Tardiness Bounds under Global EDF Scheduling on a Multiprocessor *

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

Abstract

This paper considers the scheduling of soft real-time sporadic task systems under global EDF on an identical multiprocessor. Though Pfair scheduling is theoretically optimal for hard real-time task systems on multiprocessors, it can incur significant run-time overhead. Hence, other scheduling algorithms that are not optimal, including EDF, have continued to receive considerable attention. However, prior research on such algorithms has focussed mostly on hard real-time systems, where, to ensure that all deadlines are met, approximately 50% of the available processing capacity will have to be sacrificed in the worst case. This may be overkill for soft real-time systems that can tolerate deadline misses by bounded amounts (*i.e.*, bounded tardiness). In this paper, we derive tardiness bounds under preemptive and non-preemptive global EDF on multiprocessors when the total utilization of a task system is not restricted and may equal the number of processors. Our tardiness bounds depend on per-task utilizations and execution costs — the lower these values, the lower the tardiness bounds. As a final remark, we note that global EDF may be superior to partitioned EDF for multiprocessor-based soft real-time systems in that the latter does not offer any scope to improve system utilization even if bounded tardiness can be tolerated.

*Work supported by NSF grants CCR 0204312, CNS 0309825, and CNS 0408996. The first author was also supported by an IBM Ph.D. fellowship.

1 Introduction

A number of present-day and emerging applications require real-time and quality-of-service (QoS) guarantees and also have workloads that necessitate multiprocessor-based designs. Systems that track people and machines, virtual-reality systems, systems that host web-sites, and signal-processing systems such as synthetic-aperture imaging are some examples. Timing constraints in several of these applications are predominantly soft in that deadlines may be missed as long as the long-run fraction of the processing time allocated to each task in the application is in accordance with its utilization. A system design that can guarantee that deadline misses, if any, are bounded by constant amounts is sufficient to provide guarantees on long-term processor shares. Hence, scheduling methods that ensure bounded deadline misses and that can be applied when other methods cannot are of considerable value and interest.

Multiprocessor scheduling algorithms can be classified according to whether they use a *partitioning* or *global-scheduling* approach. Under partitioning, tasks are statically partitioned among processors, with tasks assigned to each processor scheduled using a separate instance of a uniprocessor scheduling algorithm. In contrast, under global scheduling, a task may execute on any processor and may migrate across processors. A single ready queue stores ready jobs, and the job with the highest priority is chosen for execution from this ready queue at each scheduling decision. The two approaches can be differentiated further based on the scheduling algorithm that is used. For instance, the *earliest-deadline-first* (EDF) or the *rate-monotonic* (RM) scheduling algorithm could be used as the per-processor scheduler under partitioning, or as the system-wide scheduler under global scheduling. Though Pfair scheduling [5], which falls under the umbrella of global scheduling, is theoretically optimal* for recurrent real-time task systems on multiprocessors, it can incur significant preemption, migration, and scheduling overheads due to its quantum-based scheduling. Hence, other scheduling algorithms that are not optimal have continued to receive considerable attention.

It is well known that EDF with job preemptions allowed is optimal for scheduling independent periodic or sporadic tasks on uniprocessors and that its utilization bound is 100% when relative deadlines are *implicit*, *i.e.*, equal periods [10]. However, the worst-case utilization bound of EDF for implicit-deadline systems is approximately 50% on multiprocessors, under both partitioning and global scheduling [7]. Moreover, partitioning schemes suffer from the drawback of offering no scope for improving system utilization even if bounded deadline misses can be tolerated. This is because, if a task set cannot be partitioned without over-utilizing some processor, then deadline misses and tardiness for tasks on that processor will increase with time. On the other hand, as we will see, the outlook is more promising if inter-processor migrations are permissible.

In 1978, Dhall and Liu showed that there exist task sets with total utilization arbitrarily close to 1.0 that cannot be correctly scheduled on m processors for any $m \geq 2$ under global EDF or RM scheduling [8]. Perhaps

*A real-time scheduling algorithm is said to be *optimal* iff it can correctly schedule (*i.e.*, without deadline misses) every task system for which a correct schedule exists.

