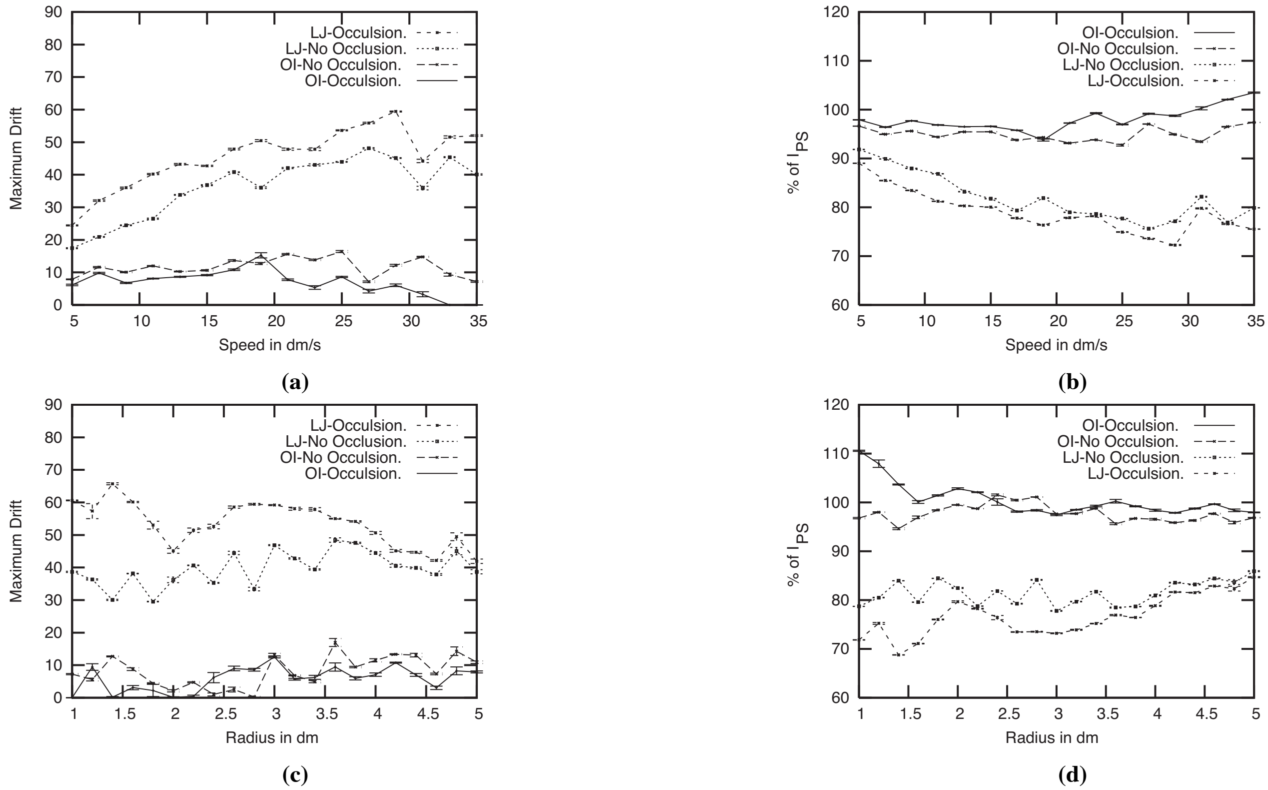


Figure 11: Selection of experiments. For clarity, the legend in each inset orders the curves in the (top-to-bottom) order they appear in that graph. **(a)** Maximum drift as a function of object speed. **(b)** Percent of ideal allocation (as defined by $\mathcal{I}_{\rm PS}$) as a function of object speed. **(c)** Maximum drift as a function of radius of rotation. **(d)** Percent of ideal allocation as a function of radius of rotation.

We conducted experiments in which we varied the distance of each object from the center of the room from 10 cm to 50 cm, the speed of each object from 0.1 m/s to 3.5 m/s (such speeds typify human motion), and the presence of an occluding object (the pole). However, due to page limitations, the graphs below present only a representative sampling of the data we collected. All simulations are run for 1,000 time steps (10 s assuming a 1 ms quantum). While the ultimate metric for determining the efficacy of a tracking system would be user perception, this metric is not currently available for reasons discussed earlier. Thus, we compared PD$^2$-OI and PD$^2$-LJ by measuring the deviance of each from $\mathcal{I}_{\rm PS}$. This metric should provide us with a reasonable impression of how well these systems will fare when Whisper is fully re-implemented. We implemented and timed both reweighting algorithms considered in our simulations on an actual testbed that is the same as that assumed in our simulations, and found that all per-slot scheduling decisions could be made in approximately 5 $\mu$s for all task systems in our experiments. We considered this value to be negligible in comparison to a 1-ms quantum and thus did not consider scheduling overheads in our simulations. In each graph presented below, 98% confidence intervals are given.

In the first two graphs, in Fig. 11(a) and (b), the distance from the center of the room to each speaker is 25 cm, and the speed at which the speakers move varies from 0.5 m/s to 3.5 m/s. Inset (a) depicts the maximal drift of any task in the system at time 1,000 as a function of the speed of the speakers. Inset (b) gives the per-task average total amount of computation completed by time 1,000, as a percentage of the task's allocations in $\mathcal{I}_{\rm PS}$, as a function of the speed of the speakers. Notice that PD$^2$-LJ's performance decreases (*i.e.*, maximal drift increases and the percentage of the $\mathcal{I}_{\rm PS}$ allocation decreases) with an increase in speed. $\mathsf{PD}^2$-$\mathsf{OI}$'s performance, on the other hand, improves with speed. The most probable explanation for this is that not all weight changes incur the same amount of drift. In particular, ideal-changeable tasks (*i.e.*, tasks that are reweighted after being scheduled) incur little drift under $\mathsf{PD}^2$-$\mathsf{OI}$. However, when a ideal-changeable task's weight change is "enacted" by $\mathsf{PD}^2$-$\mathsf{LJ}$, the amount of drift can be substantial. Hence, it is likely that, as the speed of the speakers increases, the number of ideal-changeable tasks also increases. (Note that the system is not fully loaded, so a task can receive more than $100\%$ of its ideal allocation.)

In the second two graphs, in Fig. 11(c) and (d), the speed of the speakers is 2.9 m/s, and the distance from the center of the room to each speaker varies from 10 cm to 50 cm. Inset (c) depicts the maximal drift of any task in the system at time 1,000 as a function of the distance of the speakers. Inset (d) gives the per-task average total amount of computation completed by time 1,000, as a percentage of the task's allocations in $\mathcal{I}_{\mathrm{PS}}$, as a function of the distance of the speakers. One interesting behavior in inset (c) is that the performance of $\mathsf{PD}^2$-$\mathsf{LJ}$, in the presence of occlusions, improves as the distance increases. Such behavior is likely a consequence of the fact that, as the radius of the speakers increases and the speed remains constant, the distance between each speaker and microphone is affected by the occluding object for longer periods of time. Hence, share changes that occur as a speaker becomes (or ceases being) occluded are less frequent, thus improving performance.

Note that the Whisper experiments presented here are fairly generous to $\mathsf{PD}^2$-$\mathsf{LJ}$. While weight changes occur frequently, all weight changes are of one order of magnitude; in fact, most weight changes are fairly incremental. However, even in this scenario, $\mathsf{PD}^2$-$\mathsf{LJ}$ completes at most 85% of the allocations in $\mathcal{I}_{\mathrm{PS}}$, while $\mathsf{PD}^2$-$\mathsf{OI}$ *is always* is within 95% of $\mathcal{I}_{\mathrm{PS}}$.

