Task Reweighting under Global Scheduling on Multiprocessors

Aaron Block, James H. Anderson, and UmaMaheswari C. Devi
Department of Computer Science, University of North Carolina at Chapel Hill

Abstract

We consider schemes for enacting task share changes—a process called *reweighting*—on real-time multiprocessor platforms. Our particular focus is reweighting schemes that are deployed in environments in which tasks may *frequently* request *significant* share changes. Prior work has shown that fair scheduling algorithms are capable of reweighting tasks with minimal allocation error and that partitioning-based scheduling algorithms can reweight tasks with better average-case performance, but greater error. However, preemption and migration overheads can be high in fair schemes. In this paper, we consider the question of whether global scheduling techniques can improve the accuracy of reweighting relative to partitioning-based schemes and provide improved average-case performance relative to fair scheduled systems. Our conclusion is that, for soft real-time systems, global scheduling techniques provide a good mix of accuracy and average-case performance.

1 Introduction

Real-time systems that are *adaptive* in nature have received considerable recent attention [3, 9, 10, 4]. In addition, *multiprocessor* platforms are of growing importance, due to both hardware trends such as the emergence of multicore technologies and the prevalence of computationally-intensive applications for which single-processor designs are not sufficient. In prior work [3, 4], we considered the use of both fair and partitioning-based algorithms to schedule highly-adaptive workloads on (tightly-coupled) multiprocessor platforms, where the processor shares of tasks change frequently and to a significant extent. Fair scheduling techniques achieve high accuracy in enacting share changes, but do so at the expense of potentially frequent task preemptions and migrations among processors. Partitioning algorithms, in contrast, entail less overhead, but provide poorer (but sometimes acceptable) accuracy. The focus of this paper is adaptive global scheduling algorithms that avoid the high preemption and migration costs of fair scheduling techniques, yet have superior accuracy relative to partitioning-based schemes. The primary drawback of global scheduling algorithms is that, in order to fully utilize a multiprocessor system, bounded deadline misses must be acceptable. The key issue we address is whether the lower migration/preemption overheads and improved accuracy of such algorithms are sufficient to compensate for their inability to meet all deadlines.

Whisper. To motivate the need for this work, we consider two example applications under development at the University of North Carolina. The first of these is the Whisper tracking system, which performs full-body tracking in virtual environments [11]. Whisper tracks users via an array of wall- and ceiling-mounted microphones that detect white noise emitted from speakers attached to each user's hands, feet, and head. Like many tracking systems, Whisper uses *predictive techniques* to track objects. The workload on Whisper is intensive enough to necessitate a multiprocessor design. Furthermore, adaptation is required because the computational cost of making the "next" prediction in tracking an object depends on the accuracy of the previous one. Thus, the processor shares of the tasks that are deployed to implement these tracking functions will vary with time. In fact, the variance can be as much as *two orders of magnitude*. Moreover, adaptations must be enacted within *time scales as short as 10 ms.*

ASTA. The second application is the ASTA video-enhancement system [2]. ASTA is capable of improving the quality of an underexposed video feed so that objects that are indistinguishable from the background become clear and in full color. In ASTA, darker objects require more computation to correct. Thus, as dark objects move in the video, the processor shares of the tasks assigned to process different areas of the video will change. ASTA will eventually be deployed in a military-grade full-color night vision system, so tasks will need to change shares as fast as a soldier's head can turn. In the planned configuration, a 10-processor multicore platform will be used.

Dynamic sporadic tasks. In this paper, we are primarily concerned with *dynamic sporadic tasks*. Each such task T_i releases a sequence of *jobs*, T_i^1, T_i^2,... Each task is defined by the *execution cost* of each of its jobs, denoted $\mathsf{e}(T_i^j)$, and its *weight* at any time t, denoted $\mathsf{wt}(T_i, t)$, which specifies the fraction of a single processor it requires. This differs from the usual definition of a *sporadic* task, wherein per-job execution costs and weights do not change. While the terms "share," "weight," and "utilization" are often used interchangeably, we use *weight* to denote a task's desired utilization, and *share* to denote its actual guaranteed utilization. In each scheduling scheme we consider, a task's share is determined by its weight; in some of these schemes, the two are always equal, while in others, they may differ. We refer to the process of enacting task weight/share changes as *reweighting*.

Summary of results. In this paper, we consider five reweighting-capable scheduling algorithms: a previous fair algorithm developed by us called called PD^2-OF [3], which is a derivative of the PD^2 Pfair algorithm [1]; a previous partitioning-based algorithm developed by us called the *partitioned-adaptive scheduling* (PAS) algorithm [4]; the *non-preemptive-partitioned-adaptive-scheduling* (NP-PAS) algorithm, which is a non-preemptive variant of PAS; and two new algorithms proposed herein, the *changeable-earliest-deadline-first* (CNG-EDF) algorithm, which is a derivative of the well known global-earliest-deadline-first (EDF) algorithm, and the *non-preemptive-*

Scheme	Tardiness	Drift	Overload	Migrations	Preemptions
PD^2-OF	0	2	0	every quantum	every quantum
PAS	1	$\mathsf{e}_{\mathsf{max}}(T_i)$	W	weight-change events	weight-change events & job releases
NP-PAS	$\mathsf{e}(T_i^j)$ +$\mathsf{e}_{\mathsf{max}}(T_i)$+1	$\mathsf{e}_{\mathsf{max}}(T_i)$	W	weight-change events	weight-change events
CNG-EDF	$\kappa(m-1)$	$\mathsf{e}_{\mathsf{max}}(T_i)$	0	job releases	job releases
NP-CNG-EDF	$\kappa(m)$	$\mathsf{e}_{\mathsf{max}}(T_i)$	0	only in-between jobs	never

Table 1: Summary of worst-case results.

changeable-earliest-deadline-first (NP-CNG-EDF) algorithm, which is a non-preemptive variant of CNG-EDF.

Our results are summarized in Table 1, which lists the accuracy, migration cost, and preemption cost of each of the above schemes. Accuracy is assessed in terms of three quantities, "drift," "overload error," and "tardiness," which are measured in terms of the system's scheduling quantum size. *Drift* is the error, in comparison to an ideal allocation, that results due to a reweighting event [3]. (Under an ideal allocation, tasks are reweighted instantaneously, which is not possible in practice.) *Overload error*, which arises under partitioning-based schemes (see [4]), is the error that results from a scheduler's inability to allocate a task a share equal to its desired weight. *Tardiness* is the maximal amount by which any job can miss its deadline. Of these three types of error, overload error is potentially the most detrimental, since drift is a one-time error assessed per reweighting event and tardiness is bounded in the schemes we consider. Overload error, on the other hand, accumulates over time.

In Table 1, $\mathsf{e}_{\mathsf{max}}(T_i)$ denotes the *maximum execution cost* of any job of the task T_i, $\mathsf{wt}_{\mathsf{max}}(T_i)$ denotes the *maximal weight* of task T_i at any time, and W denotes the maximal weight of the $(m \cdot \lfloor 1/X \rfloor + 1)^{st}$ "heaviest" task (by maximal weight), where m is the number of processors and X is the maximal weight of the heaviest task. Furthermore,

$$\kappa(\ell) = \frac{\sum_{T_z \in \mathcal{E}_{\max}(T,\,\ell)} \mathsf{e}_{\mathsf{max}}(T_z)}{m - \sum_{T_z \in \mathcal{X}_{\max}(T,\,m-1)} \mathsf{wt}_{\mathsf{max}}(T_z)} + \mathsf{e}(T_i^j), \quad (1)$$

where $\mathcal{E}_{\max}(T, \ell)$ is the set of ℓ tasks in T with the highest *maximal* execution cost and $\mathcal{X}_{\max}(T, m-1)$ is the set of $m-1$ tasks in T with the heaviest *maximal* weight. (This bound is derived from prior work by Devi and Anderson on multiprocessor EDF scheduling [6].) Table 1 shows that algorithms that allow more frequent migrations and preemptions, like PD^2-OF, produce little drift, no overload error, and no tardiness; however, algorithms that restrict the frequency of migrations and preemptions can produce greater drift, overload error, and/or tardiness.

Contributions. Our theoretical contributions include devising CNG-EDF and NP-CNG-EDF reweighting rules, and establishing the error bounds for CNG-EDF and NP-CNG-EDF in Table 1. The question that then remains is: for the five aforementioned algorithms, how do drift, overload error, and tardiness compare to any error due to migration and preemption costs? We attempt to answer this question via extensive simulation studies of Whisper and ASTA. In these studies, real migration and preemption costs were assumed based on actual measured values. These studies confirm the expectation that, while CNG-EDF and NP-CNG-EDF provide a good compromise of accuracy and average case performance, *there exists no single "best" algorithm*: for each algorithm, application scenarios exist for which that algorithm is the best choice.

The rest of this paper is organized as follows. In Secs. 2 and 3, we discuss the CNG-EDF and NP-CNG-EDF algorithms in greater detail. Then, in Sec. 4, we establish the properties mentioned above. Our experimental evaluation is presented in Sec. 5. We conclude in Sec. 6.

2 System Model and Scheduling

In this section, we define our system model and the CNG-EDF and NP-CNG-EDF reweighting algorithms.

Sporadic task systems. We denote the i^{th} task of a task system T as T_i (where tasks are ordered by some arbitrary method), and denote the j^{th} job of the task T_i as T_i^j (where jobs are ordered by the sequence in which they are invoked). A *sporadic task* is defined by an *execution cost*, denoted $\mathsf{e}(T_i)$, and *weight*, denoted $\mathsf{wt}(T_i)$, which specifies the fraction of a single processor it requires. (It is customary to define a sporadic task by its execution cost and the minimum separation time between its successive jobs—we define the latter in terms of weight and execution cost below.) Fig. 1(a) depicts a one-processor system scheduled via EDF with four tasks, as defined in the figure's caption. (The other insets in the figure are considered later.) The first job of a task may be invoked or *released* at any time at or after time zero. The release time of job T_i^j is denoted $\mathsf{r}(T_i^j)$. Successive job releases of task T_i must be separated by at least $\mathsf{e}(T_i)/\mathsf{wt}(T_i)$ time. For example, in Fig. 1(a), $\mathsf{r}(T_1^1) = 0$ and $\mathsf{r}(T_1^2) = 3$. The *absolute deadline* (or just *deadline*) of job T_i^j, denoted $\mathsf{d}(T_i^j)$, equals $\mathsf{r}(T_i^j) + \mathsf{e}(T_i)/\mathsf{wt}(T_i)$. For example, in Fig. 1(a), $\mathsf{d}(T_1^1) = 3$ and $\mathsf{d}(T_1^2) = 6$. We consider a sporadic task T_i to be *active* at time t if there exists a job T_i^j (called T_i*'s active job*) such that $t \in [\mathsf{r}(T_i^j), \mathsf{d}(T_i^j))$.

Dynamic sporadic task systems. A *dynamic sporadic task system* is an extension of a sporadic task system, where the weight of each task T_i is a function of time t and its execution cost can vary with each job T_i^j. We use $\mathsf{wt}(T_i, t)$ and $\mathsf{e}(T_i^j)$, respectively, to denote these two quantities. (For the remainder of the paper, whenever we refer to a "task" we are referring to a "dynamic sporadic task.") We use $\mathsf{wt}_{\mathsf{min}}(T_i)$ ($\mathsf{wt}_{\mathsf{max}}(T_i)$) to denote the *minimum* (*maximum*) *allowed weight for* T_i. As a shorthand, we use T_i:$[a, b]$ to denote a task T_i such that $\mathsf{wt}_{\mathsf{min}}(T_i) = a$ and $\mathsf{wt}_{\mathsf{max}}(T_i) = b$, and T_i:a to denote T_i:$[a, a]$. Furthermore, we use $\mathsf{e}_{\mathsf{max}}(T_i)$ to denote the maximal execution cost of any job of T_i. Fig. 1(b) gives an example.

For dynamic sporadic tasks, the *absolute deadline* of a job T_i^j equals $\mathsf{r}(T_i^j) + \mathsf{e}(T_i^j)/\mathsf{wt}(T_i, \mathsf{r}(T_i^j))$. In the absence of reweighting, consecutive job releases ($\mathsf{r}(T_i^j)$ and $\mathsf{r}(T_i^{j+1})$) of a task T_i must be separated by at least $\mathsf{e}(T_i^j)/\mathsf{wt}(T_i, \mathsf{r}(T_i^j))$. For example,

in Fig. 1(b), $\mathsf{r}(T_1^2) - \mathsf{r}(T_1^1) = 2/(1/3) = 6$, $\mathsf{r}(T_1^3) - \mathsf{r}(T_1^2) = 2/(1/2) = 4$, and $\mathsf{d}(T_1^3) = 10 + 1/(1/2) = 12$.

A task T_i *changes weight* or *reweights* at time t if $\mathsf{wt}(T_i, t-\epsilon) \neq \mathsf{wt}(T_i, t)$ where $\epsilon \to 0^+$. If a task T_i changes weight at a time t_c between the release and the deadline of some job T_i^j, then the following three actions *may* occur:

- The execution cost of T_i^j *may* be reduced to the amount of time for which T_i^j has executed prior to t_c.
- $\mathsf{r}(T_i^{j+1})$ *may* be less than $\mathsf{r}(T_i^j) + \mathsf{e}(T_i^j)/\mathsf{wt}(T_i, \mathsf{r}(T_i^j))$.
- If T_i^{j+1} is released before $\mathsf{r}(T_i^j) + \mathsf{e}(T_i^j)/\mathsf{wt}(T_i, \mathsf{r}(T_i^j))$, then since $\mathsf{d}(T_i^j) = \mathsf{r}(T_i^j) + \mathsf{e}(T_i^j)/\mathsf{wt}(T_i, \mathsf{r}(T_i^j))$, jobs T_i^j and T_i^{j+1} will "overlap." (In the variant of the sporadic model defined earlier, every job's deadline is at or before its successors's release.) Hence, we say that a job T_i^j is *active* at time t iff $t \in [\mathsf{r}(T_i^j), \mathsf{min}(\mathsf{r}(T_i^{j+1}), \mathsf{d}(T_i^j)))$.

The reweighting rules we present in Sec. 3 state under what conditions the above actions occur and by how much before $\mathsf{r}(T_i^j) + \mathsf{e}(T_i^j)/\mathsf{wt}(T_i, \mathsf{r}(T_i^j))$ the job T_i^{j+1} can be released. Since a reweighting event may cause a job's execution cost to decrease, we introduce the notion of a job T_i^j's *actual execution cost*, denoted $\mathsf{ae}(T_i^j)$, which represents the total amount of execution time that T_i^j will receive.