due to this negative result, also referred to as the "Dhall effect," the real-time research community had largely ignored global scheduling. However, in the recent past, several researchers have noted that the Dhall effect is due to the presence of tasks with high and low utilizations and have shown that it can be overcome by restricting per-task utilizations. First, in [14], Srinivasan and Baruah showed that on m processors, EDF can correctly schedule any independent periodic task system (with implicit deadlines) in which the maximum utilization of any task, $u_{\max}$, is at most $m/(2m-1)$, as long as the total utilization of all tasks, U_{sum}, is at most $m^2/(2m-1)$. They also proposed a variant of EDF that prioritizes tasks with utilization exceeding $m/(2m-1)$ over the rest, and showed that the modified algorithm can correctly schedule any task system for which U_{sum} does not exceed $m^2/(2m-1)$. Later, Goossens *et al.* [9] showed that EDF can correctly schedule any task system with total utilization not exceeding $m-(m-1)u_{\max}$ on m processors, and improving upon the result in [14], Baruah [4] proposed an EDF variant that can schedule any task system if U_{sum} is at most $(m+1)/2$.

The above-mentioned results were derived using a result in [11] that relates the speed of the processors running an optimal algorithm to the speed required for the processors running EDF to avoid deadline misses. This was noted by Baker in [2], who then extended the processor-time demand argument commonly used in uniprocessor analysis to multiprocessors, and used it to derive a sufficient schedulability condition under EDF for independent periodic or sporadic task systems when relative deadlines are at most periods (*i.e.*, *constrained* deadline systems). Baker's condition reduces to that of Goossens *et al.* when relative deadlines equal periods. Recently, a new schedulability test for constrained deadline systems was proposed by Bertogna *et al.* [6].

The schedulability test for global EDF depends on $u_{\max}$ — the lower the value of $u_{\max}$, the higher the total utilization of a task system that is schedulable. Nevertheless, even with $u_{\max}=0.5$, half the total available processing capacity will have to be wasted, if every deadline must be met. This may be overkill for soft real-time systems that can tolerate bounded deadline misses.

The research discussed above is for preemptive EDF (or, simply EDF), under which a job may be preempted by another arriving higher-priority job. To our knowledge, non-preemptive EDF (NP-EDF) has been considered only in [3], where a sufficient schedulability condition is derived for task systems in which the maximum execution cost of any task is less than the minimum period of any task.

Non-trivial schedulability tests have been developed for global RM scheduling also [1, 2]. However, the schedulable utilizations that these tests allow are less than that allowed by EDF. Furthermore, like partitioning algorithms, global RM (or any global static-priority algorithm) may not be suited for soft real-time systems. This is because, it is easy to construct task systems in which tardiness for low-priority tasks increases with time when scheduled under RM.

Contributions. In this paper, we address the issue of determining the amount by which any deadline may be missed under global EDF or NP-EDF on a multiprocessor, if the total system utilization is not restricted and may be any value up to m, where m is the total number of processors. We show that the tardiness bound depends on per-task utilizations and execution costs, and for EDF is at most $((m-1) \cdot e_{\max} - e_{\min})/(m - (m-2) \cdot u_{\max}) + e_{\max}$, where $e_{\max}$ and $e_{\min}$ are the maximum and minimum execution costs of any task. For NP-EDF, we derive a bound of $(m \cdot e_{\max} - e_{\min})/(m - (m-1) \cdot u_{\max}) + e_{\max}$, which is worse than that of EDF by over $e_{\max}/(m-(m-1)u_{\max})$. Better bounds, which may be possible by considering the $m-1$ tasks with highest utilizations and execution costs, are also provided in the paper. The difference between the EDF and NP-EDF bounds is approximately $e_{\max}/m$ when $u_{\max} \approx 0.0$, and increases to around $((m+1) \cdot e_{\max})/2$ as $u_{\max} \rightarrow 1.0$. However, for task systems in which much fewer than m tasks have an execution cost of $e_{\max}$ or a utilization of $u_{\max}$, the difference between the two bounds narrows considerably if computed using the more accurate methods described in the paper. It is also interesting to note that as $m \rightarrow \infty$, both bounds converge to $e_{\max}/(1-u_{\max})+e_{\max}$. Because a job may be blocked for up to $e_{\max}$ time units by a lower-priority job under NP-EDF, the small difference between the two bounds at low utilizations or for high values of m suggests that the analysis of EDF may not be tight. In a similar vein, the significantly higher difference at high utilizations, when close to m tasks have a utilization of $u_{\max}$ and an execution cost of $e_{\max}$, suggests that a better bound may be possible for NP-EDF. Nevertheless, these results should enable the class of soft real-time applications described above to be EDF-scheduled on multiprocessors. We consider the NP-EDF-result to be particularly significant in that, due to its potential to lower worst-case execution-cost estimates by eliminating preemptions, a large overall improvement in system utilization should be possible if bounded deadline misses are acceptable.