# 6 Concluding Remarks

We have shown (for the first time) that fine-grained reweighting is possible on fair-scheduled multiprocessor platforms. The experiments reported herein show that our reweighting rules enable greater precision in adapting than $\mathsf{PD}^2$-$\mathsf{LJ}$. However, this added precision comes at the price of higher scheduling costs. $\Omega(max(N, M \log N))$ time is *required* to reweight $N$ tasks simultaneously. In contrast, $\mathsf{PD}^2$-$\mathsf{LJ}$ entails only $O(M \log N)$ time. However, as noted earlier, experiments conducted on our testbed system indicate that scheduling overheads will likely be small in practice under either scheme. Moreover, we have shown in a related paper that this precision-versus-overhead tradeoff can be balanced by using schemes that are hybrids of "pure" $\mathsf{PD}^2$-$\mathsf{OI}$ and $\mathsf{PD}^2$-$\mathsf{LJ}$ [5].

As mentioned earlier in this paper, we have ignored the issue of reweighting heavy tasks. The inclusion of heavy tasks complicates the reweighting rules, since such a task can release a new subtask with a window length of two in (nearly) every time slot. As a result, one "wrong" scheduling decision can force a cascade of "wrong" scheduling decisions. For non-adaptive systems, it is possible to calculate the length of such a cascade, and make scheduling decisions based on that information. However, for adaptive systems, the lengths of these cascades will change with time. Thus, when scheduling adaptive heavy tasks, particular care must be taken to "correct" such "cascades." Because of the complexity involved in constructing and proving reweighting rules for heavy tasks, we refer the reader to the first author's upcoming Ph.D. dissertation, which addresses this issue.

One major drawback to $\mathsf{PD}^2$-$\mathsf{OI}$ scheduling is that it (like all Pfair algorithms) suffers from potentially high migration and preemption costs. These costs can be mitigated by using adaption schemes based upon partitioning [4] and global EDF [7]. However, as noted earlier, under partitioning, fine-grained reweighting is (provably) impossible; and under global EDF, it is possible only if deadline misses are permissible. Because of these various tradeoffs, all three approaches are of value.

As mentioned earlier, while our focus in this paper has been scheduling techniques that *facilitate* fine-grained adaptations, techniques for determining *how* and *when* to adapt are equally important. Such techniques can either be application-specific (*e.g.*, adaptation policies unique to a tracking system like Whisper) or more generic (*e.g.*, feedback-control mechanisms incorporated within scheduling algorithms [8]). Both techniques warrant further study, especially in the domain of multiprocessor platforms.

# References


[1] J. Anderson, A. Block, and A. Srinivasan. Quick-release fair scheduling. In *Proc. of the 24th IEEE Real-time Sys. Symp.*, pp. 130–141, 2003.

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

[3] S. Baruah, J. Gehrke, and C.G. Plaxton. Fast scheduling of periodic tasks on multiple resources. In *Proc. of the 9th Int'l Parallel Processing Symp.*, pp. 280–288, 1995.

[4] A. Block and J. Anderson. Accuracy versus Migration Overhead in Multiprocessor Reweighting Algorithms. In *Proc. of the 12th Int'l Conf. on Parallel and Distributed Sys.*, July 2006, to appear.

[5] A. Block and J. Anderson. Task reweighting on multiprocessors: Efficiency versus accuracy. In *Proc. of the 13th Int'l Workshop on Parallel and Distributed Real-time Sys.*, 2005, on CD ROM.

[6] A. Block, J. Anderson, and G. Bishop. Fine-grained task reweighting on multiprocessors. In *Proc. of the 11th IEEE Int'l Conf. on Embedded and Real-Time Computing Sys. and Apps.*, pages 329–435. IEEE, August 2005.

[7] A. Block, J. Anderson and U. Devi. Task Reweighting under Global Scheduling on Multiprocessors. In *Proc. of the 18th Euromicro Conf. on Real-time Sys.*, July 2006, to appear.

[8] C. Lu, J. Stankovic, G. Tao, and S. Son. Design and evaluation of a feedback control edf scheduling algorithm. In *Proc. of the 20th IEEE Real-time Sys. Symp.*, pp. 44–53, 1999.

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

[10] A. Srinivasan and J. Anderson. Optimal rate-based scheduling on multiprocessors. *Journal of Compupter and Sys. Sciences*, to appear.

[11] A. Srinivasan and J. Anderson. Fair scheduling of dynamic task sys. on multiprocessors. In *Proc. of the 11th Int'l Workshop on Parallel and Distributed Real-time Sys.*, 2003, on CD ROM.

[12] I. Stoica, H. Abdel-Wahab, K. Jeffay, S. Baruah, J. Gehrke, and C.G. Plaxton. A proportional share resource allocation algorithm for real-time, time-shared sys. In *Proc. of the 17th IEEE Real-time Sys. Symp.*, pp. 288–299, 1996.

[13] N. Vallidis. *WHISPER: A Spread Spectrum Approach to Occlusion in Acoustic Tracking*. PhD thesis, University of North Carolina, Chapel Hill, North Carolina, 2002.

# A Appendix

In this section, we prove that PD$^2$-OI correctly schedules any AIS task system that satisfies (W). Note that this proof is only a slight modification of the correctness proof for PD$^2$-LJ originally presented by Srinivasan and Anderson in [11].

## A.1 Preliminaries

Before proving that PD$^2$-OI correctly schedules any AIS task system, we introduce some basic concepts and properties that are useful in the proof. We begin by introducing the "adaptive generalized intra-sporadic" task model. After this, we introduce the notion of a "displacement." Lastly, we introduce some properties and definitions pertaining to PD$^2$-OI.

**The AGIS task model.** To prove the scheduling correctness of PD$^2$-OI in an AIS system, we consider an extension of the AIS task model called the *adaptive generalized intra-sporadic* (*AGIS*) *task model*. The AGIS model generalizes the AIS model by allowing some subtasks to be "absent." An *absent* subtask is never scheduled; however, such subtasks are considered to be part of a given task system, and as such, they have both releases and deadlines. If a subtask is not absent then we say that it is *present*. The function $\mathsf{AB}(T_j)$ denotes whether the subtask $T_j$ is absent or present, where $\mathsf{AB}(T_j) = 0$, if $T_j$ is absent, and $\mathsf{AB}(T_j) = 1$, if $T_j$ is present. For example, in Fig. 12, $V_3$ is absent from the system. Hence, $\mathsf{AB}(V_3) = 0$, and for $j \in \{1, 2, 4, 5\}$, $\mathsf{AB}(V_j) = 1$.

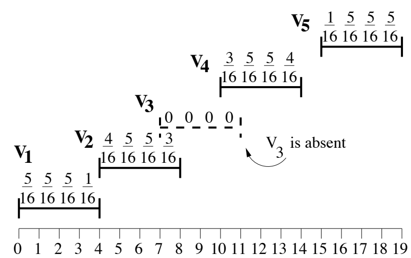


Figure 12: Per-task slot allocations in $\mathcal{I}_{\mathrm{SW}}$ and $\mathcal{I}_{\mathrm{CSW}}$ for a task in an AGIS system for a task $V$ with weight of $5/16$ in which $V_3$ is absent, and with IS seperations between the subtasks $V_1$ and $V_2$ and the subtasks $V_4$ and $V_5$. The dashed window lines indicate that a subtask is absent.