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_i at time t*, denoted $\mathsf{swt}(T_i, t)$, to represent the "last enacted weight of T_i". Formally, $\mathsf{swt}(T_i, t)$ equals $\mathsf{wt}(T_i, u)$, where u is the last time at or before t that a weight change was enacted for T_i. It is important to note that, *henceforth, we compute task deadlines and releases using scheduling weights.*

Scheduling. Under both CNG-EDF and NP-CNG-EDF, "ready" jobs are prioritized by deadline, with earlier deadlines having higher priority. ("Ready" will be formally defined shortly.) Deadline ties are resolved arbitrarily, but consistently. Under CNG-EDF, an arriving job with higher priority preempts the executing job with the lowest priority if no processor is available. The preempted job may later resume execution on a different processor. Under NP-CNG-EDF, the arriving job waits until some job completes execution and a processor becomes available. Thus, under NP-CNG-EDF, once scheduled, a job is guaranteed execution until completion without interruption. Fig. 1(b) depicts a CNG-EDF schedule of the task system T described above, and Fig. 1(c) depicts a NP-CNG-EDF schedule of the same system.

For an arbitrary scheduling algorithm $\mathcal{A}$ and an arbitrary task system T, we let $\mathcal{S}$ denote an m-processor schedule $\mathcal{A}$ of T, and let $\mathsf{A}(\mathcal{S}, T_i^j, t_1, t_2)$ denote the total time allocated to T_i^j in $\mathcal{S}$ in $[t_1, t_2)$. Similarly, we use $\mathsf{A}(\mathcal{S}, T_i, t_1, t_2)$ and $\mathsf{A}(\mathcal{S}, T, t_1, t_2)$, respectively, to denote the total time allocated to all jobs of T_i in $\mathcal{S}$ and all tasks of T in $\mathcal{S}$, over the interval $[t_1, t_2)$. We say that the value of $\mathsf{A}(\mathcal{S}, T_i^j, 0, t)$ is the amount that T_i^j has *executed by* t. For example in Fig. 1(b), $\mathsf{A}(\mathcal{S}, T_1^1, 0, 6) = 2$, $\mathsf{A}(\mathcal{S}, T_1^1, 0, 12) = 2$, and $\mathsf{A}(\mathcal{S}, T_1^1, 3, 12) = 0$.

Definition 1 (Halted). As discussed later, if a reweighting event in schedule $\mathcal{S}$ occurs at time t, then it is possible that some job T_i^j is *halted* at t. In this case, $\mathsf{ae}(T_i^j)$ is set to $\mathsf{A}(\mathcal{S}, T_i^j, 0, t)$.

Definition 2 (Completed). If $\mathcal{S}$ is an m-processor CNG-EDF or NP-CNG-EDF schedule of the task system T, then a job $T_i^j \in T$ is said to have *completed by time t in $\mathcal{S}$* iff T_i^j has executed for $\mathsf{e}(T_i^j)$ by t in $\mathcal{S}$, or T_i^j has halted by time t. A task T_i is said to have *completed at time t in $\mathcal{S}$* if at time t every job of T_i that has been released by t has completed. A task T_i is said to have *entirely completed by time t* iff *all* jobs of T_i in T have completed. For example, in Fig. 1(b), T_1^1 has completed by time 3, T_1 is complete (but *not* entirely complete) at time 3 but not complete at time 6, and T_4 is entirely complete at time 4.

Definition 3 (Pending and Ready). For an arbitrary scheduling algorithm $\mathcal{A}$, if $\mathcal{S}$ is an m-processor schedule of the task system T under $\mathcal{A}$, then a job T_i^j is said to be *pending at time t in $\mathcal{S}$* if $\mathsf{r}(T_i^j) \leq t$ and T_i^j is not complete by t in $\mathcal{S}$. For example, in Fig. 1(a), the job T_2^1 is pending over the range $[0, 9)$. Note that a job can be pending, but not active, if it misses its deadline. A pending job T_i^j is said to be *ready at time t in $\mathcal{S}$* if all prior jobs of task T_i have completed by t. For example, in Fig. 1(a), the job T_2^1 is ready over the range $[0, 9)$. A job T_i^j can be pending but not ready if T_i^{j-1} has not completed by $\mathsf{r}(T_i^j)$.

3 Task Reweighting

We now introduce two new reweighting rules that are CNG-EDF extensions of the PD^2-OF reweighting rules presented by us previously [3]. As mentioned before, these rules work by modifying future job release times and deadlines. At the end of this section, we discuss how to adjust these rules for NP-CNG-EDF.

For simplicity, we assume that the actual execution cost for any job is equal to its specified execution cost, *unless* a task reweights while a job is active. Then *and only then* can the actual execution cost of a job be less than its execution cost. (This assumption can be removed at the expense of more complicated notation.) In this scenario, the actual execution cost of the job is determined by the rules we present shortly.

Let T be a task system in which some task T_i initiates a weight change from weight w to weight v at time t_c. Let $\mathcal{S}$ be the m-processor CNG-EDF schedule of T. Let T_i^j be the active job of T_i at t_c. If $\mathsf{e}(T_i^j) - \mathsf{A}(\mathcal{S}, T_i^j, 0, t) > 0$, then let $\mathsf{rem}(T_i^j, t_c) = \mathsf{e}(T_i^j) - \mathsf{A}(\mathcal{S}, T_i^j, 0, t)$; otherwise, $\mathsf{rem}(T_i^j, t_c) = \mathsf{e}(T_i^{j+1})$. Note that $\mathsf{rem}(T_i^j, t_c)$ denotes the actual remaining computation in T_i's current job or the size of T_i's next job if the current job has completed. The *deviance of job T_i^j of task T_i at time t* is defined as $\mathsf{dev}(T_i^j, t) = \int_{\mathsf{r}(T_i^j)}^{t} \mathsf{swt}(T_i, u)du - \mathsf{A}(\mathcal{S}, T_i^j, 0, t)$. The choice of which rule to apply depends on whether deviance is positive or negative. If positive, then we say that T_i is *positive-changeable at time t_c from weight w to v*; otherwise T_i is *negative-changeable at time t_c from weight w to v*. Because T_i initiates its weight change at t_c, $\mathsf{wt}(T_i, t_c) = v$ holds; however, T_i'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_i,

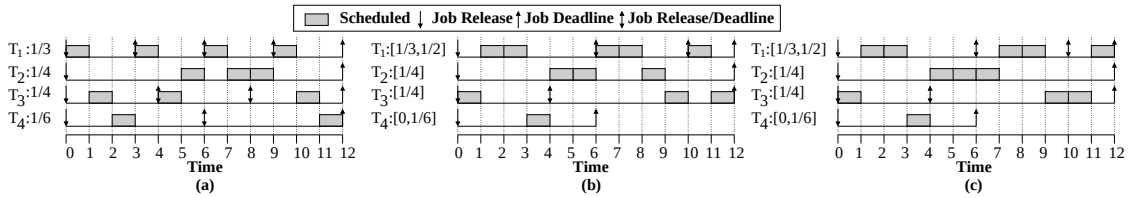

Figure 1: A one-processor (sporadic or dynamic sporadic) system T with four tasks. Inset **(a)** depicts an EDF schedule T where the tasks are defined as follows: T_1 with weight 1/3 and $\mathsf{e}(T_1) = 1$, T_2 with weight 1/4 and $\mathsf{e}(T_2) = 3$, T_3 with weight 1/4 and $\mathsf{e}(T_3) = 1$, and T_4 with weight 1/6 and $\mathsf{e}(T_4) = 1$. In insets **(b)** and **(c)**, the tasks are defined as follows: T_1 has an initial weight of 1/3 and increases to 1/2 at time 6, $\mathsf{e}(T_1^1) = \mathsf{e}(T_1^2) = 2$, and $\mathsf{e}(T_1^3) = 1$; T_2 has a constant weight of 1/4 and $\mathsf{e}(T_2^1) = 3$; T_3 has a constant weight of 1/4, $\mathsf{e}(T_3^1) = 1$, and $\mathsf{e}(T_3^2) = 2$; and T_4 has an initial weight of 1/6 and decrease to 0 at time 6 (*i.e.*, T_4 "leaves" the system at 6) and $\mathsf{e}(T_4^1) = 1$. Inset (b) depicts a CNG-EDF schedule of T. Inset (c) depicts a NP-CNG-EDF schedule of T. All ties are broken in favor of the task with the lower index.

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 P: If T_i is positive-changeable at time t_c from weight w to v, then one of two actions is taken: **(i)** if $\mathsf{d}(T_i^j) > \mathsf{rem}(T_i, t_c)/v$, then T_i^j is halted, its weight change is enacted, and a new job of size $\mathsf{rem}(T_i, t_c)$ is issued for it with a release time of t_c; **(ii)** otherwise, its weight change is enacted at time $\mathsf{d}(T_i^j)$, *i.e.*, the scheduling weight does not change until the end of the current job.

Rule N: If T_i is negative-changeable at time t_c from weight w to v, then one of two actions is taken: **(i)** if $v > w$, then T_i^j is halted, its weight change is enacted, and a new job of size $\mathsf{rem}(T_i, t_c)$ is issued for it with a release time equal to the time t at which $\mathsf{dev}(T_i^j, t) = 0$ holds; **(ii)** otherwise, the weight change is enacted at time $\mathsf{d}(T_i^j)$.

Intuitively, Rule P changes a task's weight by halting its current job and issuing a new job of size $\mathsf{rem}(T_i, t_c)$ with the new weight if doing so would improve its deadline. A (one-processor) example of a positive-changeable task is given in Fig. 2(a). (We discuss the terms drift, IDEAL allocations, and SW allocations in Sec. 4.) The depicted example consists of a task system T with four tasks as defined in the figure's caption. Note that, since T_2, T_3, and T_4 have the same deadline, we have arbitrarily chosen T_4 to have the lowest priority. In inset (a), T_4 is positive-changeable since at time 2 it has not yet been scheduled. Note that halting T_4's current job and issuing a new job of size one improves T_4's scheduling priority, *i.e.*, $\mathsf{d}(T_4^1) = 6 > \frac{7}{2} = \mathsf{d}(T_4^2)$. Notice that the second job of T_4 is issued 6/4 quanta after time 2. This spacing is in keeping with a new job of weight 4/6 issued at time 2.

Rule N changes the weight of a task by one of two approaches: **(i)** if a task *increases* its weight, then Rule N adjusts the release time of its next job so that it is commensurate with the new weight; **(ii)** if a task *decreases* its weight, then Rule N waits until the end of the current job and then issues the next job with a deadline that is commensurate with the new weight. A (one-processor) example of a negative-changeable task that increases its weight is given in Fig. 2(b). The depicted example consists of the same tasks as in (a), except that we have chosen T_4 to have the highest priority. Notice that the second job of T_4 is issued at time 3, which is the time such that $\mathsf{dev}(T_4, 3) = \int_0^3 \mathsf{swt}(T_i, u)du - \mathsf{A}(\mathcal{S}, T_4, 0, 3) = 1 - 1 = 0$. Recall that the deadline (release time) of the i^{th} ($(i+1)^{th}$) job of a task T_j is given by $\mathsf{r}(T_i^j) + \mathsf{e}(T_i^j)/(\mathsf{swt}(T_i, \mathsf{r}(T_i^j)))$. Hence, if a task T_i of weight v were to issue a job of size $y = \mathsf{A}(\mathcal{S}, T_i^j, 0, t_c) - \mathsf{dev}(T_i^j, t)$ at time t_c, then the release time of its next job would be $t_c + y/v$. A (one-processor) example of a negative-changeable task that decreases its weight is given in Fig. 2(c). The depicted example consists of the same four tasks except that T_4 has an initial weight of 4/6 and decreases its weight at time 1, and T_1 joins the system as soon as T_4's weight change is enacted.

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.

Modifications for NP-CNG-EDF**.** In order to adapt the rules P and N to work for NP-CNG-EDF, the only modification we need to make is when these rules are initiated. If a task reweights *before or after* the active job has been scheduled, then the rules P and N are initiated as before. (Note that if the active job *has not* been scheduled, then its deviance is positive, and if the active job *has* been scheduled, its deviance is negative.) However, if a task changes its weight while the active job T_i^j is executing, then the initiation of the weight change is delayed *until* T_i^j *has completed* or T_i^j *is no longer active*, whichever is first. Note that when a task T_i changes its weight from u to v at time t_c in NP-CNG-EDF, then $\mathsf{wt}(T_i, t_c) = v$ holds, regardless of whether the initiation of rule P or N must be delayed.

4 Tardiness and Drift Bounds

In this section, we formally present and prove tardiness and drift bounds for the CNG-EDF algorithm. Because any set of reweighting rules will cause the "actual" schedule to deviate from the "ideal" schedule, the tardiness bounds reflect CNG-EDF's accuracy at *scheduling* the job-set created by CNG-EDF. The drift bounds, on the other-hand, reflect CNG-EDF's accuracy at creating a job-set that mimics the "ideal" task system, where weight changes can always be initiated and enacted instantaneously. To this end, we introduce two new theoretical scheduling algorithms: the *scheduling-weight processor-sharing* (SW) scheduling algorithm and the *ideal processor-sharing* (IDEAL) scheduling algorithm. Both algorithms have the ability to preempt and swap tasks at arbitrarily small intervals. However, SW allocates each task a share equal to its *scheduling weight*; moreover, SW *will not allocate* capacity to a task if its active job has received an allocation equal to its actual execution cost. IDEAL, on the other hand, al-

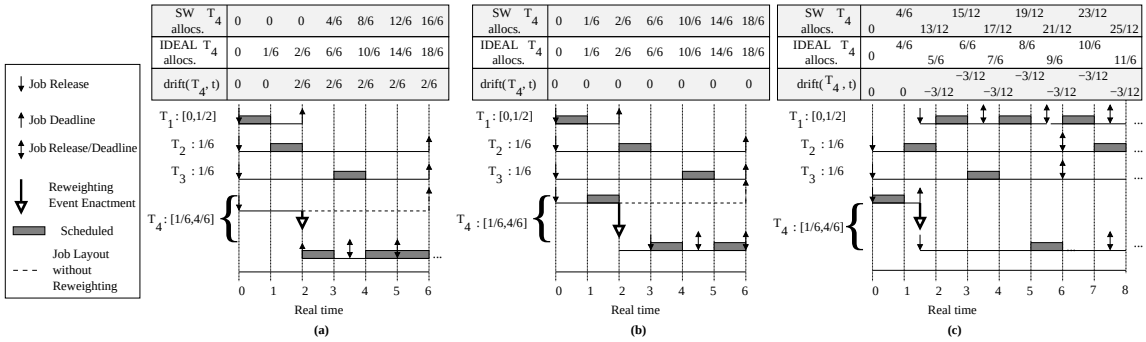

Figure 2: A one-processor system consisting of four tasks, T_1:$[0, 1/2]$, T_2:$1/6$, T_3:$1/6$, and T_4:$[1/6, 4/6]$, where the execution cost of every job is one. The dotted lines represent the interval up to T_4's next deadline, which due to reweighting has been changed (as indicated by the solid arrow). The drift, allocations in IDEAL, and allocations in SW for T_4 are labeled as a function of time across the top. **(a)** The CNG-EDF schedule for the scenario where T_1 is in the system initially and leaves at time 2, T_4 has an initial weight of 1/6 that increases to 4/6 at time 2, and T_4 has the lowest scheduling priority. Since T_4 is not scheduled by time 2, it has positive deviance and changes its weight via Rule P, causing T_4^1 to be halted, T_4^2 to be released at 2 with a deadline of 9/2, and T_4's drift to become 2/6. **(b)** The same scenario as in (a) except that T_4 has higher priority than both T_2 and T_3. Since T_4 has been scheduled by time 2, it has negative deviance and changes its weight via Rule N, causing its next job to have a release time of 3 while maintaining a drift of zero. **(c)** T_1 joins the system at time 6/4 and T_4 has an initial weight of 4/6 that decreases to 1/6 at time 1. Since T_4 has negative deviance at time 1, it is changed via Rule N, causing T_4's next job to have a deadline of 15/2 and T_4 to have a drift of $-3/12$.

locates each task a share equal to its *weight* at each instant; and, unlike SW, IDEAL *will not stop allocating* capacity to a task *unless* that task has received an allocation equal to the total execution cost of all of its jobs. (For simplicity, we have assumed that every job 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 IDEAL would equal zero between active jobs.) We provide below a more in-depth explanation of these two algorithms.