Other work on soft real-time systems. Most prior work on soft real-time systems has focussed on uniprocessor systems only. The main objective has been to decrease the response time for aperiodic tasks, or to derive probabilistic or deterministic bounds on deadline misses for task systems provisioned using average-case task parameters, as opposed to worst-case parameters [12]. While such complex formulations of soft real-time systems are important and useful for multiprocessor systems also, in this paper we seek to improve the system utilization of a multiprocessor for a simple model.

Soft real-time scheduling on multiprocessors has previously been considered in [13]. Here, a tardiness bound is derived for a suboptimal Pfair scheduling algorithm that is less expensive than optimal algorithms.

Organization. The rest of this paper is organized as follows. Our system model is formally presented in Sec. 2. Tardiness bounds are derived for EDF and NP-EDF in Sec. 3. In Sec. 4, an empirical evaluation of the tightness of the bounds is presented. Finally, Sec. 5 concludes.

2 System Model

A sporadic task system τ comprised of $n > 0$ independent, sporadic tasks is to be scheduled upon a multiprocessor platform comprised of $m \geq 2$ identical processors. Each task $T_i(e_i, p_i)$, where $1 \leq i \leq n$, is characterized by a minimum inter-arrival time, also referred to as its *period*, $p_i > 0$, an *execution cost* $e_i \leq p_i$, and a *relative deadline* $D_i = p_i$. Every task T_i may be invoked zero or more times with two consecutive invocations separated by at least p_i time units. Each invocation of T_i is referred to as a *job* of T_i and the k^{th} job of T_i, where $k \geq 1$, is denoted $T_{i,k}$. The first job may be invoked or *released* at any time at or after time zero. The release time of job $T_{i,k}$ is denoted $r_{i,k}$. A periodic task system, in which every two consecutive jobs of every task are separated by exactly p_i time units, is a special instance of a sporadic task system. Every job of T_i has a worst-case execution requirement of e_i time units. The *absolute deadline* (or just deadline) of $T_{i,k}$, denoted $d_{i,k}$ and given by $r_{i,k} + D_i$, is the time at or before which $T_{i,k}$ should complete execution. The notation $T_i(e_i, p_i)$ is sometimes used to concisely denote the execution cost and period of task T_i. Each task is sequential, and at any time may execute on at most one processor.

We say that a sporadic task system τ is *concrete*, if the release time of every job of each of its tasks is specified, and *non-concrete*, otherwise. Note that infinite concrete task systems can be specified for every non-concrete task system. When unambiguous from context, we omit specifying the type of the task system. The results in this paper are for non-concrete task systems, and hence hold for every concrete task system.

The *utilization* of T_i is denoted u_i and is given by e_i/p_i. The *total utilization* of a task system τ is defined as $U_{sum}(\tau) = \sum_{i=1}^{n} u_i$. We place no constraint on total utilization except that $U_{sum}(\tau) \leq m$ hold. The maximum utilization and the maximum execution cost of any task in τ are denoted $u_{\max}(\tau)$ and $e_{\max}(\tau)$, respectively. The minimum execution cost of any task is denoted $e_{\min}(\tau)$. $\mathcal{U}_{\max}(\tau, k)$, where $k \leq n$, denotes a subset of k tasks with highest utilizations in τ, *i.e.*, a subset of k tasks of τ each with utilization at least as high as that of every task in $\tau \setminus \mathcal{U}_{\max}(\tau, k)$.[†] $\mathcal{E}_{\max}(\tau, k)$ is defined analogously with respect to execution costs. (In all of these max and min terms, the task system will be omitted when it is unambiguous.) A task system is *preemptive* if the execution of its jobs may be interrupted and resumed later, and *non-preemptive*, otherwise. Under both preemptive EDF (or just EDF) and non-preemptive EDF (or NP-EDF), ready jobs are prioritized on the basis of their deadlines, with jobs with earlier deadlines having higher priority than jobs with later deadlines. Ties among jobs with the same deadline are resolved arbitrarily, but consistently. Under 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-EDF, the arriving job waits until some job completes execution and a processor becomes available. Thus, under NP-EDF, once scheduled, a job is guaranteed execution until completion without interruption.