In an AGIS task system, $T_j$ is $T_k$'s *predecessor* (and $T_k$ is $T_j$'s *successor*) iff $T_j$ and $T_k$ are both present and there are no present subtasks that have an index between $j$ and $k$. For example, in Fig. 12, $V_2$ is $V_4$'s predecessor, and $V_4$ is $V_2$'s successor. The per-slot allocations to a subtask $T_j$ in the AGIS variants of the $\mathcal{I}_{\mathrm{SW}}$ and $\mathcal{I}_{\mathrm{CSW}}$ schedules are the same as in the AIS variants, except that if a subtask is absent, then its per-slot allocation is zero in all time slots. For example, in Fig. 12, the per-slot allocations to all subtasks except $V_3$ are the same as in an AIS system, and $V_3$'s per-slot allocation is zero for each time slot. Throughout this appendix, we use $\mathsf{LAG}(\tau, t)$ to denote $\mathsf{LAG}(\mathcal{S}, \mathcal{I}_{\mathrm{CSW}}, \tau, t)$ and $\mathsf{lag}(T, t)$ to denote $\mathsf{lag}(\mathcal{S}, \mathcal{I}_{\mathrm{CSW}}, T, t)$, where $\mathcal{S}$ is the PD$^2$-OI schedule of a task system $\tau$.

Since under the AGIS task model absent tasks never receive any allocations, by Def. 2, such a subtask would never be complete unless it was halted. Therefore, we amend the definition of complete so that an absent subtask $U_j$ is considered to be complete in all schedules as soon as it is released, *i.e.*, $\mathcal{D}(\mathcal{S}, U_j) = \mathcal{D}(\mathcal{I}_{\mathrm{SW}}, U_j) = \mathcal{D}(\mathcal{I}_{\mathrm{CSW}}, U_j) = \mathsf{r}(U_j)$, where $\mathcal{S}$ is the PD$^2$-OI schedule of a task system $\tau$. For example, in Fig. 12, $\mathcal{D}(\mathcal{I}_{\mathrm{SW}}, V_3) = \mathcal{D}(\mathcal{I}_{\mathrm{CSW}}, V_3) = \mathsf{r}(V_3) = 7$. Furthermore, in order to make the definition of $\mathcal{I}_{\mathrm{CSW}}$ consistent, we say that a subtask $U_j$ is complete in the $\mathcal{I}_{\mathrm{CSW}}$ schedule at time $\mathsf{r}(U_j)$ if $U_j$ is a

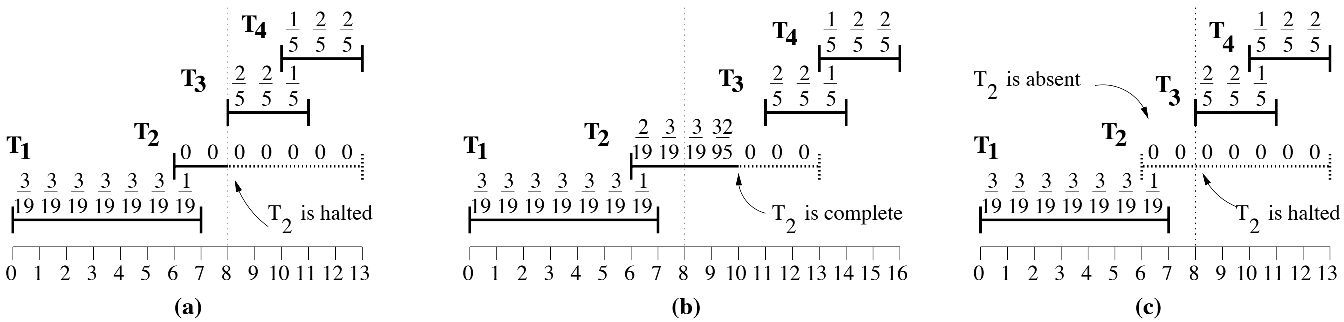


Figure 13: Per-task $\mathcal{I}_{\mathrm{CSW}}$ allocations for a task $T$ in an AGIS system that has an initial weight of $3/19$ and initiates a weight change increase to $2/5$ at time 8. **(a)** $T$ changes its weight via rule O causing $T_2$ to be halted at time 8. **(b)** $T$ changes its weight via rule I causing $T_2$ to be complete at time 10. **(c)** $T_2$ is absent, and so $T$ is omission-changeable at time 8. Thus, causing $T_2$ to be halted at time 8.

halted subtask, *i.e.*, if $\mathsf{H}(U_j) < \infty$, then $\mathcal{D}(\mathcal{I}_{\mathrm{CSW}}, U_j) = \mathsf{r}(U_j)$. For example, in Fig. 13(a), the subtask $T_2$ is halted at time 8, so $\mathcal{D}(\mathcal{I}_{\mathrm{CSW}}, T_2) = \mathsf{r}(T_2) = 6$. In contrast, in Fig. 13(b), the subtask $T_2$ is never halted, and therefore $\mathcal{D}(\mathcal{I}_{\mathrm{CSW}}, T_2) = 10$.

We now address the issue of how rules O and I are applied in an AGIS system. Notice that if a task $U$ initiates a weight change at $t_c$, and the last-released subtask of $U$, $U_j$ is absent, then that subtask has not yet been scheduled. Therefore, if $t_c < \mathsf{d}(U_j)$, then $U$ is omission-changeable at $t_c$. For example, in Fig. 13(c), the subtask $T_2$ is absent, therefore it is not scheduled by time 8, when $T$ initiates a weight change. Thus, $T$ is omission-changeable at time 8. Furthermore, note that an absent subtask can be considered to be "halted," even though it was never eligible to be scheduled. For example, in Fig. 13(c), when $T$ changes its weight via rule O at time 8, $T_2$ is halted, even though it is absent.

**Displacements.** We now introduce the notions of an "instance" of a task system and a task "displacement." An *instance* of a task system is obtained by specifying a unique assignment of release times for each subtask and weight changes for each task. Before continuing, it is worth pointing out that since halted subtasks are never scheduled, and rules O and I behave the same whether the last-released subtask is absent or not, $\mathsf{PD}^2$-$\mathsf{OI}$ produces the same scheduled regardless of whether a halted subtask is absent or present. Thus, we assume that in every task instance presented in this paper, if a subtask is halted, then it is absent. By definition, the *removal* of a subtask (*i.e.*, changing a subtask from present to absent) from one instance of an AGIS task system results in another instance. (Note that only present subtasks can be removed.) Let $X^{(i)}$ denote a subtask of any task in an AGIS task system $\tau$. Let $\mathcal{S}$ denote the $\mathsf{PD}^2$-$\mathsf{OI}$ schedule of $\tau$. Assume that removing $X^{(1)}$ scheduled at slot $t_1$ in $\mathcal{S}$ causes $X^{(2)}$ to shift from slot $t_2$ to $t_1$, where $t_1 \neq t_2$, which in turn may cause other shifts. We call this shift a *displacement* and represent it by the four-tuple $\langle X^{(1)}, t_1, X^{(2)}, t_2 \rangle$. A displacement $\langle X^{(1)}, t_1, X^{(2)}, t_2 \rangle$ is *valid* iff $\mathsf{r}(X^{(2)}) \leq t_1$. Because there can be a cascade of shifts, we may have a *chain* of displacements, as illustrated in Fig. 14. Removing a subtask may also lead to slots in which some processors are idle. In a schedule $\mathcal{S}$, if $k$ processors are idle in slot $t$, then we say that there are $k$ *holes*

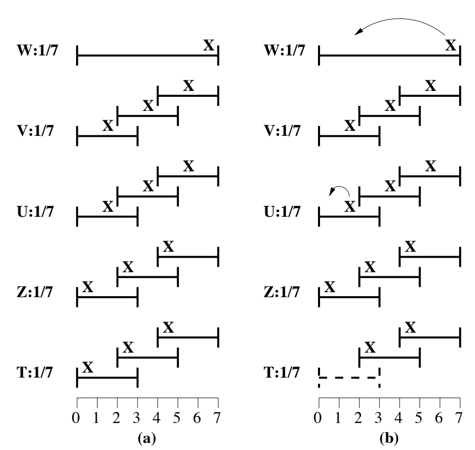


Figure 14: A schedule for four tasks of weight 3/7 and one task of weight 1/7 on two processor. Inset **(b)** illustrates the displacements caused by the removal of subtask $T_1$ from the schedule shown in inset **(a)**.

in $\mathcal{S}$ in slot $t$. Note that holes may exist because of late subtask releases or absent subtasks, even if total utilization is $M$. We now present three lemmas that describe the relationship among subtasks in a chain of a displacement. These three lemmas were originally proven by Srinivasan and Anderson for the "generalized IS task model," *i.e.*, an IS task system, where subtasks can be absent [10]. Since the logic of their proof holds for AGIS task systems, we state these lemmas without proof.