4.1 Tardiness and Lag

We begin by defining the SW scheduling algorithm.

The SW scheduling algorithm. In order to establish tardiness bounds for CNG-EDF, we compare allocations produced by CNG-EDF to those produced by SW. Under SW, at each instant t, each non-complete job of each task T_i is allocated a fraction of a processor equal to $\mathsf{swt}(T_i, t)$. Furthermore, we consider SW to be "clairvoyant" in the sense that SW can use the value of $\mathsf{ae}(T_i^j)$ to determine if T_i^j has completed before it has halted. More specifically, for any schedule $\mathcal{SW}$ under SW of any task system T, we say that T_i^j has *completed by time t in $\mathcal{SW}$* iff T_i^j has executed for $\mathsf{ae}(T_i^j)$ by t.

For example, consider the one-processor task system T depicted in Fig. 3. Inset (a) depicts a CNG-EDF schedule and insets (b) depicts T's SW schedule. Notice that in the SW schedule T_1 does not receive any allocations over the interval $[3, 6)$. This is because at time 3 the total allocation to T_1^1 in the SW schedule equals $\mathsf{ae}(T_1^1) = 1$, hence, T_1^1 is complete at time 3. However, at time 6, T_1^2 is released, and therefore T_1 has an incomplete job with a scheduling weight of 1/2. Hence, T_1 begins to receive allocations equal to its scheduling weight, which is now 1/2. Note that we assume that every job release, deadline, execution cost, and actual execution cost for a SW schedule to be the same as that in CNG-EDF.

Lag. If $\mathcal{S}$ is an m-processor schedule under CNG-EDF of the task system T and $\mathcal{SW}$ is an m-processor schedule under SW of the same task system T, then the *lags at time t of a job T_i^j, task T_i, and task system T*, respectively, are defined by Eqns. (2)–(4).

$$\mathsf{lag}(T_i^j, t) = \mathsf{A}(\mathcal{SW}, T_i^j, 0, t) - \mathsf{A}(\mathcal{S}, T_i^j, 0, t) \quad (2)$$

$$\mathsf{lag}(T_i, t) = \mathsf{A}(\mathcal{SW}, T_i, 0, t) - \mathsf{A}(\mathcal{S}, T_i, 0, t) \quad (3)$$

$$\mathsf{LAG}(T, t) = \mathsf{A}(\mathcal{SW}, T, 0, t) - \mathsf{A}(\mathcal{S}, T, 0, t) \quad (4)$$

Note that $\mathsf{LAG}(T, t) = \sum_{T_i \in T} \mathsf{lag}(T_i, t)$. The lag of a job (or task or system) represents by how much a job (or task or system) is under/over-allocated compared to the SW schedule at time t. For example, in Fig. 3, $\mathsf{lag}(T_3^1, 1) = 1/4 - 0 = 1/4$, $\mathsf{lag}(T_3^1, 2) = 2/4 - 1 = -1/2$, $\mathsf{lag}(T_1^1, 2) = 2/3 - 0 = 2/3$, $\mathsf{lag}(T_1^1, 3) = 3/3 - 0 = 1$, and $\mathsf{lag}(T_1^1, 6) = 3/3 - 1 = 0$.

4.2 Tardiness Proof

In prior work, Devi and Anderson [6] proved that in any m-processor EDF schedule of a *sporadic* task system T (where the total weight of all tasks is at most m) the tardiness of each job of any task T_i is at most $\kappa(m-1)$ where, $\kappa(m-1)$ is as defined in (1). Their proof consists primarily of three lemmas/theorems: **(i)** if the LAG of T is bounded in the m-processor EDF schedule $\mathcal{S}$ of T, then tardiness is bounded; **(ii)** the LAG of T in $\mathcal{S}$ is bounded; **(iii)** by (i) and (ii), the tardiness of each job of any task T_i in T is at most $\kappa(m-1)$.

Since Devi and Anderson were proving tardiness bounds for a sporadic task system, they were able to utilize the fact that a job T_i^j and its successor T_i^{j+1} do not "overlap," *i.e.*, $\mathsf{d}(T_i^j) \leq \mathsf{r}(T_i^{j+1})$ holds for any sporadic task T_i. However, this property can be weakened without affecting their proof (barring some minor notational changes), so that their proof can be adapted to prove tardiness bounds for a *dynamic* sporadic task system. Specifically, the Devi and Anderson proof can be used to show that the tardiness of CNG-EDF is bounded by $\kappa(m-1)$. If the following properties hold.

(W) $\sum_{T_i \in T} \mathsf{wt}(T_i, t) \leq m$ for all t.

(V) For any job T_i^j and its successor T_i^{j+1}, if $\mathsf{d}(T_i^j) > \mathsf{r}(T_i^{j+1})$,

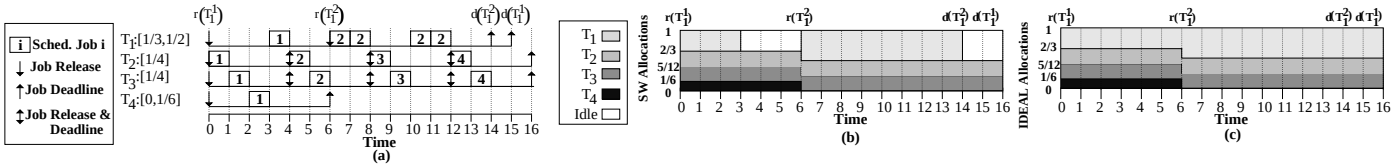

Figure 3: A one-processor task system T with four tasks: $\mathsf{e}(T_1^1) = 5$ and T_1 has an initial weight of 1/3 that increases to 1/2 via case (i) of rule P at time 6 and as a result, $\mathsf{ae}(T_1^1) = 1$, $\mathsf{e}(T_1^2) = 4$, $\mathsf{r}(T_1^2) = 6$ and $\mathsf{d}(T_1^2) = 14$; T_2 has a constant weight of 1/4 and a constant execution cost of 1; T_3 has a constant weight of 1/4 and a constant execution cost of 1; and T_4 has an initial weight of 1/6 that decreases to 0 at time 6 and $\mathsf{e}(T_4^1) = 1$. Inset **(a)** depicts the CNG-EDF schedule of T. The number in each box denotes which job is scheduled, *e.g.*, over the range $[0, 1)$, T_2^1 is executing, and over the range $[4, 5)$, T_2^2 is executing. Inset **(b)** depicts the SW schedule of T. Note that since $\mathsf{ae}(T_1^1) = 1$ at time 3, T_1^1 is complete in SW, *i.e.*, T_1 receives no allocations under SW over the range $[3, 6)$. Inset **(c)** depicts the IDEAL schedule of T. Note that the IDEAL allocation to any task T_i equals $\int_{t_1}^{t_2} \mathsf{wt}(T_i, u)du$ if T_i is active over the range $[t_1, t_2)$. For insets (b) and (c), the releases and deadlines of the jobs T_1^1 and T_1^2 are depicted.

then T_i^j, must have completed before $\mathsf{r}(T_i^{j+1})$ in both the CNG-EDF and SW schedules of T.

Since (W) can be easily satisfied, for the remainder of this subsection, we show that CNG-EDF satisfies property (V). (Unfortunately, due to space constraints, we are not able to present the Devi and Anderson proof with the necessary (minor) adjustments in the body of this paper. Therefore, we have placed this proof in an appendix to this paper, which can be found on the author's web-page at `http://www.cs.unc.edu/~anderson/papers.html`.) In order to show that property (V) holds, we show that for any job T_i^j in an arbitrary dynamic sporadic task system T, if $\mathsf{d}(T_i^j) > \mathsf{r}(T_i^{j+1})$, then T_i^j must have completed before $\mathsf{r}(T_i^{j+1})$ in both the CNG-EDF and SW schedules of T. To this end, let $\mathcal{S}$ be the m-processor CNG-EDF schedule of some dynamic task system T, where $\sum_{T_i \in T} \mathsf{wt}(T_i, t) \leq m$ for all t, and let $\mathcal{SW}$ be the m-processor SW schedule of the same task system.

Lemma 1. *For a task T_i, if $\mathsf{r}(T_i^{j+k}) < \mathsf{d}(T_i^j)$, where $j, k \geq 1$, then T_i^j will have completed by $\mathsf{r}(T_i^{j+k})$ in $\mathcal{S}$ and $\mathcal{SW}$.*

Proof. Suppose that $\mathsf{r}(T_i^{j+k}) < \mathsf{d}(T_i^j)$ holds. By the definition of $\mathsf{d}(T_i^j)$, the minimum separation between job releases, and rules P and N, $\mathsf{r}(T_i^{j+k}) < \mathsf{d}(T_i^j)$ holds only if T_i reweighted and halted while T_i^j was active. Without loss of generality, let t_c be the earliest such time. Then, by the rules P and N, $\mathsf{r}(T_i^{j+k}) \geq t_c$. Hence, T_i^j will have halted and thus completed by $\mathsf{r}(T_i^{j+k})$ in $\mathcal{S}$.

It remains to be shown that T_i^j will have completed by $\mathsf{r}(T_i^{j+k})$ in $\mathcal{SW}$. Since T_i^j is halted at t_c, it must be the case that T_i changed its weight via case (i) of rule P or N at t_c. However, both cases follow easily by the clairvoyant nature of $\mathcal{SW}$. □

Modifications for NP-CNG-EDF. In NP-CNG-EDF, if a job is released and is ready at time t, and the newly-released job has a deadline that is earlier than some other job executing at t, the newly released job cannot preempt the lower-priority job. If no processor is available at t, then this will lead to a *priority inversion*. In such a scenario, the waiting ready, higher-priority job is referred to as a *blocked job*, and the executing lower-priority job is referred to as a *blocking job*. A task T_i is said to be *blocked* at t if T_i is not executing at t and the earliest pending job (*i.e.*, the ready job) of T_i has a higher priority than at least one job executing at t. For example, in Fig. 1(c), T_2 is the blocking job over the interval $[6, 7)$, and T_1 is the blocked job over the same interval.

The major difference between Devi and Anderson's tardiness-bound proof for EDF and NP-EDF for sporadic tasks is that in their NP-EDF proof they calculate the upper bound on the length of time for which a task can be blocked. Because of the blocking factor, the bound they construct for NP-EDF is $\kappa(m)$. As before, Devi and Anderson in their proof for NP-EDF rely on the property of sporadic tasks that consecutive jobs do not "overlap." And, as before, this requirement can be weakened without affecting their proof, so that their proof holds for a *dynamic* sporadic task set, so long as conditions (V) and (W) hold. Since the reweighting rules for NP-CNG-EDF are essentially the same as the reweighting rules for CNG-EDF, by Lem. 1, conditions (V) and (W) hold for any m-processor NP-CNG-EDF schedule, of any task set T so long as $m \leq \sum_{T_i \in T} \mathsf{wt}(T_i, t)$ holds for all t. (As before, due to space constraints we are forced to present the Devi and Anderson proof, with its modification, in its entirety in an appendix found on the author's web page.) Hence, NP-CNG-EDF's tardiness bound is $\kappa(m)$.

4.3 Drift

We now turn our attention to the issue of measuring "drift" under CNG-EDF. In order to measure the "drift" of a task system T, we compare the SW schedule of T to that of an "ideal" reweighting scheme that enacts reweighting changes instantaneously. Under the *ideal processor sharing* (IDEAL) scheduling algorithm, at each instant t, each task T_i in T is allocated a share equal to its weight $\mathsf{wt}(T_i, t)$. Hence, if $\mathcal{I}$ is the IDEAL schedule of T, then over the interval $[t_1, t_2)$, the task T_i is allocated $\mathsf{A}(\mathcal{I}, T_i^j, t_1, t_2) = \int_{t_1}^{t_2} \mathsf{wt}(T_i, u)du$ time. As we mentioned earlier, IDEAL is similar to SW, with two major exceptions: **(i)** under IDEAL, each task receives an allocation equal to its *weight*, whereas under SW, each task receives an allocation equal to its *scheduling weight*; and **(ii)** under IDEAL, a task does not stop receiving allocations unless its total allocation equals the total execution cost of all of its jobs, whereas under SW, a task will stop receiving allocations if its active job has received an allocation equal to its actual execution cost. For example, consider the IDEAL schedule of the task system T depicted in Fig. 3(c). Notice that, over the range $[3, 6)$, the task T_1 receives allocations equal to its weight at every instant. Compare this to the SW schedule (inset (b)), in which T_1 receives *no* allocations over the range $[3, 6)$.

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. For example, under a *uniprocessor* EDF (*i.e.*, CNG-EDF without weight changes) schedule, the difference between the ideal and actual allocations for a task lies within $(-\mathsf{e_{max}}(T_i), \mathsf{e_{max}}(T_i))$. When a weight change occurs, the same bounds are maintained except that they may be centered at a different value. For example, in Fig. 2(a), the range is originally $(-1, 1)$, but after the reweighting event, it is $(-4/6, 8/6)$. This lost allocation is called *drift*. Given this loss (barring further reweighting events) T_i's drift will not change. In general, a task's drift per reweighting event will be non-negative (non-positive) if it increases (decreases) its weight. Under CNG-EDF, the drift of a task T_i at time t is defined as

$$\mathsf{drift}(T_i, t) = \mathsf{A}(\mathcal{I}, T_i^j, 0, u) - \mathsf{A}(\mathcal{SW}, T_i^j, 0, u), \qquad (5)$$

where $\mathcal{SW}$ is the schedule of T under SW, $\mathcal{I}$ is the schedule of T under IDEAL, and u is the last time a reweighting event of T_i was enacted before t.

Theorem 1. *The absolute value of the per-event drift under* CNG-EDF *for each task* T_i *is less than* $\mathsf{e_{max}}(T_i)$.

Proof Sketch. If a task T_i changes its weight at time t_c via rule P, then when this weight change is enacted at time t_e (*i.e.*, at t_c under case (i) or at $\mathsf{d}(T_i^j)$ under case (ii)), then it is as though allocation equal to $\mathsf{A}(\mathcal{I}, T_i^j, \mathsf{r}(T_i^j), t_e) - \mathsf{A}(\mathcal{SW}, T_i^j, \mathsf{r}(T_i^j), t_e)$ is "lost." For example in Fig. 2(a), the task T_4 "loses" an allocation of 2/6. Since this value (per reweighting event) is always less than $\mathsf{e_{max}}(T_i)$, the absolute value of drift is less than $\mathsf{e_{max}}(T_i)$.

If a task T_i changes its weight at time t_c via rule N, and T_i decreases its weight (case (ii)), then the weight change will be enacted at $\mathsf{d}(T_i^j)$. Since the maximum allocation T_i can receive in $\mathcal{SW}$ during T_i^j is $\mathsf{e_{max}}(T_i)$, $\mathsf{A}(\mathcal{SW}, T_i^j, t_c, \mathsf{d}(T_i^j)) - \mathsf{A}(\mathcal{I}, T_i^j, t_c, \mathsf{d}(T_i^j)) \leq \mathsf{e_{max}}(T_i)$. Thus, the absolute value of the drift incurred is at most $\mathsf{e_{max}}(T_i)$. For example, in Fig. 2(c), the drift incurred by T_4 is $-3/12$, *i.e.*, $\mathsf{drift}(T_4, t) = -3/12$, where $t \geq 3/2$. If T_i increases its weight (case (i)), then it incurs zero drift, since it *immediately* enacts the weight change (*i.e.*, the scheduling weight changes immediately). Hence, the absolute value of the drift incurred by this reweighting event is less than $\mathsf{e_{max}}(T_i)$. For example, in Fig. 2(b), the drift incurred by T_4 is 0, *i.e.*, $\mathsf{drift}(T_4, t) = 0$, where $t \geq 2$. □

Modifications for NP-CNG-EDF**.** Note that delaying the initiation of a reweighting event does not substantially increase the drift incurred per reweighting event, since the longest a reweighting event can be delayed is the execution cost of the active job. If T_i^j is the active job of T_i at t_c, and if T_i's reweighting event is delayed until some time t, then at t either **(i)** T_i^j has a non-positive deviance (*i.e.*, T_i^j completes before its deadline), or **(ii)** T_i^j is not active at t (*i.e.*, T_i^j does not complete before its deadline, and thus is not active at t). In either case, the active job (if it exists) is negative-changeable. Hence, if the task increases its weight, then the only drift the task will incur for this reweighting event results from delaying the initiation of its reweighting event, *i.e.*, at most $\mathsf{e_{max}}(T_i)$. If T_i decreases its weight, then delaying the reweighting event will not affect drift, since the enactment of the reweighting event would occur at $\mathsf{d}(T_i^j)$ regardless of whether the initiation of the reweighting event was delayed or not.

5 Experimental Results

The results of this paper are part of a longer-term project on adaptive real-time allocation in which both Whisper and ASTA described earlier, will be used as test applications. In this section, we provide extensive simulations of Whisper and ASTA as scheduled by PD^2-OF, PAS, NP-PAS, CNG-EDF, and NP-CNG-EDF.

Whisper. As noted earlier, Whisper tracks users via speakers that emit white noise attached to each user's hands, feet, and head. Microphones located on the wall or ceiling receive these signals and a tracking computer calculates each speaker's position by measuring signal delays. 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. By varying the number of samples, Whisper can trade measurement accuracy for computation—with more samples, the more accurate and more computationally intensive the calculation. As a signal becomes weaker, the number of samples is increased to maintain the same level of accuracy. As the distance between a speaker and microphone increases, the signal strength decreases. This behavior (along with the use of predictive techniques mentioned in the introduction) can cause task-share changes of up to two orders of magnitude every 10ms. Since Whisper continuously performs calculations on incoming data, at any point in time, it does not have a significant amount of "useful" data stored in cache. As a result, migration/preemption costs in Whisper are fairly small (at least, on a tightly-coupled system, as assumed here, where the main cost of a migration is a loss of cache affinity). In addition, fairness and real-time guarantees are important due to the inherent "tight coupling" among tasks required to accurately perform triangulation calculations.

ASTA system. Before describing ASTA in detail, we review some basics of videography. All video is a collection of still images called *frames*. Associated with each frame is an *exposure time*, which denotes the amount of time the camera's shutter was open while taking that frame. Frames with faster exposure times capture moving objects with more detail, while frames with slower exposure times are brighter. If a frame is *underexposed* (*i.e.*, the exposure time is too fast), then the image can be too dark to discern any object. The ASTA system can correct underexposed video while maintaining the detail captured by faster exposure times by combining the information of multiple frames. To intuitively understand how ASTA achieves this behavior, consider the following example. If a camera, **A**, has an exposure time of $1/30^{th}$ of a second, and a second camera, **B**, has an exposure time of $1/15^{th}$ of a second, then for every two frames shot by camera **A** the shutter is open for the same time as one frame shot by **B**. ASTA is capable of exploiting this observation in order to allow camera **A** to shoot frames with the detail of $1/30^{th}$ of a second exposure time but the brightness of $1/15^{th}$ of a second exposure time. As noted earlier, darker objects require more computation than lighter objects to correct. Thus, as dark objects move in the video, the processor shares of tasks assigned to process different areas of the video will change. As a result, tasks will need to adjust their weights as quickly as an object can move

across the screen. Since ASTA continuously performs calculations based on previous frames, it performs best when a substantial amount of "useful" data is stored in the cache. As a result, migration/preemption costs in ASTA are fairly high. In addition, while strong real-time and fairness guarantees would be desirable in ASTA, they are not as important here as in Whisper, because tasks can function more independently in ASTA.

Experimental system set up. Unfortunately, at this point in time, it is not feasible to produce experiments involving a real implementation of either Whisper or ASTA, for several reasons. First, both the existing Whisper and ASTA systems are single-threaded (and non-adaptive) and consist of several thousands of 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 scheduling and resource-allocation trade-offs involved. The development of PD^2-OF, PAS, NP-PAS, CNG-EDF, and NP-CNG-EDF can be seen as an attempt to articulate these tradeoffs. Additionally, the focus of this paper is on scheduling methods that facilitate adaptation—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 and multimedia systems that are well beyond the scope of this paper. For these reasons, we have chosen to evaluate the schemes discussed in this paper via simulations of Whisper and ASTA. While just simulations, most of the parameters used here were obtained by implementing and timing the scheduling algorithms discussed in this paper and some of the signal-processing and video-enhancement code in Whisper and ASTA, respectively, on a real multiprocessor testbed. Thus, the behaviors in these simulations should fairly accurately reflect what one would see in a real Whisper or ASTA implementation.

For both Whisper and ASTA, the simulated platform was assumed to be a shared-memory multiprocessor, with four 2.7-GHz processors and a 1-ms quantum. All simulations were run 61 times. Both systems were simulated for 10 secs. (Note that longer simulations return similar results.) We implemented and timed each scheduling scheme considered in our simulations on an actual testbed that is the same as that assumed in our simulations, and found that all scheduling and reweighting computations could be completed within 5μs. We considered this value to be negligible in comparison to a 1-ms quantum and thus did not consider scheduling overheads in our simulations. For both Whisper and ASTA, we conducted two types of experiments: **(i)** all preemption and migration costs were the same and corresponded to a loss of cache affinity; and **(ii)** the preemption cost was set to some value and the migration cost was varied. If a task was preempted and then migrated, we assumed that it incurred the maximum of the two costs. We ignored the issue of bus contention, since in prior work, Holman and Anderson have shown that bus contention can be virtually eliminated in Pfair-scheduled systems by *staggering* quantum allocations on different processors [7]. Staggering would be trivial to apply in PAS and NP-PAS as well, since in PAS, processors run nearly independently of each other. Furthermore, since CNG-EDF and NP-CNG-EDF are event-based rather than quantum-based, jobs are unlikely to begin executing simultaneously. Based on measurements taken on our testbed system, we estimated Whisper's migration cost as 2μs–10μs, and ASTA's as 50μs–60μs. While we believe that these costs may be typical for a wide range of systems, in our experiments we varied the preemption/migration cost over a slightly larger range. For all experiments, the maximum execution cost of PAS and NP-PAS was 7ms and 5ms for CNG-EDF and NP-CNG-EDF. These values were determined by profiling each system beforehand to determine the "best" compromise of accuracy and performance.

While the ultimate metric for determining the efficacy of both systems would be user perception, this metric is not currently available, for reasons discussed earlier. Therefore, we compared each of the tested schemes by comparing against allocations in the IDEAL algorithm. In particular, we measured both the "average under-allocation" and "fairness factor" for each task set at the end of each simulation (*i.e.*, 10 secs.). The *average under-allocation* (UA) is the average amount each task is behind its IDEAL allocation (this value is defined to be nonnegative, *i.e.*, for a task that is not behind its IDEAL, this value is zero). The *fairness factor* (FF) of a task set is the largest deviance from the allocations in IDEAL between any two tasks (*e.g.*, if a system has three tasks, one that deviates from its IDEAL allocation by -10, another by 20, and the third by 50, then the FF is $50 - (-10) = 60$). The FF is a good indication of how fairly a scheme allocates processing capacity. A lower FF means the system is more fair. For applications like Whisper, where the output generated by multiple tasks is periodically combined, a low FF is important, since if any one task is "behind," then performance of the entire system is impacted; however, for applications like ASTA, where tasks are more independent, a high FF does not affect the system performance nearly as much. These metrics should provide us with a reasonable impression of how well the tested schemes will perform when Whisper and ASTA are fully re-implemented.

Whisper experiments. In our Whisper experiments, we simulated three speakers (one per object) revolving around pole in a 1m $\times$ 1m room with a microphone in each corner, as shown in Fig. 4. The pole creates potential occlusions. One task is required for each speaker-microphone pair, for a total of 12 tasks. In each simulation, the speakers were evenly distributed around the pole at an equal distance from the pole, and rotated around the pole at the same speed. The starting position for each speaker was set randomly. 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 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

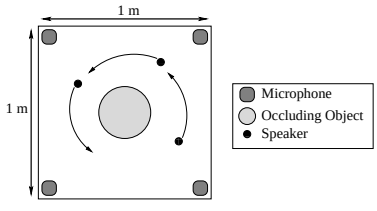

Figure 4: The simulated Whisper system.

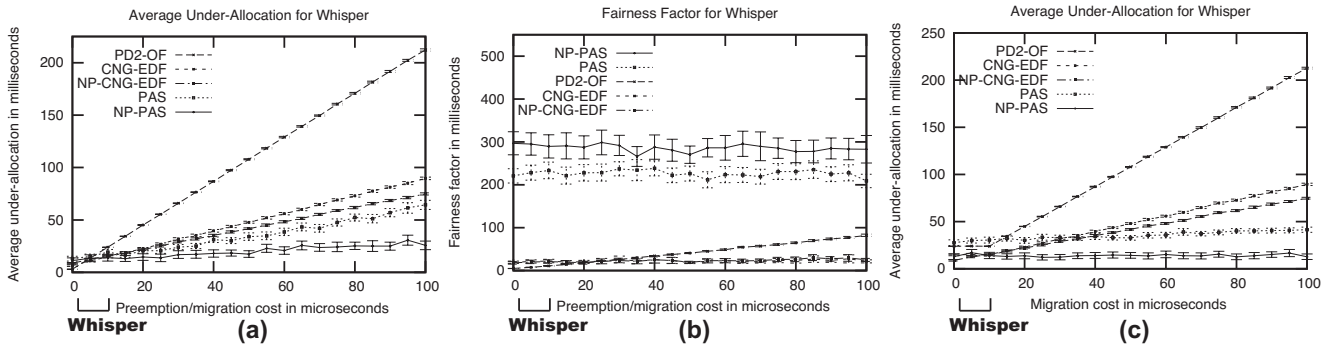

Figure 5: **(a)** The average under-allocation (UA) and **(b)** the fairness factor (FF) for Whisper as a function of preemption/migration cost, and **(c)** the average UA for Whisper as a function of migration cost (preemption cost is fixed at 10μs), as scheduled by each tested algorithm. The key in each graph is in the order that the schemes appear in that graph at 100μs. 98% confidence intervals are shown. Note that in (b), CNG-EDF and NP-CNG-EDF are indistinguishable from each-other.

accumulate-and-multiply operation) on our testbed system.

In the Whisper 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, the weight of each task changes only once for every 5cm of distance between its associated speaker and microphone. Sixth, all speakers and microphones are omnidirectional. Finally, all tasks have a minimum weight based on measurements from our testbed system and a maximum weight of 1.0. A task's current weight at any time lies between these two extremes and depends on the corresponding speaker's current position. Even with theses assumptions, frequent share adaptations are required.

We conducted Whisper experiments in which the tracked objects were sampled at a rate of 1,000 Hz, the distance of each object from the room's center was set at 50cm, the speed of each object was set at 5 m/sec. (this is within the speed of human motion), and the maximum execution cost, migration, and preemption cost were varied. However, due to page limitations, the graphs below are a representative sampling our collected data.