**Lemma 1.** *Let $X^{(1)}$ be a subtask that is removed from $\tau$, where all halted subtasks are absent, and let the resulting chain of displacements in $\mathcal{S}$ be $C = \Delta_1, \Delta_2, ..., \Delta_k$, where $\Delta_i = \langle X^{(i)}, t_i, X^{(i+1)}, t_{i+1}\rangle$. Then $t_{i+1} > t_i$, for all $i \in \{1, ..., k\}$*

**Lemma 2.** *Let $\Delta = \langle X^{(1)}, t_1, X^{(2)}, t_2\rangle$ be a valid displacement in $\mathcal{S}$, in which all halted subtasks are absent. If $t_1 < t_2$ and there is a hole in slot $t_1$ in that schedule, then $X^{(2)}$ is $X^{(1)}$'s successor in $\tau$.*

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

**Reweighting properties.** The following simple property follows directly from the definition of $\mathsf{PD}^2$-$\mathsf{OI}$.

**(RW)** Suppose $T$ initiates a weight change at time $t_c \ge \mathsf{r}(T_1)$ and $T_j$ is the last-released subtask of $T$ at $t_c$. If $\mathsf{r}(T_j) \le t_c < \mathsf{d}(T_j)$, then $T$ is either omission- or ideal-changeable at $t_c$.

In Sec. 4, we presented property (V), which relates the release times, completion times, and deadlines of two subtasks $T_i$ and $T_{i+1}$. In the following proof, it is useful to extend this property to relate the release times, completion times, and deadlines of two subtasks $T_i$ and $T_k$, where $i < k$.

**(GV)** For the subtasks $T_i$ and $T_k$, where $i < k$, if $\mathsf{d}(T_i) - \mathsf{b}(T_i) > \mathsf{r}(T_k)$, then $\mathcal{D}(\mathcal{I}_{\mathrm{CSW}}, T_i) \le \mathsf{r}(T_k)$ and $\mathcal{D}(\mathcal{S}, T_i) \le \mathsf{r}(T_k)$.

The proof of (GV) follows directly from (V). Notice that the above property holds even if $T_i$ or $T_j$ is absent.

We now state two properties about the relationship between the initiation and enactment of reweighting events.

**(X1)** If a task $T$ initiates a weight change at time $t_c$, then it is enacted no later than time $\mathsf{r}(T_k)$, where $T_k$ is the next-released subtask of $T$.

**(X2)** If a weight change is enacted over the range $(\mathsf{r}(T_\ell), \mathsf{d}(T_\ell)]$, then that change must have been initiated over the range $[\mathsf{r}(T_\ell), \mathsf{d}(T_\ell)]$.

Given that property (C) guarantees that no sequence of reweighting events can delay the next weight-change enactment after a weight change has been initiated, and that the rules I and O guarantee a subtask is released within one quantum of a weight change enactment, the subtask $T_k$ (as defined in (X1)) will eventually be released. Thus, from the definitions of rules O and I, (X1) should be fairly intuitive. As for (X2), it follows from the contrapositive of (X1).

We now introduce four properties about the per-slot allocations of a task and the completion time of a subtask in an AGIS system that are useful in the correctness proof.

**AF1:** For all $t \ge 0$, $\mathsf{A}(\mathcal{I}_{\mathrm{CSW}}, T, t) \le \mathsf{swt}(T, t)$.

**AF2:** Let $\tau$ be an AGIS system in which all halted subtasks are absent. For any present subtask $T_i$ of the task $T \in \tau$ and its successor $T_k$, if $\mathsf{b}(T_i) = 1$ and $\mathsf{r}(T_k) \geq \mathcal{D}(\mathcal{I}_{\text{CSW}}, T_i)$, then $\mathsf{A}(\mathcal{I}_{\text{CSW}}, T, \mathcal{D}(\mathcal{I}_{\text{CSW}}, T_i) - 1, \mathcal{D}(\mathcal{I}_{\text{CSW}}, T_i) + 1) \leq \mathsf{swt}(T, \mathcal{D}(\mathcal{I}_{\text{CSW}}, T_i))$.

**AF3:** For any subtask $T_i$, $\mathcal{D}(\mathcal{I}_{\text{CSW}}, T_i) \leq \mathsf{d}(T_i)$.

**AF4:** For any subtask $T_i$ and any time $t$, if $t < \mathsf{r}(T_i) \vee t \geq \mathcal{D}(\mathcal{I}_{\text{CSW}}, T_i)$, then $\mathsf{A}(\mathcal{I}_{\text{CSW}}, T_i, t) = 0$.

Given the examples in Fig. 12 and Fig. 13, both (AF1) and (AF4) should be fairly intuitive. As for (AF2), notice that in Fig. 12, $\mathsf{A}(\mathcal{I}_{\text{CSW}}, V, \mathcal{D}(\mathcal{I}_{\text{CSW}}, V_1) - 1, \mathcal{D}(\mathcal{I}_{\text{CSW}}, V_1) + 1) = 1/16 + 4/16 \leq \mathsf{swt}(V, 4) = 5/16$ and $\mathsf{A}(\mathcal{I}_{\text{CSW}}, V, \mathcal{D}(\mathcal{I}_{\text{CSW}}, V_4) - 1, \mathcal{D}(\mathcal{I}_{\text{CSW}}, V_4) + 1) = 4/16 \leq \mathsf{swt}(V, 14) = 5/16$. Also notice that in Fig. 13(b), which depicts a task $T$ that changes its weight from $3/19$ to $2/5$ via rule I at time 8, $\mathsf{A}(\mathcal{I}_{\text{CSW}}, T, \mathcal{D}(\mathcal{I}_{\text{CSW}}, T_2) - 1, \mathcal{D}(\mathcal{I}_{\text{CSW}}, T_2) + 1) = 32/95 + 0 \leq \mathsf{swt}(T, 10) = 2/5$. Also note that in Fig. 13(c), which depicts a task $T$ that changes its weight from $3/19$ to $2/5$ via rule O at time 8, $\mathsf{A}(\mathcal{I}_{\text{CSW}}, T, \mathcal{D}(\mathcal{I}_{\text{CSW}}, T_1) - 1, \mathcal{D}(\mathcal{I}_{\text{CSW}}, T_1) + 1) = 1/19 + 0 \leq \mathsf{swt}(T, 7) = 3/19$. Finally, notice that (AF2) does not apply to the system depicted in Fig. 13(a), since $T_2$ is halted, but it is not absent. As for (AF3), recall that in the absence of reweighting events, $\mathsf{d}(T_i) = \mathcal{D}(\mathcal{I}_{\text{CSW}}, T_i)$. To increase the completion time of $T_i$ (and hence $\mathcal{D}(\mathcal{I}_{\text{CSW}}, T_i)$) in $\mathcal{I}_{\text{CSW}}$, $T$ would have to enact a weight change in the range $(\mathsf{r}(T_i), \mathsf{d}(T_i))$ that decreases the weight of $T$, without halting $T_i$. However, by (X2), a change enacted in the range $(\mathsf{r}(T_i), \mathsf{d}(T_i))$ must have been initiated in the range $[\mathsf{r}(T_i), \mathsf{d}(T_i))$. Thus, by (RW) when such a change is initiated, $T$ is either omission- or ideal-changeable. Since only rule I can change the weight of a task without halting the last-released subtask, when such a change is initiated, $T$ must be ideal-changeable. However, by rule I no weight decrease that is initiated in the range $[\mathsf{r}(T_i), \mathsf{d}(T_i))$ can be enacted before $\mathcal{D}(\mathcal{I}_{\text{CSW}}, T_i)$. Thus, the completion time for a subtask is upper-bounded by the deadline of the subtask.