The first set of graphs in Fig. 5 show the result of the Whisper simulations conducted to compare PD^2-OF, PAS, NP-PAS, CNG-EDF, and NP-CNG-EDF. Insets (a) and (b) depict the average UA and FF, respectively, for each scheme, where the preemption cost is varied from 0 to 100μs and the migration cost equals the preemption cost. Inset (c) depicts the average UA for each scheme, where the preemption cost is set at 10μs (the maximum expected preemption cost for Whisper) and the migration cost is varied from 0 to 100μs. There are five things worth noting here. First, when the preemption/migration cost is varied over the range 2 to 10μs, the UA is about the same for all schemes (inset (a)); however, PD^2-OF has the best FF (inset (b)). Second, while CNG-EDF and NP-CNG-EDF do not have the best UA for the expected preemption/migration costs for Whisper, for higher preemption/migration costs, *i.e.*, preemption/migration costs larger than 10μs, CNG-EDF and NP-CNG-EDF both have a substantially better UA than PD^2-OF and better FF than either PAS or NP-PAS. Third, as the migration cost (but not preemption cost) of a task increases, the UA of PAS and NP-PAS increases slowly (inset (c)). However the performance of the other three schemes decays quickly. Fourth, the confidence intervals for the FF for CNG-EDF, NP-CNG-EDF, and PD^2-OF are smaller than for PAS and NP-PAS, since CNG-EDF, NP-CNG-EDF, and PD^2-OF have better accuracy. Fifth, in inset (c), PD^2-OF and CNG-EDF's UA do not appreciably increase until the migration cost exceeds 10μs. This is because, until the migration cost is 10μs, PD^2-OF and CNG-EDF incur the maximum of the migration or preemption cost, which is 10μs.

ASTA experiments. In our ASTA experiments, we simulated a 640×640-pixel video feed where a grey square that is 160×160 pixels moves around in a circle with a radius of 160 pixels on a white background. This is illustrated in Fig. 6. The grey square makes one complete rotation every ten seconds. The position of the grey square on the circle is random. Each frame is divided into sixteen 160×160-pixel regions; each of these regions is corrected by a different task. A task's weight is determined by whether the grey square covers its region. By analyzing ASTA's code, we determined that the grey square takes three times more processing time to correct than the white background. Hence, if the grey square completely covers a task's region, then its weight is three times larger than that of a task with an all-white region. The video is shot at a rate of 25 frames per second, and as a result, each frame has an exposure time of 40ms.

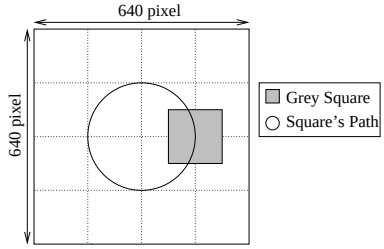

Figure 6: The simulated ASTA system.

The second set of graphs, in Fig. 7, show the result of the ASTA simulations conducted to compare the five scheduling algorithms. Insets (a) and (b) depict the average UA and FF, for each scheme, where the preemption cost is varied from 0 to 100μs and the migration cost equals the preemption cost. Inset (c) depicts the average UA for each scheme, where the preemption cost is set at 60μs (the maximum expected preemption cost for ASTA) and the migration cost is varied from 0 to 100μs. There are two things worth noting here. First, when the preemption/migration cost is varied over the range 50 to 60μs, NP-PAS and PAS have the smallest UA (inset (a)); however, CNG-EDF and NP-CNG-EDF both

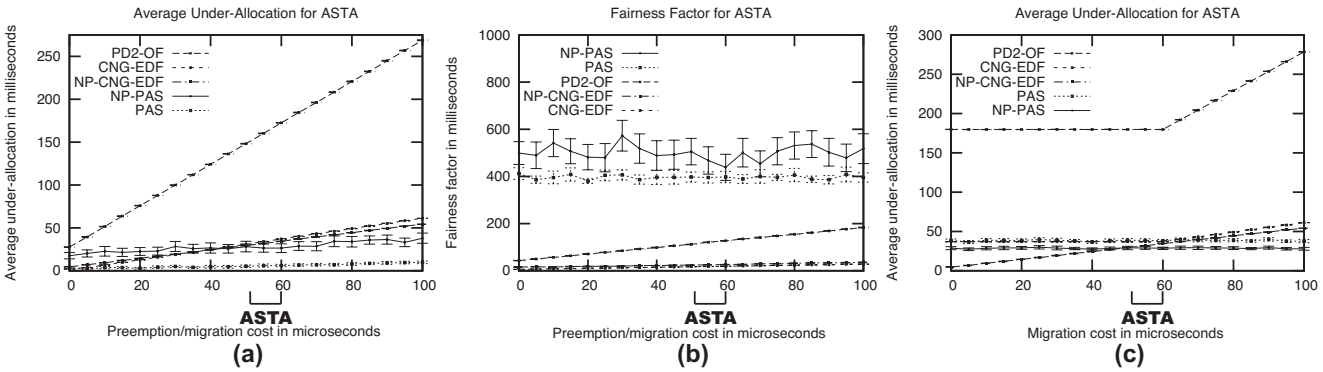

Figure 7: **(a)** The average under-allocation (UA) and **(b)** the fairness factor (FF) for ASTA as a function of preemption/migration cost, and **(c)** the average UA for ASTA as a function of migration cost (preemption cost is fixed at 60μs), as scheduled by each tested algorithm. The key in each graph is in the order that the schemes appear in that graph at 100μs. 98% confidence intervals are shown. Note that in (b), CNG-EDF and NP-CNG-EDF are indistinguishable from each-other.

Scheme	Provides Hard Real-Time Guarantees	Has Low Migration/ Preemption Costs	Provides Strong Fairness Guarantees
PD^2-OF	✓		✓
(NP-)PAS		✓	
(NP-)CNG-EDF		✓	✓

Table 2: Summary of algorithm performance.

have an UA that is competitive with both PAS and NP-PAS (inset (a)) and have a *substantially* smaller FF (inset (b)). Second, in inset (c) PD^2-OF and CNG-EDF's UA do not appreciably increase until the migration cost equals 60μs. This occurs for the same reason that PD^2-OF and CNG-EDF did not noticeably increase, until 10μs in Fig. 5(c).

6 Concluding Remarks

We have presented a two new multiprocessor reweighting schemes, CNG-EDF and NP-CNG-EDF, which reduce migration costs and preemptions at the expense of allowing deadline misses. We have also presented both analytical and experimental comparisons of these schemes with a more accurate but more migration-prone scheme, PD^2-OF, and two less accurate partitioning schemes that have lower tardiness, PAS and NP-PAS. These results suggest that when it is critical that every task make its deadline and migration/preemption costs are low (*i.e.*, systems like Whisper), then PD^2-OF is the best choice; when preemption/migration costs are high (*i.e.*, either Whisper or ASTA as implemented on a system where the processors are not as tightly integrated), average case performance is of the utmost importance, and fairness and timeless are less important, then either PAS or NP-PAS may be the best choice; and when preemption/migration costs are high and and a good mix of average-case performance and fairness factor is beneficial (*i.e.*, systems like ASTA), then either CNG-EDF or NP-CNG-EDF may be the best choice. Thus, *each algorithm is of value* and will be the best choice in certain application scenarios, as summarized in Table 2.

While our focus in this paper has been on 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 [9]). Both kinds of techniques warrant further study, especially in the domain of multiprocessor platforms.

References

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

[2] E. Bennett and L. McMillan. Video enhancement using per-pixel virtual exposures. *ACM Trans. on Graphics*, 24(3):845–852, 2005.

[3] 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 Comp. Sys. and Apps.*, pages 429–35, 2005.

[4] A. Block and J. Anderson. Accuracy verus migration overheads in multiprocessor reweighting. Algorithms. In submission.

[5] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah. A categorization of real-time multiprocessor scheduling problems and algorithms. In Joseph Y. Leung, editor, *Handbook on Scheduling Algorithms, Methods, and Models*, pages 30.1–30.19. Chapman Hall/CRC, Boca Raton, Florida, 2004.

[6] U. Devi and J. Anderson. Tardiness bounds under global EDF scheduling on a multiprocessor. In *Proc. of the 26th IEEE Real-time Sys. Symp.*, pages 330–41, 2005.

[7] P. Holman and J. Anderson. Implementing Pfairness on a symmetric multiprocessor. In *Proc. of the 10th IEEE Real-time and Embedded Technology and App. Symp.*, pages 544–553, 2004.

[8] J. Lopez, J. Diaz, and D. Garcia. Utilization bounds for EDF scheduling on real-time multiprocessor systems. *Real-Time Sys.*, 28(1):39–68, 2004.

[9] 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.*, pages 44–53, 1999.

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

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

7 Appendix

In this section, we provide the tardiness proofs for CNG-EDF and NP-CNG-EDF. Note that these proofs are only a slight modification of the tardiness proofs for EDF and NP-EDF originally presented by Devi and Anderson in [6]. For brevity, we use the function $\mathsf{tardiness}(\mathcal{S}, T_i^j)$ to denote the tardiness of the job T_i^j in the schedule $\mathcal{S}$. The tardiness proofs given below require one additional property concerning CNG-EDF and NP-CNG-EDF, which is stated below.

(E) For all jobs T_i^j in a task system scheduled via CNG-EDF or NP-CNG-EDF, $\mathsf{e}(T_i^j) = \mathsf{e}_{\max}(T_i)$, unless task T_i reweighted while T_i^{j-1} was active, and caused $\mathsf{ae}(T_i^{j-1})$ to be less than $\mathsf{e}(T_i^{j-1})$.

7.1 CNG-EDF

In this subsection, we show that tardiness of any job in any m-processor CNG-EDF schedule of any task system T for which $\mathsf{W}_{\text{sum}}(T, t) \leq m$, for all t, is at most $\kappa(m-1)$, where $\kappa(m-1)$ is as defined in Eq. (1) and $\mathsf{W}_{\text{sum}}(T, t) = \sum_{T_i \in T} \mathsf{swt}(T_i, t)$. As mentioned in Sec. 4, the following proof has three steps: **(i)** prove that a bounded LAG implies a bounded tardiness; **(ii)** bound the LAG of the system; and **(iii)** combine (i) and (ii) to produce a bounded tardiness for each job. Before continuing we introduce a few additional lemmas.

Since the total allocations to tasks in a m-processor CNG-EDF schedule $\mathcal{S}$ at any instant is at most m, if $[t_1, t_2)$ is a busy interval in $\mathcal{S}$, then the total allocation to tasks in T in $\mathcal{S}$ over the range $[t_1, t_2)$ is at most $m(t_2 - t_1)$. Hence by the definition of LAG, $\mathsf{LAG}(T, t_2) \leq \mathsf{LAG}(T, t_1)$. Thus, we have the following lemma.

Lemma 2. *For any m-processor CNG-EDF schedule $\mathcal{S}$ of the task system T, if $\mathsf{LAG}(T, t+\delta) > \mathsf{LAG}(T, t)$, where $\delta > 0$, and $\mathsf{W}_{\text{sum}}(T, t') \leq m$ for $t' \geq 0$, then $[t, t+\delta)$ is a non-busy interval in $\mathcal{S}$.*

Lemma 3. *If at some time $t \geq 0$, some job T_i^j in some task system T has completed in both $\mathcal{SW}$ and $\mathcal{S}$, then $\mathsf{A}(\mathcal{SW}, T_i^j, 0, t) = \mathsf{A}(\mathcal{S}, T_i^j, 0, t)$, where $\mathcal{SW}$ and $\mathcal{S}$ are, respectively, the m-processor SW and CNG-EDF schedules of T.*

Proof. By definition of both $\mathcal{SW}$ and $\mathcal{S}$, once a job T_i^j has received its actual execution cost it does not receive any additional allocations. Hence, if t is as defined in the statement of the lemma, then at t, $\mathsf{A}(\mathcal{SW}, T_i^j, 0, t) = \mathsf{A}(\mathcal{S}, T_i^j, 0, t) = \mathsf{ae}(T_i^j)$. □

The following corollary follows directly from Lemma 3.

Corollary 1. *If for some time $t \geq 0$, a task T_i is not pending in a CNG-EDF schedule of the task system T, then $\mathsf{lag}(T_i, t) \leq 0$.*

Now, we can show that if the LAG of T is bounded, then the tardiness of T is bounded.

Lemma 4. *Let T be a task system such that the deadline of every job is at most t_d and let the tardiness of every job $T_q^\ell \in T$ with a deadline less than t_d be at most $\mathcal{Z} + \mathsf{e}(T_q^\ell)$ in the m-processor CNG-EDF scheduled, $\mathcal{S}$, of T, where $\mathcal{Z} \geq 0$, $t_d = \mathsf{d}(T_i^j)$, and T_i^j some job in T. If $\mathsf{LAG}(T, t_d) \leq m \cdot \mathcal{Z} + \mathsf{e}(T_i^j)$, then $\mathsf{tardiness}(\mathcal{S}, T_i^j) \leq \mathcal{Z} + \mathsf{e}(T_i^j)$.*

Proof. To derive a contradiction, we assume that the job T_i^j has not completed in $\mathcal{S}$ by t_d. Throughout this proof, we denote the m-processor SW schedule of T as $\mathcal{SW}$.

By Lem. 3, if at some time t, all tasks in in T are entirely complete in both $\mathcal{SW}$ and $\mathcal{S}$, then $\mathsf{LAG}(T, t) = 0$. By the definition of $\mathcal{SW}$, at time t_d, all tasks in T are entirely complete in $\mathcal{SW}$. Therefore, at t_d, the amount of work remaining to be completed by all tasks in T in the schedule $\mathcal{S}$ equals $\mathsf{A}(\mathcal{SW}, T, 0, t_d) - \mathsf{A}(\mathcal{S}, T, 0, t_d) = \mathsf{LAG}(T, t_d)$.

Because there are no new jobs at or after t_d, there can be no preemptions, at or after t_d. (Note that this includes new jobs issued via the rules P and N.) Let $\delta = \mathsf{A}(\mathcal{S}, T_i^j, 0, t_d)$, and let $y = \mathcal{Z} + \delta/m$. We consider two cases depending on whether $[t_d, t_d + y)$ is busy in $\mathcal{S}$ or not.

Case 1: $[t_d, t_d + y)$ is busy. In this case, the total amount allocated to all tasks in T in $\mathcal{S}$ over the range $[t_d, t_d + y)$ is exactly $my = m\mathcal{Z} + \delta$. Since the amount of work that remains to be completed in $\mathcal{S}$ for all tasks in T at t_d is $\mathsf{LAG}(T, t_d)$, the amount of work remaining in $\mathcal{S}$ for all tasks pending at $t_d + y$ is at most $\mathsf{e}(T_i^j) - \delta$. Thus the latest time that T_i^j resumes execution in $\mathcal{S}$ after t_d is $t_d + y$, and because there are no preemptions at or after t_d, T_i^j is complete in $\mathcal{S}$ at or before $t_d + y + \mathsf{e}(T_i^j) - \delta \leq t_d + \mathcal{Z} + \mathsf{e}(T_i^j)$. Hence $\mathsf{tardiness}(\mathcal{S}, T_i^j) \leq t_d + \mathcal{Z} + \mathsf{e}(T_i^j) - \mathsf{d}(T_i^j) = \mathcal{Z} + \mathsf{e}(T_i^j)$. This case is illustrated in Fig. 8(a).

Case 2: $[t_d, t_d + y)$ is non-busy. Let t' denote the first (earliest) non-busy instant in the range $[t_d, t_d + y)$. Because the release time of any job in T is before t_d, any pending job T_a^b is ready at t' if T_a^b's predecessor jobs have completed in $\mathcal{S}$ by t'. Since at least one processor is idle at t', we have the following property.

(J) At most $m-1$ tasks have pending jobs in $\mathcal{S}$ at or after t'.

Since at least one processor is idle at t', if T_i^j is not complete by $t_d + y$ in $\mathcal{S}$, then some job of T_i is executing at t'. Because there are no preemptions after t_d and $t' < t_d + y$, if T_i^j is executing at t', then T_i^j completes in $\mathcal{S}$ before $t_d + y + \mathsf{e}(T_i^j) - \delta \leq t_d + \mathcal{Z} + \mathsf{e}(T_i^j)$. Thus, the tardiness of T_i^j is less than $\mathcal{Z} + \mathsf{e}(T_i^j)$, which satisfies the lemma.

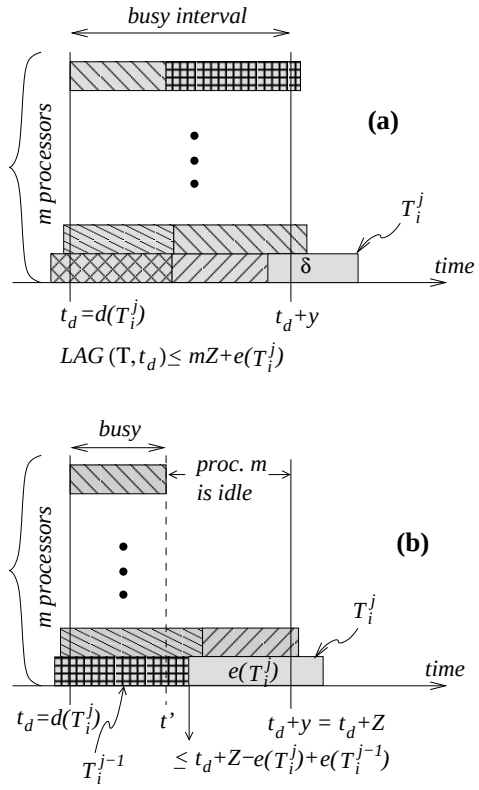

Figure 8: Lemma 4. **(a)** $[t_d, t_d + y)$ is busy. $T_{i,j}$ commences execution at or before $t_d + y$. **(b)** $[t_d, t_d + y)$ is not busy.

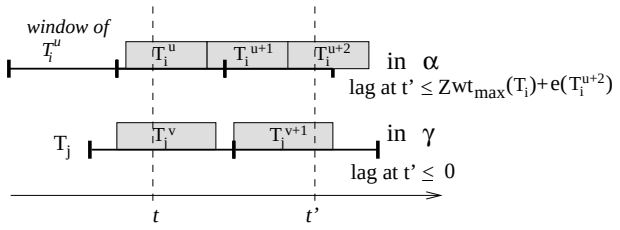

Figure 9: Lemma 5. Partitioning of tasks in T. Jobs of a sample task in each subset are shown. In this figure, all jobs except those of T_i, which is in α, complete executing by their deadlines.

The remaining possibility is that $j > 1$, and that a predecessor job (T_i^{j-1}) to T_i^j is executing at t' in $\mathcal{S}$. In this case, T_i^j could not have executed in $\mathcal{S}$ before t_d, and hence both $\delta = 0$ and $y = \mathcal{Z}$ hold. If the job T_i^{j-1} is not complete in $\mathcal{S}$ by $\mathsf{d}(T_i^{j-1})$, then by Lem. 1, $\mathsf{r}(T_i^j) \geq \mathsf{d}(T_i^{j-1})$. Thus, by the definition of deadline and the fact that a task's weight is at most 1, $t_d - \mathsf{e}(T_i^j) \geq \mathsf{d}(T_i^j) - \mathsf{e}(T_i^j) \geq \mathsf{r}(T_i^j) \geq \mathsf{d}(T_i^{j-1})$. Since $t_d - \mathsf{e}(T_i^j) \geq \mathsf{d}(T_i^{j-1})$ and since $\mathsf{tardiness}(T_i^{j-1}, \mathcal{S}) \leq \mathcal{Z} + \mathsf{e}(T_i^{j-1})$, the last time at which T_i^{j-1} could complete in $\mathcal{S}$ is $t_d - \mathsf{e}(T_i^j) + \mathcal{Z} + \mathsf{e}(T_i^{j-1})$. Since T_i^{j-1} is not complete in $\mathcal{S}$ by $\mathsf{d}(T_i^{j-1})$, by the definition of rules P and N, it cannot be the case that T_i^{j-1} was halted. Since T_i^{j-1} was not halted, $\mathsf{ae}(T_i^{j-1}) = \mathsf{e}(T_i^{j-1})$. Hence, by (E), $\mathsf{e}(T_i^j) = \mathsf{e}_{\mathsf{max}}(T_i)$. Hence, the last time at which T_i^{j-1} could complete is $t_d + \mathcal{Z}$. Because $t' < t_d + y = t_d + \mathcal{Z}$, by (J), T_i^j commences execution in $\mathcal{S}$ at or before $t_d + \mathcal{Z}$, and hence T_i^j is complete in $\mathcal{S}$ by $t_d + \mathcal{Z} + \mathsf{e}(T_i^j)$. This is illustrated in Fig. 8(b). □

Now, we can bound the LAG of T.

Lemma 5. *Let T be a task system such that the deadline of every job is at most t_d and let the tardiness of every job $T_q^\ell \in T$ with a deadline less than t_d be at most $\mathcal{Z} + \mathsf{e}(T_q^\ell)$ in the m-processor* CNG-EDF *scheduled, $\mathcal{S}$, of T, where $\mathcal{Z} \geq 0$, $\mathsf{W}_{\mathrm{sum}}(T, s) \leq m$ for all $s \geq 0$, and t_d is the deadline for some job in T. For any maximally non-busy time interval $[t, t')$ in $\mathcal{S}$, where $0 \leq t < t' \leq t_d$, the number of jobs in T that are executing at t' in $\mathcal{S}$, k is at most $m - 1$ and $\mathsf{LAG}(T, t') \leq \mathcal{Z} \cdot \sum_{T_i \in \mathcal{X}_{\max}(T,k)} \mathsf{wt}_{\mathsf{max}}(T_i) + \sum_{T_i \in \mathcal{E}_{\max}(T,k)} \mathsf{e}_{\mathsf{max}}(T_i)$.*

Proof. Let T and $\mathcal{S}$ be as defined in the statement of the lemma, and let t and t' be arbitrary values such that $0 \leq t < t' \leq t_d$ and $[t, t')$ is maximally non-busy for $\mathcal{S}$, and k be the number of jobs in T that are executing in $\mathcal{S}$ at the end of the interval. Let $\mathcal{SW}$ denote the m-processor SW schedule of T.

By the definition of busy, the fact that t'_- is non-busy, and by Eq. (4), $\mathsf{LAG}(T, t') = \sum_{T_i \in T} \mathsf{lag}(T_i^j, t')$ Therefore, an upper bound on the LAG of T at t' can be determined by establishing an upper bound on the lag at t' of each task in T . For this, we partition tasks in T into two subsets, α and γ, as defined below, and then determine an upper bound on the lag of a task in both subset. This partitioning is illustrated in Fig. 9

α = subset of all tasks in T executing in $\mathcal{S}$ throughout $[t, t')$.

γ = subset of all tasks in T not executing in $\mathcal{S}$ for some part of $[t, t')$.

Upper bound on the lag of a task in α. Let T_i be a task in α and let T_i^j be its job executing in $\mathcal{S}$ at t. Let δ denote the amount of time that T_i^j has executed in $\mathcal{S}$ before t. We first determine the lag of T_i at t by considering two cases depending on $\mathsf{d}(T_i^j)$. (We consider t' afterwards.)

Case 1: $\mathsf{d}(T_i^j) < t$**.** Because T_i^j is executing at t in $\mathcal{S}$ and any newly arriving job, which would have a deadline greater than $t > \mathsf{d}(T_i^j)$, cannot preempt T_i^j, the time at which T_i^j completes in $\mathcal{S}$ is $t + \mathsf{e}(T_i^j) - \delta$. ($T_i^j$ could complete earlier if its actual execution time is less than $\mathsf{e}(T_i^j)$.) By definition of T, $\mathsf{tardiness}(\mathcal{S}, T_i^j) \leq \mathcal{Z} + \mathsf{e}(T_i^j)$. Therefore, $\mathsf{d}(T_i^j) \geq t + \mathsf{e}(T_i^j) - \delta - (\mathcal{Z} + \mathsf{e}(T_i^j))$ holds.

By the definition of $\mathcal{SW}$ and Lem. 1, T_i^j is complete in $\mathcal{SW}$ by $\mathsf{d}(T_i^j)$, and any job of T_i that succeeds T_i^j is allocated (in $\mathcal{SW}$) at most $\mathsf{wt}_{\mathsf{max}}(T_i)$ at every instant in the range $[\mathsf{d}(T_i^j), t)$ in which T_i is active. (Since T_i^j is not complete by $t \geq \mathsf{d}(T_i^j)$, by Lem. 1, $\mathsf{r}(T_i^{j+1}) \geq \mathsf{d}(T_i^j)$.) Thus, the under-allocation to T_i in $\mathcal{S}$ over the range $[0, t)$ equals the under-allocation (relative to $\mathcal{SW}$) to T_i^j in $\mathcal{S}$, which is $\mathsf{e}(T_i^j) - \delta$, and the allocation to later jobs in T_i over the range $[\mathsf{d}(T_i^j), t)$ in $\mathcal{SW}$. Hence, $\mathsf{lag}(T_i, t)$ is at most $\mathsf{e}(T_i^j) - \delta + (t - \mathsf{d}(T_i^j)) \cdot \mathsf{wt}_{\mathsf{max}}(T_i) \leq \mathsf{e}(T_i^j) - \delta + (\mathcal{Z} + \delta) \cdot \mathsf{wt}_{\mathsf{max}}(T_i)$.

Case 2: $\mathsf{d}(T_i^j) \geq t$**.** In this case, the amount of work done by T_i^j in $\mathcal{SW}$ up to time t is at most $\mathsf{e}(T_i^j) - \int_t^{\mathsf{d}(T_i^j)} \mathsf{swt}(T_i, u)du$. Because all prior jobs of T_i are complete by t in both $\mathcal{S}$ and $\mathcal{SW}$, and T_i^j has executed for δ time units before t in $\mathcal{S}$, $\mathsf{lag}(T_i, t) \leq \mathsf{e}(T_i^j) - \int_t^{\mathsf{d}(T_i^j)} \mathsf{swt}(T_i, u)du \leq \mathsf{e}(T_i^j) - \delta \leq \mathsf{e}(T_i^j) - \delta + (\mathcal{Z} + \delta) \cdot \mathsf{wt}_{\mathsf{max}}(T_i)$

Thus, for both cases, we have

$$\mathsf{lag}(T_i, t) \leq \mathsf{e}(T_i^j) - \delta + (\mathcal{Z} + \delta) \cdot \mathsf{wt}_{\mathsf{max}}(T_i) \tag{6}$$

Next, in order to determine the lag of T_i, at t', we calculate the allocations to T_i over the range $[t, t')$ in both the $\mathcal{S}$ and $\mathcal{SW}$ schedules. In $\mathcal{SW}$, T_i is allocated a share of at most $\mathsf{swt}(T_i, u)$ at every instant $u \in [t, t')$, for a total allocation of at most $\int_t^{t'} \mathsf{swt}(T_i, u)du$. Because T_i executes throughout $[t, t')$ in $\mathcal{S}$, its total allocations in $\mathcal{S}$ over $[t, t')$ is $t' - t$. Hence, $\mathsf{A}(\mathcal{SW}, T_i, t, t') - \mathsf{A}(\mathcal{S}, T_i, t, t') \leq \int_t^{t'} \mathsf{swt}(T_i, u) - 1du$, and so, by the definition of lag and Eq. (6),

$$\begin{aligned} \mathsf{lag}(T_i, t') &\leq \mathsf{e}(T_i^j) - \delta + (\mathcal{Z} + \delta) \cdot \mathsf{wt}_{\mathsf{max}}(T_i) \\ &\quad + \int_t^{t'} (\mathsf{swt}(T_i, u) - 1)\, du \\ &\leq \mathsf{e}(T_i^j) + \mathcal{Z} \cdot \mathsf{wt}_{\mathsf{max}}(T_i) \end{aligned} \tag{7}$$

since $t > t'$, $\mathsf{wt}_{\mathsf{max}}(T_i) \leq 1$, and $\delta \geq 0$.

Upper bound on the lag of a task in γ. Let T_i be a task in γ and let t'' be the last time in the range $[t, t')$ such that T_i was not executing in $\mathcal{S}$ (note that t'' may be t'_-). Since CNG-EDF is work conserving and there is an idle processor at t'', T_i is not pending at t''. Hence by Cor. 1, $\mathsf{lag}(T_i, t'') \leq 0$. (Note that if $t'' = t'_-$, then this case is now complete.) By definition of t'', T_i is executing in $\mathcal{S}$ continuously over the range $[t'', t')$. Since over this range $\mathcal{SW}$ allocates T_i *at most* $t' - t''$ and $\mathcal{S}$ allocates T_i, $t' - t''$, $\mathsf{lag}(T_i, t'') \geq \mathsf{lag}(T_i, t')$. Therefore, $\mathsf{lag}(T_i, t') \leq 0$.

By Eq. (4), $\mathsf{LAG}(T, t')$ is given by the sum of the lags of tasks in subsets α and γ. As shown above, only tasks in α may have a positive lag, and thus $\mathsf{LAG}(T, t') = \sum_{T_i \in \alpha \cup \gamma} \mathsf{lag}(T_i, t') \leq \sum_{T_i \in \alpha} \mathsf{lag}(T_i, t')$, and by Eq. (9), $\mathsf{LAG}(T, t') \leq \sum_{T_i \in \alpha} (\mathsf{e}_{\mathsf{max}}(T_i) + \mathcal{Z} \cdot \mathsf{wt}_{\mathsf{max}}(T_i))$. Since $[t, t')$ is maximal non-busy and k tasks are executing in $\mathcal{S}$ at the end of the interval, $k \leq m - 1$ holds. Hence, since a task in α executes in $\mathcal{S}$ throughout $[t, t')$, there could be at most k tasks in α. Therefore $\mathsf{LAG}(T, t') \leq \mathcal{Z} \cdot \sum_{T_i \in \mathcal{X}_{\max}(T,k)} \mathsf{wt}_{\mathsf{max}}(T_i) + \sum_{T_i \in \mathcal{E}_{\max}(T,k)} \mathsf{e}_{\mathsf{max}}(T_i)$. □

Finally, to determine a tardiness bound for CNG-EDF, we are left with determining as small a $\mathcal{Z}$ as possible such that the upper bound given by Lem. 5 is at most the lower bound required in Lem. 4.