As a consequence of (AF1) and property (W), $\mathsf{LAG}$ can only increase over a time slot if there is a hole in that slot. Hence, the lemma below follows.

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

## A.2 Correctness Proof

Having defined the AGIS task model, displacements, and some basic properties, we can now prove Thm. 2. Suppose that Thm. 2 does not hold. Then, there exist a time $t_d$ and a task system $\tau$ as given in Defs. 3 and 4 below.

**Definition 3** **($t_d$).** $t_d$ is the earliest time at which any AGIS task system instance misses a deadline under $\mathsf{PD}^2$-$\mathsf{OI}$.

**Definition 4** **($\tau$ and $\mathcal{S}$).** $\tau$ is an instance of an AGIS task system with the following properties.

**(T1)** $\tau$ misses a deadline under $\mathsf{PD}^2$-$\mathsf{OI}$ at $t_d$.

**(T2)** No task system satisfying (T1) has fewer present subtasks in $[0, t_d)$ than $\tau$.

In the remainder of this proof, we let $\mathcal{S}$ denote $\mathsf{PD}^2$-$\mathsf{OI}$ schedule of $\tau$.

By (T1), (T2), and Def. 3, exactly one subtask in $\tau$ misses its deadline at $t_d$: if several subtasks miss their deadlines, all but one can be removed and the remaining subtask will still miss its deadline, contradicting (T2). We now prove several properties about $\mathcal{S}$.

**Lemma 5.** *The following properties hold for $\tau$ and $\mathcal{S}$, where $T_i$ is any subtask in $\mathcal{S}$.*

**(a)** *For any present subtask $T_i$, $\mathsf{d}(T_i) \leq t_d$.*

**(b)** *There are no holes in slot $t_d - 1$.*

**(c)** $\mathsf{LAG}(\tau, t_d) = 1$.

**(d)** $\mathsf{LAG}(\tau, t_d - 1) \geq 1$.

**(e)** *For any present subtask $T_i$, $\mathsf{H}(T_i) = \infty$.*

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

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

**Proof of (c).** By (1), we have

$$\mathsf{LAG}(\tau, t_d) = \mathsf{A}(\mathcal{I}_{\text{CSW}}, \tau, 0, t_d) - \mathsf{A}(\mathcal{S}, \tau, 0, t_d).$$

In the above equation, the term $\mathsf{A}(\mathcal{I}_{\text{CSW}}, \tau, 0, t_d)$ equals the total number of present subtasks in $\tau$. The second term corresponds to the number of subtasks scheduled by $\mathsf{PD}^2$-$\mathsf{OI}$ in $[0, t_d)$. Since exactly one subtask misses its deadline, the difference between these two terms is $1$, *i.e.*, $\mathsf{LAG}(\tau, t_d) = 1$. □

**Proof of (d).** By (b), there are no holes in slot $t_d - 1$. Hence, by Lemma 4, $\mathsf{LAG}(\tau, t_d - 1) \geq \mathsf{LAG}(\tau, t_d)$. Therefore by (c), $\mathsf{LAG}(\tau, t_d - 1) \geq 1$. □

**Proof of (e).** If $T_i$ is a halted subtask, then it is never scheduled. Hence, it can be removed and a deadline will still be missed at $t_d$. Contradicting (T2). □

**Definition 4 ($A_t$, $B_t$, and $I_t$).** Before continuing, we define the sets $A_t$, $B_t$, and $I_t$, which are all defined with respect to schedule $\mathcal{S}$ and some time $t$. $A_t$ denotes the set of tasks that have a subtask scheduled at $t$. $B_t$ denotes the set of tasks that are not scheduled at $t$, and receive some allocation in $\mathcal{I}_{\text{CSW}}$ at slot $t$, *i.e.*, $\mathsf{A}(\mathcal{I}_{\text{CSW}}, T, t) > 0$ for $T \in B_t$. $I_t$ denotes the set of all tasks that are in the system at $t$ but are not in $A_t$ or $B_t$. These three sets are illustrated in Fig. 15.

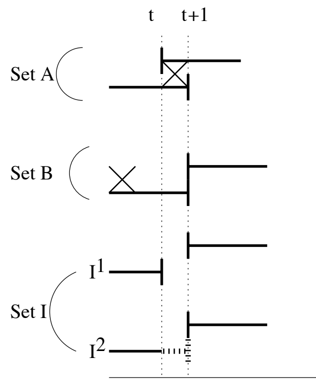


Figure 15: The task sets $A$, $B$, and $I$. There are two types of tasks in set $I$: tasks that have a deadline at or before $t$ (denoted $I^1$) and tasks that have a deadline after $t$ but complete in $\mathcal{I}_{\mathrm{CSW}}$ at or before $t$ (denoted $I^2$).

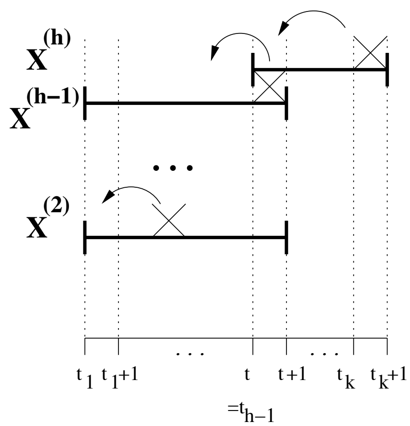


Figure 16: An illustration of the chain of displacements that occurs by removing $X^{(1)} = U_j$ in Lemma 6. $X^{(h)}$ must be the successor of $X^{(h-1)}$ because there is a hole in slot $t$.

**Lemma 6.** *Let $t < t_d - 1$ be a time at which there is a hole in $\mathcal{S}$. Let $U$ be any task in $B_t$ or $A_t$. Let $U_j$ be the subtask with the largest index such that $\mathsf{r}(U_j) \le t < \mathsf{d}(U_j)$ and $U_j$ is scheduled at or before $t$. Then $\mathsf{d}(U_j) = t + 1 \wedge \mathsf{b}(U_j) = 1$.*

*Proof.* Let $t$ and $U$ be as defined in the statement of the lemma. If $U \in A_t$, then since $t < t_d - 1$ and $U_j$ is scheduled at or before $t$, $\mathsf{d}(U_j) \ge t + 1$. If $U \in B_t$, then since $\mathsf{A}(\mathcal{I}_{\mathrm{CSW}}, U, t) > 0$, by the definition of $\mathcal{D}$ and AF3, $\mathsf{d}(U_j) \ge t + 1$. Suppose that the following holds to derive a contradiction.

$$\mathsf{d}(U_j) > t + 1 \text{ or } \mathsf{d}(U_j) = t + 1 \wedge \mathsf{b}(U_j) = 0 \tag{6}$$

We now show that $U_j$ can be removed and a deadline will still be missed at $t_d$, contradicting (T2). (Before we continue, notice that since $U_j$ is scheduled it is present, and as a result $U_j$ can be removed.) Let the chain of displacements caused by removing $U_j$ be $\Delta_1, \Delta_2, ..., \Delta_k$, where $\Delta_i = \langle X^{(i)}, t_i, X^{(i+1)}, t_{i+1} \rangle$, $X^{(1)} = U_j$. By Lemma 1, $t_{i+1} > t_i$, for $1 \le i \le k$. The chain of displacements under consideration is illustrated in Fig. 16.