Theorem 2. *The tardiness for every job $\hat{T}_i^j$ of any task system $\hat{T}$, where $\mathsf{W}_{\mathrm{sum}}(\hat{T}, t) \leq m$, for any time t is at most $\kappa(m-1)$ in any* CNG-EDF *schedule for $\hat{T}$ on m processors.*

Proof. To derive a contradiction, assume that there exists a task system $\hat{T}$ such that $\mathsf{W}_{\mathrm{sum}}(\hat{T}, t) \leq m$ for all t and there exists some job $\hat{T}_i^j$ in $\hat{T}$ such that $\hat{T}_i^j$ has a tardiness greater than $\kappa(m-1)$ in some m-processor CNG-EDF schedule, $\hat{\mathcal{S}}$, of $\hat{T}$. Let T denote the task system obtained from $\hat{T}$ by removing all jobs with deadlines greater than $\hat{T}_i^j$ and let $\mathcal{S}$ be the m-processor CNG-EDF schedule of T. Assuming that CNG-EDF resolves ties among jobs consistently, every job T_k^ℓ is scheduled at the same time in $\mathcal{S}$ as its corresponding job $\hat{T}_k^\ell$ in $\hat{\mathcal{S}}$. Hence the tardiness of every job in T is the same in both schedule.

Let $t_d = \mathsf{d}(T_i^j)$, Since T_i^j misses its deadline, $\mathsf{lag}(T_i^j, t_d) > 0$. Since no job in T has a deadline greater than t_d, no task in T no task has negative lag at t_d. Thus, by Eq. (4), $\mathsf{LAG}(T, t_d) \geq 0$. Since $\mathsf{LAG}(T, 0) = 0$, by Lem. 2, there exists a non-busy interval in $[0, t_d)$ for $\mathcal{S}$. Let t' be the end of the latest non-busy instant before t_d. By Lem. 2,

$$\mathsf{LAG}(T, t_d) \leq \mathsf{LAG}(T, t'). \tag{8}$$

By the definition of T, the tardiness of any job T_ℓ^q with deadline less than t_d is at most $\kappa(m-1) = \mathcal{Z} + \mathsf{e}(T_\ell^q)$, for all $1 \leq \ell \leq n$, where n is the number of tasks in T and

$$\mathcal{Z} = \frac{\sum_{T_z \in \mathcal{E}_{\max}(T, m-1)} \mathsf{e}_{\max}(T_z)}{m - \sum_{T_z \in \mathcal{X}_{\max}(T, m-1)} \mathsf{wt}_{\max}(T_z)}$$

Hence, by Eq. (8) and Lem. 5,

$$\begin{aligned} \mathsf{LAG}(T, t_d) &\leq \mathsf{LAG}(T, t') \\ &\leq \mathcal{Z} \cdot \left(\textstyle\sum_{T_z \in \mathcal{X}_{\max}(T, m-1)} \mathsf{wt}_{\max}(T_z) \right) \\ &+ \left(\textstyle\sum_{T_z \in \mathcal{E}_{\max}(T, m-1)} \mathsf{e}_{\max}(T_z) \right). \end{aligned} \tag{9}$$

By definition of T and T_i^j, the tardiness of T_i^j is greater than $\mathcal{Z} + \mathsf{e}(T_i^j)$. Hence by Lem. 4, $\mathsf{LAG}(T, t_d) \geq m \cdot \mathcal{Z} + \mathsf{e}(T_i^j)$, which by Eq. (9) implies that

$$\begin{aligned} m \cdot \mathcal{Z} + \mathsf{e}(T_i^j) \quad < \quad & \mathcal{Z} \cdot \left(\sum_{T_i \in \mathcal{X}_{\max}(T, m-1)} \mathsf{wt}_{\max}(T_i) \right) \\ & + \left(\sum_{T_i \in \mathcal{E}_{\max}(T, m-1)} \mathsf{e}_{\max}(T_i) \right), \end{aligned}$$

i.e., $\mathcal{Z} < \frac{\sum_{T_z \in \mathcal{E}_{\max}(T, m-1)} \mathsf{e}_{\max}(T_z)}{m - \sum_{T_z \in \mathcal{X}_{\max}(T, m-1)} \mathsf{wt}_{\max}(T_z)}$. This contradicts the definition of $\mathcal{Z}$. Hence T does not exist. □

7.2 NP-CNG-EDF

In this section we establish the tardiness bounds for NP-CNG-EDF. The approach for deriving a tardiness bound for NP-CNG-EDF differs from that used for CNG-EDF in that we must also consider the length of time a task is blocked.

Before continuing it is useful to introduce some additional notation and definitions. First, for any set of jobs Ψ in the system T, we define the LAG of Ψ as

$$\mathsf{LAG}(\Psi, t) = \sum_{T_i^j \in \Psi} \mathsf{lag}(T_i^j, t).$$

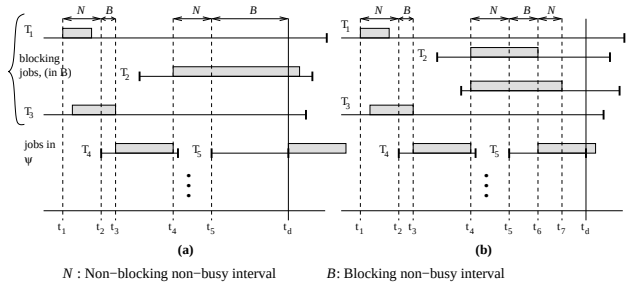

Figure 10: Illustration of the reasoning required for NP-CNG-EDF .

Note that $\mathsf{LAG}(\Psi, t)$ can also be defined as the sum of the lag of all tasks that have a job in Ψ at time t that are active or pending at time t_-.

Definition 4 (Blocking interval). For a task system T the range $[t_1, t_2)$ is the *blocking interval for* Ψ in the m-processor NP-CNG-EDF schedule of T if at least one job in Ψ is blocked in $[t_1, t_2)$. Moreover, $[t_1, t_2)$ is considered to be *maximally blocking*, if every non-empty subinterval of $[t_1, t_2)$ is a blocking interval, and either $t_1 = 0$ or t_{1-} is non-blocking.

Definition 5 (Pending blocking jobs ($\mathcal{B}$) and work (B)). For any job set Ψ in the task system T, we denote the set of all jobs that are in T and not in Ψ, block one or more jobs in Ψ at the time t_-, and may execute in the m-processor NP-CNG-EDF schedule $\mathcal{S}$ at t as $\mathcal{B}(T, \Psi, t, \mathcal{S})$. Furthermore, we denote the total amount of time that the jobs in $\mathcal{B}(T, \Psi, t, \mathcal{S})$ execute beyond t in $\mathcal{S}$, *i.e.*, the total amount of work pending for those jobs at t as $\mathsf{B}(T, \Psi, t, \mathcal{S})$. (For brevity, we denote $\mathsf{B}(T, \Psi, t, \mathcal{S})$ as $\mathsf{B}(\Psi, t)$ when T and $\mathcal{S}$ are obvious.) The set of all jobs of tasks in T, not in the job set Ψ that can block some job in Ψ in a m-processor NP-CNG-EDF schedule $\mathcal{S}$ of T at some time is denoted $\mathcal{B}(T, \Psi, \mathcal{S})$, or simply $\mathcal{B}$, when the parameters are obvious.

Non-busy interval categories. With respect to a job set Ψ, an interval $[t_1, t_2)$ is said to be *busy* only if every processor is executing some job of Ψ throughout the interval. With this definition, it is easy to see that the LAG of Ψ can increase only across a non-busy interval, and so Lem. 2 applies to Ψ in a schedule for T. Also note that by this definition, a blocking interval for Ψ is also a non-busy interval for Ψ. However, not every instant in which a job in $\mathcal{B}$ is executing need be a blocking instant. Thus, a non-busy interval in a NP-CNG-EDF schedule for $\Psi \cup \mathcal{B}$ can be classified as either **(i)** a *blocking, non-busy interval* or **(ii)** a *non-blocking, non-busy interval*. (Note that in this section, every non-busy interval or a blocking interval is taken to be maximal, unless otherwise stated. We refrain from explicitly saying so for conciseness.)

By Lem. 2, we know that the LAG of a job set Ψ can increase only across a non-busy interval (not necessarily maximal). Thus, if Ψ is the set of jobs with a deadline at most t_d, in order to determine an upper bound of LAG at t_d it is sufficient to determine an upper bound at the end of the latest non-busy interval. As discussed above, a non-busy interval for Ψ in an NP-CNG-EDF schedule is either blocking or non-blocking. Therefore, to determine an upper bound on the LAG of Ψ at t_d, we determine an upper-bound on LAG at the end of the last blocking, non-busy interval, or the last non-blocking, non-busy interval, whichever is later. For example, in Fig. 10(a), $[t_4, t_d)$ is the latest non-busy interval. Within this interval, subinterval $[t_4, t_5)$ is non-blocking, while $[t_5, t_d)$ is blocking. Therefore, we will determine an upper bound for LAG at t_d by considering $[t_5, t_d)$. Similarly, in Fig. 10(b), an upper bound on LAG will be determined at t_7, by considering the interval $[t_6, t_7)$. Next, because every job of Ψ that is not executing in a non-blocking, non-busy interval is either not pending or pending but not ready (because a prior job is executing) the procedure for determining LAG at the end of of such an interval is identical to that used for CNG-EDF in Lem. 2. (In Lem. 2,

we showed that the LAG of a task that does not execute at the end of a non-busy interval is at most zero.) However, since throughout a non-busy interval (relative to Ψ) in which there is blocking, there is at least one task that has a ready job in Ψ is not executing, the lag of a task that is not executing at the end of such a non-busy interval cannot be taken to be zero. For example, in Fig. 10(a), the task T_5 is not executing in $[t_5, t_d)$ and the lag of its task at t_d is positive. Therefore, the procedure for determining LAG is slightly different in this case.

A blocking job with a deadline later than t_d, that is executing but is incomplete at t_d, will continue to execute beyond t_d, which will delay the executing of pending jobs in Ψ. Hence, in order to determine a tardiness bound for a job in Ψ, apart from an upper bound on the amount of work pending for jobs in Ψ at t_d, *i.e.*, $\mathsf{LAG}(\Psi, t_d)$, we also need to determine an upper bound on the total amount of work pending for jobs that are blocking those of Ψ at t_d, *i.e.*, $\mathsf{B}(t_d, \mathcal{S})$. In Fig. 10(a), if we assume that T_2 is the only pending blocking task at t_d, we will also need an estimate of the amount of time that T_2 executes after t_d. In Fig. 10(b), the amount of pending work for jobs in $\mathcal{B}$ is positive at t_6, while it is zero at and after t_7. Note that unless the latest non-busy instant is t_d, the amount of blocking work that is pending at t_d will be zero.

As with CNG-EDF, we then determine a lower bound on the sum of the blocking work B and the LAG of Ψ at t_d that is necessary for the tardiness of a job with a deadline at most t_d to exceed a given value, and an upper bound of the maximum value for the same that is possible with a given task system. Finally, we use these to arrive at a tardiness bound. The lemma that follows parallels Lem. 4 and its proof is similar.

Lemma 6. *Let T be a task system such that the tardiness of every job $T_q^\ell \in T$ with a deadline less than t_d be at most $\mathcal{Z} + \mathsf{e}(T_q^\ell)$ in the m-processor* NP-CNG-EDF *schedule, $\mathcal{S}$, of T, where $\mathcal{Z} \geq 0$, $t_d = \mathsf{d}(T_i^j)$, and T_i^j is some job in T. Let Ψ denote the set of all jobs in T with a deadline at most t_d. If $\mathsf{LAG}(\Psi, t_d) + \mathsf{B}(\Psi, t_d) \leq m \cdot \mathcal{Z} + \mathsf{e}(T_i^j)$, then* $\mathsf{tardiness}(T_i^j, \mathcal{S}) \leq \mathcal{Z} + \mathsf{e}(T_i^j)$.

An upper bound of $\mathsf{LAG}(\Psi, t') + \mathsf{B}(\Psi, t')$, where t' is the end of maximally non-busy interval is given by the next lemma. Its proof is only slightly different from that of Lem. 5.

Lemma 7. *Let T, Ψ, $\mathcal{S}$, and m be as defined in Lemma 6. Let $[s, t')$, where $0 \leq s < t' \leq t_d$, be a maximally non-busy interval for Ψ in $[0, t_d)$ in $\mathcal{S}$, such that either $t' = t_d$ or t' is busy. Let the tardiness in $\mathcal{S}$ of every job $T_q^\ell \in T$ with a deadline less than t_d be at most $\mathcal{Z} + \mathsf{e}(T_q^\ell)$, where $\mathcal{Z} \geq 0$. Then $\mathsf{LAG}(\Psi, t') + \mathsf{B}(\Psi, t') \leq \mathcal{Z} \cdot \left(\sum_{T_i \in \mathcal{X}_{\max}(T, m-1)} \mathsf{wt}_{\max}(T_i)\right) + \sum_{T_i \in \mathcal{E}_{\max}(T, m)} \mathsf{e}_{\max}(T_i)$.*

Proof. Referring to the statement of the lemma, $[s, t')$ is a maximal non-busy interval for Ψ. Hence, every instant in the interval is either a blocking, non-busy instant, or a non-blocking, non-busy instant. We consider two cases depending on whether t'_- is blocking.

CASE A: t'_- is a non-blocking instant. This case is illustrated in Fig. 11(a). Let t be the earliest instant at or after s such that at every instant in $[t, t')$, either at least one processor is idle, or at least one job in $\mathcal{B}$ is executing, or both hold. However, the jobs in $\mathcal{B}$ that are executing in this interval do not block any job in Ψ. Therefore, in both cases, every job of Ψ that is not executing at some instant in $[t, t')$ is either inactive at that instant or is active, but has no pending jobs. Hence, for the purpose of determining the $\mathsf{LAG}(\Psi, t')$, the jobs in $\mathcal{B}$ that are executing in $[t, t')$ can be ignored and the interval in which they are executing can be taken to be idle intervals on the respective processors. Therefore, the $\mathsf{LAG}(\Psi, t')$, can be determined in the same manner as that used in the case preemptive

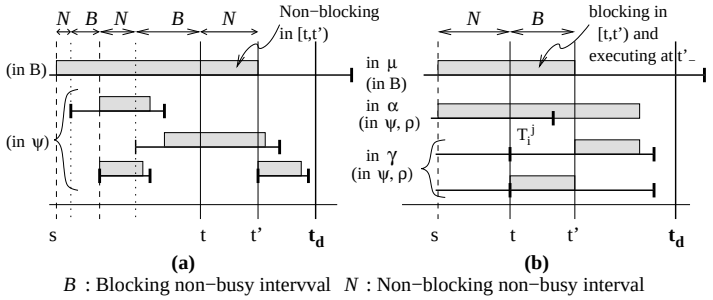

Figure 11: Lemma 7. **(a)** CASE A. **(b)** CASE B.