Note that at slot $t_i$, the priority of $X^{(i)}$ is at least that of $X^{(i+1)}$, because $X^{(i)}$ was chosen over $X^{(i+1)}$ in $\mathcal{S}$. Thus, because $X^{(1)} = U_j$, by (6), for each subtask $1 \le i \le k + 1$, either $\mathsf{d}(X^{(i)}) > t + 1$ or $\mathsf{d}(X^{(i)}) = t + 1 \wedge \mathsf{b}(X^{(i)}) = 0$. We now show that the displacements do not extend beyond slot $t$. Assume to the contrary that $t_{k+1} > t$. Consider $h \in \{2, ..., k + 1\}$ such that $t_h > t$ and $t_{h-1} \leq t$. Such an $h$ exists because $t_1 \leq t < t_{k+1}$. Because there is a hole in slot $t$ and $t_{h-1} \leq t < t_h$, by Lemma 3, $t_{h-1} = t$ and $X^{(h)}$ is $X^{(h-1)}$'s successor. Since a subtask cannot be scheduled before it is released, $\mathsf{r}(X^{(h)}) < t+1$. Since $h-1 \leq k$, either $\mathsf{d}(X^{(h-1)}) > t+1$ or $\mathsf{d}(X^{(h-1)}) = t+1 \wedge \mathsf{b}(X^{(h-1)}) = 0$ holds. Therefore, since $\mathsf{r}(X^{(h)}) < t+1$, $\mathsf{d}(X^{(h-1)}) - \mathsf{b}(X^{(h-1)}) > \mathsf{r}(X^{(h)})$ holds. Thus, by (GV), $X^{(h-1)}$ is scheduled before $\mathsf{r}(X^{(h)}) \leq t$ in $\mathcal{S}$, contradicting our assumption that $X^{(h-1)}$ is scheduled in slot $t$.

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

Since, by part (d) of Lemma 5, $\mathsf{LAG}(\tau, t_d - 1) \geq 1$ and, by definition of $\mathsf{LAG}$, $\mathsf{LAG}(\tau, 0) = 0$, there exists a time $t_{\mathcal{H}}$ such that

$$0 \leq t_{\mathcal{H}} < t_d - 1 \wedge \mathsf{LAG}(\tau, t_{\mathcal{H}}) < 1 \wedge \mathsf{LAG}(\tau, t_{\mathcal{H}} + 1) \geq 1. \tag{7}$$

Without loss of generality, let $t_{\mathcal{H}}$, be the latest such time, *i.e.*, for all $u$ such that $t_{\mathcal{H}} < u \leq t_d - 1$, $\mathsf{LAG}(\tau, u) \geq 1$. We now show that such a $t_{\mathcal{H}}$ cannot exist, thus contradicting our starting assumption that $t_d$ and $\tau$ exist. For brevity, we use $A$ to denote $A_{t_{\mathcal{H}}}$, $B$ to denote $B_{t_{\mathcal{H}}}$, and $I$ to denote $I_{t_{\mathcal{H}}}$.

**Lemma 7.** *$B$ is non-empty*

*Proof.* Let the number of holes is slot $t_{\mathcal{H}}$ be $h$. Then, $\mathsf{A}(\mathcal{S}, \tau, t_{\mathcal{H}}) = M - h$. By (13), $\mathsf{LAG}(\tau, t_{\mathcal{H}} + 1) = \mathsf{LAG}(\tau, t_{\mathcal{H}}) + \mathsf{A}(\mathcal{I}_{\mathrm{CSW}}, \tau, t_{\mathcal{H}}) - \mathsf{A}(\mathcal{S}, \tau, t_{\mathcal{H}})$. Thus, because $\mathsf{LAG}(\tau, t_{\mathcal{H}} + 1) > \mathsf{LAG}(\tau, t_{\mathcal{H}})$, we have $\mathsf{A}(\mathcal{I}_{\mathrm{CSW}}, \tau, t_{\mathcal{H}}) > M - h$. Since, for every $V \notin A \cup B$, $\mathsf{A}(\mathcal{I}_{\mathrm{CSW}}, V, t_{\mathcal{H}}) = 0$, it follows that $\mathsf{A}(\mathcal{I}_{\mathrm{CSW}}, A \cup B, t_{\mathcal{H}}) > M - h$. Therefore, by (AF1), $\sum_{T \in A} (\mathsf{swt}(T, t_{\mathcal{H}})) + \sum_{T \in B} (\mathsf{A}(\mathcal{I}_{\mathrm{CSW}}, T, t_{\mathcal{H}})) > M - h$. Because the number of tasks scheduled in slot $t_{\mathcal{H}}$ is $M - h$, $|A| = M - h$. Because $\mathsf{swt}(T, t) \leq 1$, for any task $T$ at any time $t$, $\sum_{T \in A} \mathsf{swt}(T, t_{\mathcal{H}}) \leq M - h$. Thus, $\sum_{T \in B} \mathsf{A}(\mathcal{I}_{\mathrm{CSW}}, T, t_{\mathcal{H}}) > 0$. Hence, $B$ is not empty. □

**(TK)** Let $U$ be any task in $B$ and let $U_j$ be the subtask of $U$ with the largest index such that $\mathsf{r}(U_j) \leq t_{\mathcal{H}} < \mathsf{d}(U_j)$ and $U_j$ is scheduled before $t_{\mathcal{H}}$. Then, no present subtask with an index greater than $j$ (including $U_j$'s successor), is released before $\mathsf{d}(U_j)$.

Property (TK) easily follows from the fact that there is a hole in slot $t_{\mathcal{H}}$ and no subtask of $B$ is scheduled in slot $t_{\mathcal{H}}$.

**Lemma 8.** *Let $U$ be any task in $B$. Let $U_j$ be the subtask of $U$ with the largest index such that $\mathsf{r}(U_j) \leq t_{\mathcal{H}} < \mathsf{d}(U_j)$ and $U_j$ is scheduled before $t_{\mathcal{H}}$. Then, $\mathcal{D}(\mathcal{I}_{\mathrm{CSW}}, U_j) = t_{\mathcal{H}} + 1$.*

*Proof.* Let $U$ and $U_j$ be defined as in the statement of the lemma. Since $U_j$ is the subtask of $U$ with the largest index such that $\mathsf{r}(U_j) \leq t_{\mathcal{H}} < \mathsf{d}(U_j)$ and $U_j$ is scheduled before $t_{\mathcal{H}}$ (by the definition of $B$, no subtask of $U \in B$ is scheduled in slot $t_{\mathcal{H}}$), by Lemma 6, $\mathsf{d}(U_j) = t_{\mathcal{H}} + 1$. Thus, by (AF3), $\mathcal{D}(\mathcal{I}_{\mathrm{CSW}}, U_j) \leq t_{\mathcal{H}} + 1$.

We now show that for $\ell \neq j$, $\mathsf{A}(\mathcal{I}_{\mathrm{CSW}}, U_\ell, t_{\mathcal{H}}) = 0$. First, we consider $\ell > j$. From the definition of $U_j$, at least one of the following three conditions must hold: **(i)** $\mathsf{r}(U_\ell) > t_{\mathcal{H}}$; **(ii)** $\mathsf{d}(U_\ell) \leq t_{\mathcal{H}}$; or **(iii)** $U_\ell$ is not scheduled by $t_{\mathcal{H}}$. If conditions (i) or (ii) hold, then by (AF3) and (AF4), $\mathsf{A}(\mathcal{I}_{\mathrm{CSW}}, U_\ell, t_{\mathcal{H}}) = 0$. If condition (iii) holds, then by (TK) either $\mathsf{r}(U_\ell) > \mathsf{d}(U_j)$ (in which case condition (i) holds) or $U_\ell$ is not present (in which case $\mathsf{A}(\mathcal{I}_{\text{CSW}}, U_\ell, t_\mathcal{H}) = 0$). Thus, for $\ell > j$, $\mathsf{A}(\mathcal{I}_{\text{CSW}}, U_\ell, t_\mathcal{H}) = 0$. Now, consider $\ell < j$. By Lem. 6, $\mathsf{d}(U_j) = t_\mathcal{H} + 1 \wedge \mathsf{b}(U_j) = 1$. Since every task has a weight of at most $1/2$, every subtask with a b-bit of one has a window length of at least three. Hence, $\mathsf{r}(U_j) \leq t_\mathcal{H} - 2$. By (GV), $\mathsf{d}(U_\ell) - \mathsf{b}(U_\ell) > \mathsf{r}(U_j)$ holds only if $\mathcal{D}(\mathcal{I}_{\text{CSW}}, U_\ell) \leq \mathsf{r}(U_j)$. Thus, either $\mathcal{D}(\mathcal{I}_{\text{CSW}}, U_\ell) \leq \mathsf{r}(U_j)$ or $\mathsf{d}(U_\ell) - \mathsf{b}(U_\ell) \leq \mathsf{r}(U_j)$ holds. Since, $\mathsf{r}(U_j) \leq t_\mathcal{H} - 2$, either $\mathcal{D}(\mathcal{I}_{\text{CSW}}, U_\ell) \leq \mathsf{r}(U_j) \leq t_\mathcal{H} - 2$ or $\mathsf{d}(U_\ell) \leq \mathsf{r}(U_j) + \mathsf{b}(U_\ell) \leq t_\mathcal{H} - 1$ holds. Since, by (AF3), $\mathcal{D}(\mathcal{I}_{\text{CSW}}, U_\ell) \leq \mathsf{d}(U_\ell)$, $\mathcal{D}(\mathcal{I}_{\text{CSW}}, U_\ell) \leq t_\mathcal{H} - 1$ holds in either case. Thus, by (AF4), $\mathsf{A}(\mathcal{I}_{\text{CSW}}, U_\ell, t_\mathcal{H}) = 0$.

Thus, the allocation to each subtask of $U$, except $U_j$, in $\mathcal{I}_{\text{CSW}}$ in slot $t_\mathcal{H}$ is zero. By the definition of $B$, $\mathsf{A}(\mathcal{I}_{\text{CSW}}, U, t_\mathcal{H}) > 0$. Thus, since $\mathsf{A}(\mathcal{I}_{\text{CSW}}, U, t_\mathcal{H}) = \sum_{U_i \in U} \mathsf{A}(\mathcal{I}_{\text{CSW}}, U_i, t_\mathcal{H})$, there must exist at least one present subtask with a positive allocation in $\mathcal{I}_{\text{CSW}}$ in the slot $t_\mathcal{H}$. Thus, $\mathsf{A}(\mathcal{I}_{\text{CSW}}, U_j, t_\mathcal{H}) > 0$. By (AF4), this implies that $\mathcal{D}(\mathcal{I}_{\text{CSW}}, U_j) \geq t_\mathcal{H} + 1$. Since we have already established that $\mathcal{D}(\mathcal{I}_{\text{CSW}}, U_j) \leq t_\mathcal{H} + 1$, we have $\mathcal{D}(\mathcal{I}_{\text{CSW}}, U_j) = t_\mathcal{H} + 1$. □

**Lemma 9.** *There is no hole in slot* $t_\mathcal{H} + 1$.

*Proof.* First, we show that at least one subtask is scheduled at $t_\mathcal{H} + 1$, after which we show that there are no holes in slot $t_\mathcal{H} + 1$. Notice that if no subtask were scheduled in slot $t_\mathcal{H} + 1$, then every subtask that was released at or before $t_\mathcal{H} + 1$ would have been scheduled before $t_\mathcal{H} + 1$. Thus, if no subtask were scheduled in slot $t_\mathcal{H} + 1$, then the $\mathsf{lag}$ of every task would be would non-positive at $t_\mathcal{H} + 1$, which implies that $\mathsf{LAG}(\tau, t_\mathcal{H} + 1) \leq 0$. However, by (7), $\mathsf{LAG}(\tau, t_\mathcal{H} + 1) \geq 1$. Thus, there must be at least one subtask scheduled in slot $t_\mathcal{H} + 1$.

Now we show that there cannot be any holes in slot $t_\mathcal{H} + 1$. Suppose, to derive a contradiction, that there is a hole in slot $t_\mathcal{H} + 1$. Let $U_j$ be a subtask scheduled in slot $t_\mathcal{H} + 1$. ($U_j$ must exist, as shown above.) Since $U_j$ is scheduled at $t_\mathcal{H} + 1 < t_d$, $U_j$ does not miss its deadline. Hence, $\mathsf{d}(U_j) \geq t_\mathcal{H} + 2$. Since a subtask is not scheduled before it is released and $U_j$ is scheduled in slot $t_\mathcal{H} + 1$, $\mathsf{r}(U_j) \leq t_\mathcal{H} + 1$. Since subtasks are scheduled in order of their index, $U_j$ has the largest index of any subtask of $U$ scheduled at or before $t_\mathcal{H} + 1$. Thus, since $\mathsf{r}(U_j) \leq t_\mathcal{H} + 1 < \mathsf{d}(U_j)$, by Lemma 6, $\mathsf{d}(U_j) = t_\mathcal{H} + 2 \wedge \mathsf{b}(U_j) = 1$. Since the maximum weight of a task is $1/2$, every subtask that has a b-bit of one has a window length of at least three quanta. Thus,

$$\mathsf{r}(U_j) \leq \mathsf{d}(U_j) - 3 = t_\mathcal{H} - 1. \tag{8}$$

Since there is a hole in slot $t_\mathcal{H}$, $U_j$ is not scheduled in slot $t_\mathcal{H}$ and $\mathsf{r}(U_j) < t_\mathcal{H}$, a predecessor to $U_j$, $U_k$, must have been scheduled in slot $t_\mathcal{H}$. As with $U_j$, since $U_k$ is scheduled in slot $t_\mathcal{H} < t_d - 1$, $U_k$ is the subtask of $U$ with the largest index such that $\mathsf{r}(U_k) \leq t_\mathcal{H} < \mathsf{d}(U_k)$. Thus, by Lemma 6, $\mathsf{d}(U_k) = t_\mathcal{H} + 1 \wedge \mathsf{b}(U_k)$. $U_j$ and $U_k$ are illustrated in Fig. 17. Since $\mathsf{r}(U_j) \leq t_\mathcal{H} - 1 < \mathsf{d}(U_k) - \mathsf{b}(U_k)$, by (GV), $U_k$ is complete in $\mathcal{S}$ by $t_\mathcal{H}$ (*i.e.*, scheduled before $t_\mathcal{H}$), which contradicts the fact that $U_k$ is scheduled at $t_\mathcal{H}$. □

The following lemma contradicts our choice of $t_\mathcal{H}$ as the last slot such that $\mathsf{LAG}(\tau, t_\mathcal{H}) < 1$.