CNG-EDF in Lemma. 5 to be at most

$$\mathcal{Z} \cdot \left(\sum_{T_i \in \mathcal{X}_{\max}(T_\Psi, k)} \mathsf{wt}_{\max}(T_i) \right) + \sum_{T_i \in \mathcal{E}_{\max}(T_\Psi, k)} \mathsf{e}_{\max}(T_i), \tag{10}$$

where T_Ψ is the subset of all tasks in T whose jobs in Ψ are executing at the end of the interval $[t, t')$, and $k = |T_\Psi| \leq m - 1$.

We next determine a bound of $\mathsf{B}(\Psi, t')$. If $t' \leq t_d$, then by the statement of the lemma t' is busy. Therefore no job $\mathcal{B}$ that is executing at t'_- executes at t' or later. Hence, in this case $\mathsf{B}(\Psi, t') = 0$. The other case is that $t' = t_d$ holds. Note that each job T_i^j in $\mathcal{B}(T, \Psi, t_d, \mathcal{S})$ could execute for at most $\mathsf{e}(T_i^j)$ time units after t_d. Because k jobs are executing at the end of the interval $[t, t')$ are in Ψ, at most $m - k$ jobs of $\mathcal{B}$ are executing at t'_-. Therefore, when $t' = t_d$, $\mathsf{B}(\Psi, t_d) \leq \sum_{T_i \in \mathcal{E}_{\max}((T \setminus T_\Psi), m-k)} \mathsf{e}_{\max}(T_i)$. Hence, for either case, by (10) we have

$$\begin{aligned} \mathsf{LAG}(\Psi, t') + \mathsf{B}(\Psi, t') \quad \leq \quad & \mathcal{Z} \cdot \left(\textstyle\sum_{T_i \in \mathcal{X}_{\max}(T, m-1)} \mathsf{wt}_{\max}(T_i) \right) \\ & + \textstyle\sum_{T_i \in \mathcal{E}_{\max}(T, m)} \mathsf{e}_{\max}(T_i). \end{aligned}$$

CASE B: t'_- is a blocking instant. In this case, let t denote the earliest instant at or after s such that $[t, t')$ is a maximally-blocking interval. Since every job of Ψ has an earlier deadline than a job in $\mathcal{B}$, a job in Ψ cannot be blocked at time 0 due to a job in $\mathcal{B}$ commencing execution at time 0. Therefore $t > 0$ holds. Also, no job of Ψ (including jobs that are blocked at t) is blocked at t_-. Hence, it cannot be the case that a job in Ψ is blocked at t due to a job in $\mathcal{B}$ commencing executing at t. Rather, the blocking job should have commenced execution before t. Similarly, since every instant $[t, t')$ is a blocking instant, at which one or more ready jobs of Ψ are waiting, no job in $\mathcal{B}$ can commence execution anywhere in (t, t'). Therefore, we have the following.

(B) Every job in $\mathcal{B}$ that is executing at $t \leq \hat{t} < t'$, is executing throughout $[t_-, \hat{t}]$.

Let $\mathcal{J}$ denote the set of all jobs of $\mathcal{B}$ that are executing at t, and hence are blocking one or more jobs of Ψ. Let $b = |\mathcal{J}|$, and let μ denote the subset of all tasks in T whose jobs are in $\mathcal{J}$. By the nature of $[t, t')$, $b \geq 1$. Because, each task can have at most one job executing at any instant, we have

$$|\mathcal{J}| = |\mu| = b \geq 1. \tag{11}$$

By the definition of LAG for a set of jobs, the LAG of Ψ at t is given by the sum of the lags of all tasks in T with at least one job in Ψ that is either pending or active at t_-. Let ρ denote the set of all such tasks. (It is easy to see that no task in μ is in ρ.) Therefore,

$$\mathsf{LAG}(\Psi, t) \leq \sum_{T_i \in \rho} \mathsf{lag}(T_i, t), \tag{12}$$

Partitioning ρ. Our approach for determining an upper bound on the LAG of Ψ at t' is mostly similar to that used in Lemma 5. Because (12) holds, we first partition the tasks in ρ into the subsets α and γ, as defined below, and determine upper bounds on the lag at t of tasks in each subset, and the number of tasks in each subset. We use these to determine an upper-bound on the LAG of Ψ at t, from which, we then determine an upper bound on the LAG of Ψ at t'

$$\begin{aligned}\alpha &= \text{subset of all tasks in } \rho \text{ executing at } t_-\\ \gamma &= \text{subset of all tasks in } \rho \text{ not executing at } t_-\end{aligned}$$

Upper bound on lag at t of a task in α. Let T_i be a task in α and let T_i^j be its job executing at t_-. Let δ denote the amount of time that T_i^j has executed for before t in $\mathcal{S}$. We determine the lag of T_i at t by considering two cases depending on $\mathsf{d}(T_i^j)$.

Case 1: $\mathsf{d}(T_i^j) \leq t$. Because T_i^j cannot be preempted, the latest time that T_i^j completes executing can be $t = \mathsf{e}(T_i^j) - \delta$. (It could complete earlier if its actual executing time is lower than $\mathsf{e}(T_i^j)$). By the statement of the lemma, the tardiness of every job of T_i, with deadline less than t_d is at most $\mathcal{Z} + \mathsf{e}(T_i^j)$. Therefore, $\mathsf{d}(T_i^j) \geq t + (\mathsf{e}(T_i^j) - \delta) - (\mathcal{Z} + \mathsf{e}(T_i^j)) = t - \delta - \mathcal{Z}$ holds. By Lemma 1, T_i^j is complete in $\mathcal{SW}$ by $\mathsf{d}(T_i^j)$ and T_i is allocated a share of $\mathsf{swt}(T_i, q)$ in every instant $q \in [\mathsf{d}(T_i^j), t)$ in which it is active. Thus the under-allocation to T_i in $\mathcal{S}$ in $[0, t)$ is at most $\mathsf{e}(T_i^j) - \delta + (t - \mathsf{d}(T_i^j)) \cdot \mathsf{wt}_{\mathsf{max}}(T_i) \leq \mathsf{e}(T_i) - \delta + (\mathcal{Z} + \delta) \cdot \mathsf{wt}_{\mathsf{max}}(T_i)$. Hence $\mathsf{lag}(T_i, t) \leq \mathsf{e}(T_i) - \delta + (\mathcal{Z} + \delta) \cdot \mathsf{wt}_{\mathsf{max}}(T_i) \leq \mathsf{e}(T_i^j) + \mathcal{Z} \cdot \mathsf{wt}_{\mathsf{max}}(T_i)$.

Case 2: $\mathsf{d}(T_i^j) \geq t$. In this case, the amount of work done by the job T_i^j in $\mathcal{SW}$ up to time t is given by $\mathsf{e}(T_i^j) - \int_t^{\mathsf{d}(T_i^j)} \mathsf{wt}(T_i, u)du$. Because all prior jobs of T_i have complete executing by t in both $\mathcal{S}$ and $\mathcal{SW}$, and T_i^j has executed for δ time units before t in $\mathcal{S}$, $\mathsf{lag}(T_i, t) \leq \mathsf{e}(T_i^j) - \int_t^{\mathsf{d}(T_i^j)} \mathsf{swt}(T_i, u)du \leq \mathsf{e}(T_i^j) - \delta \leq \mathsf{e}(T_i^j) - \delta + (\mathcal{Z} + \delta) \cdot \mathsf{wt}_{\mathsf{max}}(T_i)$. Thus in both cases we have

$$\mathsf{lag}(T_i, t) \leq \mathsf{e}_{\mathsf{max}}(T_i) + \mathcal{Z} \cdot \mathsf{wt}_{\mathsf{max}}(T_i). \tag{13}$$

Upper Bound on the lag at t of a task in γ. Let T_i be a task in γ. Then, no job of T_i is executing at t_-. However, since T_i is in ρ, there is at least a job of T_i that is in Ψ that is either pending or active at t_-. We show that no job of T_i that is in Ψ is pending at t_-. Suppose that the job T_i^j is in Ψ and is pending at t_-. Then $\mathsf{d}(T_i^j) \leq t_d$ holds and because T_i is in γ, T_i^j is not executing at t_-. Since $[t, t')$ is maximally-blocking, at least on job of $\mathcal{B}$ is executing at t, which by (B), is executing at t_- as well. Because such a blocking job has its deadline after t_d and no job of T_i is executing at t_-, this implies that T_i^j is blocked at t_-, contradicting our assumption that $[t, t')$ is a maximally-blocked interval. For example, T_i^j could be as indicated in Fig. 11(b).

Thus, no job of T_i that is in Ψ is pending at t_-. Therefore, the total allocation to jobs of T_i in Ψ up to time t in $\mathcal{S}$ is at least that in $\mathcal{SW}$, and hence the lag of T_i at t is at most zero.

Because the lag of a task in γ is at most zero at t, $\sum_{T_i \in \rho} \mathsf{lag}(T_i, t) = \sum_{T_i \in \alpha} \mathsf{lag}(T_i, t) + \sum_{T_i \in \gamma} \mathsf{lag}(T_i, t) \leq \sum_{T_i \in \alpha} \mathsf{lag}(T_i, t)$. Hence by (13), $\sum_{T_i \in \alpha} \mathsf{lag}(T_i, t) \leq \sum_{T_i \in \alpha} (\mathsf{e}_{\mathsf{max}}(T_i) + \mathcal{Z} \cdot \mathsf{wt}_{\mathsf{max}}(T_i))$. Therefore, by (12), we have

$$\mathsf{LAG}(\Psi, t) \sum_{T_i \in \alpha} (\mathsf{e}_{\mathsf{max}}(T_i) + \mathcal{Z} \cdot \mathsf{wt}_{\mathsf{max}}(T_i)) \tag{14}$$

Since we need to determine an upper bound on the sum of $\mathsf{LAG}(\Psi, t')$ and $\mathsf{B}(\Psi, t')$, we also need to determine an upper bound on $\mathsf{B}(\Psi, t)$. By (B), no job of $\mathcal{B}$ that is not in $\mathcal{J}$ can execute anywhere in $[t, t')$. Hence, the amount of work pending of jobs in $\mathcal{B}$ (*i.e.*, the blocking work) at any time $u \in [t, t')$, $\mathsf{B}(\Psi, u)$, equals the amount of work pending at u for the jobs in $\mathcal{J}$. Let T_i be a task in μ. Then, the amount of work that can be pending for its job executing at t (which is in $\mathcal{J}$) can be at most the execution cost of job. Therefore we have $\mathsf{B}(\Psi, t) \leq \sum_{T_i \in \mu} \mathsf{e}_{\mathsf{max}}(T_i)$, and hence by (14), we have

$$\begin{aligned}&\mathsf{LAG}(\Psi, t) + \mathsf{B}(\Psi, t) \leq \textstyle\sum_{T_i \in \alpha} (\mathsf{e}_{\mathsf{max}}(T_i) + \mathcal{Z} \cdot \mathsf{wt}_{\mathsf{max}}(T_i)) + \\ &\textstyle\sum_{T_i \in \mu} \mathsf{e}_{\mathsf{max}}(T_i) = \sum_{T_i \in \alpha \cup \mu} \mathsf{e}_{\mathsf{max}}(T_i) + \sum_{T_i \in \alpha} \mathcal{Z} \cdot \mathsf{wt}_{\mathsf{max}}(T_i) \leq \\ &\mathcal{Z} \cdot \textstyle\sum_{T_i \in \mathcal{X}_{\max}(T, m-1)} \mathsf{wt}_{\mathsf{max}}(T_i) + \sum_{T_i \in \mathcal{E}_{\max}(T, m)} \mathsf{e}_{\mathsf{max}}(T_i),\end{aligned} \tag{15}$$

where the last inequality follows from (11) ($|\mu| = b \geq 1$) and ($|\alpha| = m - b$). $|\alpha| = m - b$ holds because every task in μ or α is executing at t_-.

Finally, we are are left with determining an upper bound on the sum of the LAG and B at t'. Let $X \leq \mathsf{B}(\Psi, t)$ denote the total amount of time that jobs in $\mathcal{J}$ execute on all m processors in $[t, t')$ (For example, if there are two jobs in $\mathcal{J}$, with one job executing for the entire interval and the second executing for the first half of the interval, then $X = 3(t' - t)/2$.) Because $[t, t')$ is maximally blocking, no processor is idle in $[t, t')$. Hence, the total time allocated to jobs in Ψ in $[t, t')$, $\mathsf{A}(\mathcal{S}, \Psi, t, t')$ is equal to $m \cdot (t' - t) - X$. In $\mathcal{SW}$, jobs in Ψ could execute for at most $m \cdot (t' - t)$ time, *i.e.*, $\mathsf{A}(\mathcal{S}, \Psi, t, t') \leq m \cdot (t' - t)$. Therefore, $\mathsf{LAG}(\Psi, t') = \mathsf{LAG}(\Psi, t) + \mathsf{A}(\mathcal{SW}, \Psi, t, t') - \mathsf{A}(\mathcal{S}, \Psi, t, t') \leq \mathsf{LAG}(\Psi, t) + X$. However, since jobs in Ψ execute for a total time of X in $[t, t')$, the pending work for jobs in Ψ, and hence those in $\mathcal{B}$ at t', $\mathsf{B}(\Psi, t')$, is at most $\mathsf{B}(\Psi, t') - X$. Thus $\mathsf{LAG}(\Psi, t') + \mathsf{B}(\Psi, t') \leq \mathsf{LAG}(\Psi, t) + \mathsf{B}(\Psi, t)$, which by (15) is at most $\mathcal{Z} \cdot \sum_{T_i \in \mathcal{X}_{\max}(T, m-1)} \mathsf{wt}_{\mathsf{max}}(T_i) + \sum_{T_i \in \mathcal{E}_{\max}(T, m)} \mathsf{e}_{\mathsf{max}}(T_i)$. □

Lemmas. 6 and 7 can be used to establish the following.

Theorem 3. *The tardiness for every job $\hat{T}_i^j$ of any task system $\hat{T}$, where $\mathsf{W}_{\mathrm{sum}}(\hat{T}, t) \leq m$, for any time t is at most $\kappa(m)$ in any* NP-CNG-EDF *schedule for $\hat{T}$ on m processors.*