**Lemma 10.** $\mathsf{LAG}(\tau, t_\mathcal{H} + 2) < 1$.

*Proof.* Let the number of holes in slot $t_\mathcal{H}$ be $h$. We now derive some properties about the per-slot allocations to tasks in the $\mathcal{I}_{\text{CSW}}$ schedule in slots $t_\mathcal{H}$ and $t_\mathcal{H} + 1$.

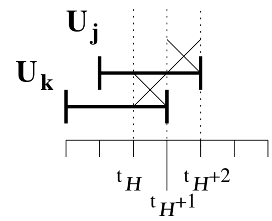


Figure 17: An illustration of the proof of Lemma 9, where there is a hole in time slots $t_{\mathcal{H}}$ and $t_{\mathcal{H}}+1$.

By the definition of $I$, if task $T$ is in $I$, then $\mathsf{A}(\mathcal{I}_{\mathrm{CSW}}, T, t_{\mathcal{H}}) = 0$. Since $\tau = A \cup B \cup I$, $\sum_{T \in \tau} \mathsf{A}(\mathcal{I}_{\mathrm{CSW}}, T, t_{\mathcal{H}}) = \sum_{T \in A \cup B} \mathsf{A}(\mathcal{I}_{\mathrm{CSW}}, T, t_{\mathcal{H}})$. Since $\mathsf{swt}(T, t) \leq 1$, for any task $T$ and any time $t$, we have $\sum_{T \in A} \mathsf{swt}(T, t_{\mathcal{H}}) \leq |A|$. Thus, by (AF1), $\sum_{T \in A} \mathsf{A}(\mathcal{I}_{\mathrm{CSW}}, T, t_{\mathcal{H}}) \leq |A|$. Because there are $h$ holes in slot $t_{\mathcal{H}}$, $M-h$ tasks are scheduled at $t_{\mathcal{H}}$, *i.e.*, $|A| = M-h$. Thus, $\sum_{T \in A} \mathsf{A}(\mathcal{I}_{\mathrm{CSW}}, T, t_{\mathcal{H}}) \leq M-h$, and hence

$$\sum_{T \in \tau} \mathsf{A}(\mathcal{I}_{\mathrm{CSW}}, T, t_{\mathcal{H}}) \leq M - h + \sum_{T \in B} \mathsf{A}(\mathcal{I}_{\mathrm{CSW}}, T, t_{\mathcal{H}}). \tag{9}$$

Let $C$ denote the set of tasks that receive a positive allocation in $\mathcal{I}_{\mathrm{CSW}}$ in slot $t_{\mathcal{H}}+1$ *and* are not in $B$. Then, the set of tasks that receive a positive allocation in $\mathcal{I}_{\mathrm{CSW}}$ is a subset of $C \cup B$. Thus, by property (W) in Sec. 4,

$$\sum_{T \in C \cup B} \mathsf{swt}(T, t_{\mathcal{H}}) \leq M. \tag{10}$$

Also, $\sum_{T \in \tau} \mathsf{A}(\mathcal{I}_{\mathrm{CSW}}, T, t_{\mathcal{H}}+1) = \sum_{T \in C \cup B} \mathsf{A}(\mathcal{I}_{\mathrm{CSW}}, T, t_{\mathcal{H}}+1)$. By (AF1), this implies that $\sum_{T \in \tau} \mathsf{A}(\mathcal{I}_{\mathrm{CSW}}, T, t_{\mathcal{H}}+1) \leq \sum_{T \in C} \mathsf{swt}(T, t_{\mathcal{H}}+1) + \sum_{T \in B} \mathsf{A}(\mathcal{I}_{\mathrm{CSW}}, T, t_{\mathcal{H}}+1)$. Thus, by (9),

$$\sum_{T \in \tau} \mathsf{A}(\mathcal{I}_{\mathrm{CSW}}, T, t_{\mathcal{H}}, t_{\mathcal{H}}+2) \leq M - h + \sum_{T \in C} \mathsf{swt}(T, t_{\mathcal{H}}+1) + \sum_{T \in B} \mathsf{A}(\mathcal{I}_{\mathrm{CSW}}, T, t_{\mathcal{H}}, t_{\mathcal{H}}+2) \tag{11}$$

Consider $U \in B$. Let $U_j$ be the subtask of $U$ with the largest index such that $\mathsf{r}(U_j) \leq t_{\mathcal{H}} < \mathsf{d}(U_j)$ that is scheduled before $t_{\mathcal{H}}$. Let $D$ denote the set of such subtasks for all tasks in $B$. Then, by Lemmas 6 and 8,

$$\text{for all } U_j \in D, \mathcal{D}(\mathcal{I}_{\mathrm{CSW}}, U_j) = \mathsf{d}(U_j) = t_{\mathcal{H}}+1 \wedge \mathsf{b}(U_j) = 1. \tag{12}$$

By (TK), $U_j$'s successor $U_k$ is not released until $t_{\mathcal{H}}+1 \geq \mathcal{D}(\mathcal{I}_{\mathrm{CSW}}, U_j)$. Since $\mathsf{r}(U_k) \geq \mathcal{D}(\mathcal{I}_{\mathrm{CSW}}, U_j)$ and $\mathsf{b}(U_j) = 1$, by (AF2), $\mathsf{A}(\mathcal{I}_{\mathrm{CSW}}, U, t_{\mathcal{H}}, t_{\mathcal{H}}+2) \leq \mathsf{swt}(U, t_{\mathcal{H}}+1)$. Thus, $\sum_{T \in B} \mathsf{A}(\mathcal{I}_{\mathrm{CSW}}, T, t_{\mathcal{H}}, t_{\mathcal{H}}+2) \leq \sum_{T \in B} \mathsf{swt}(T, t_{\mathcal{H}}+1)$.

By (11), this implies that $\sum_{T \in \tau} \mathsf{A}(\mathcal{I}_{\mathrm{CSW}}, T, t_{\mathcal{H}}, t_{\mathcal{H}}+2) \leq M - h + \sum_{T \in C \cup B} \mathsf{swt}(T, t_{\mathcal{H}}+1)$. Thus, from (10) it follows that

$$\sum_{T \in \tau} \mathsf{A}(\mathcal{I}_{\mathrm{CSW}}, T, t_{\mathcal{H}}, t_{\mathcal{H}}+2) \leq M - h + M. \tag{13}$$

By Lemma 9, there is no hole in slot $t_{\mathcal{H}}+1$. Since there are $h$ holes in slot $t_{\mathcal{H}}$, we have $\mathsf{A}(\mathcal{S}, \tau, t_{\mathcal{H}}, t_{\mathcal{H}}+2) = M-h+M$. Hence, by (13), $\sum_{T \in \tau} \mathsf{A}(\mathcal{I}_{\mathrm{CSW}}, T, t_{\mathcal{H}}, t_{\mathcal{H}}+2) \leq \sum_{T \in \tau} \mathsf{A}(\mathcal{S}, \tau, t_{\mathcal{H}}, t_{\mathcal{H}}+2)$. Using this relation in (1) we obtain, $\mathsf{LAG}(\tau, t_{\mathcal{H}}+2) =$

$\mathsf{LAG}(\tau, t_{\mathcal{H}}) + \mathsf{A}(\mathcal{I}_{\mathrm{CSW}}, T, t_{\mathcal{H}}, t_{\mathcal{H}} + 2) - \mathsf{A}(\mathcal{S}, \tau, t_{\mathcal{H}}, t_{\mathcal{H}} + 2)$. Since, $\mathsf{LAG}(\tau, t_{\mathcal{H}}) < 1$, we obtain $\mathsf{LAG}(\tau, t_{\mathcal{H}} + 2) < 1$. $\square$

It follows, by Lemma 10, that $t_d$ does not exist. Thus, Thm. 2 holds.