As discussed in the introduction, this paper is concerned with deriving a lateness or *tardiness* [12] bound for a sporadic task system scheduled under EDF or NP-EDF. Formally, the tardiness of a job $T_{i,j}$ in schedule $\mathcal{S}$ is defined as $tardiness(T_{i,j}, \mathcal{S}) = \max(0, t - d_{i,j})$, where t is the time at which $T_{i,j}$ completes executing in $\mathcal{S}$. The tardiness of a task system τ under scheduling algorithm $\mathcal{A}$ is defined as the maximum tardiness of any job of

[†]$\setminus$ is the set difference operator. $A \setminus B$ is the subset of all the elements of A that are not in B.

a task in τ in any schedule for any concrete instantiation under $\mathcal{A}$. If κ is the maximum tardiness of any task system under $\mathcal{A}$, then $\mathcal{A}$ is said to *ensure a tardiness bound of* κ. Though tasks in a soft real-time system are allowed to have nonzero tardiness, we assume that *missed deadlines do not delay future job releases*. That is, if a job of a task misses its deadline, then the release time of the next job of that task remains unaltered. Of course, we assume that no two jobs of the same task can execute in parallel. Thus, a missed deadline effectively reduces the interval over which the next job should be scheduled in order to meet its deadline.

3 Tardiness Bounds for EDF and NP-EDF

Our approach towards determining tardiness bounds under EDF and NP-EDF involves comparing the allocations to a concrete task system τ in a processor sharing (PS) schedule for τ and an actual EDF or NP-EDF schedule of interest for τ, as the case may be, and quantifying the difference between the two. In a PS schedule, each job of T_i is allocated a fraction u_i of a processor at each instant (or equivalently, a fraction u_i of each instant) in the interval between its release time and its deadline. Because $D_i = p_i$ for all i, and $U_{sum} \leq m$ holds, the total demand at any instant will not exceed m in a PS schedule, and hence no deadlines will be missed; in fact, every job will complete executing exactly at its deadline. We begin by setting the required machinery in place.

3.1 Definitions and Notation

A time interval $[t_1, t_2)$, where $t_2 \geq t_1$, consists of all times t, where $t_1 \leq t < t_2$, and is of length $t_2 - t_1$. Interval (t_1, t_2) excludes t_1. The system start time is assumed to be zero. For any time $t > 0$, the notation t^- is used to denote a time instant that is earlier than t and is arbitrarily close to t. More specifically, $t^- \geq 0$ is the latest time instant before t such that the state of the system is unchanging in $[t^-, t)$ in the following sense: **(i)** There are no job releases or deadlines in (t^-, t). **(ii)** If a processor is executing a job, say $T_{i,j}$, at t^-, then it executes $T_{i,j}$ throughout $[t^-, t)$, and if a processor is idle at t^-, then it is idle throughout $[t^-, t)$. (In other words, t^- denotes the time $t - \epsilon$ in the limit $\epsilon \to 0+$.)

Definition 1 (active tasks, active jobs, and windows): A task T_i is said to be *active* at time t, if there exists a job $T_{i,j}$ such that $r_{i,j} \leq t < d_{i,j}$; $T_{i,j}$ is said to be the *active job* of T_i at t and the interval $[r_{i,j}, d_{i,j})$ is referred to as the *window* of $T_{i,j}$. By our task model, every task can have at most one active job at any time.

Definition 2 (pending jobs): $T_{i,j}$ is said to be *pending* at t in a schedule $\mathcal{S}$ if $r_{i,j} \leq t$ and $T_{i,j}$ has not completed execution by t in $\mathcal{S}$. Note that a job with a deadline at or before t is not considered to be active at t even if it is pending at t.

We now quantify the total allocation to τ in an interval $[t_1, t_2)$ in a PS schedule for τ, denoted PS_τ. For this, let $\mathsf{A}(\mathcal{S}, T_i, t_1, t_2)$ denote the total time allocated to T_i in an arbitrary schedule $\mathcal{S}$ for τ in $[t_1, t_2)$. Then, since in PS_τ, T_i is allocated a fraction u_i of each instant at which it is active in $[t_1, t_2)$, we have

$$\mathsf{A}(\mathsf{PS}_\tau, T_i, t_1, t_2) \leq (t_2 - t_1)u_i. \tag{1}$$

The total allocation to τ in the same interval in PS_τ is then given by

$$\mathsf{A}(\mathsf{PS}_\tau, \tau, t_1, t_2) \leq \sum_{T_i \in \tau} (t_2 - t_1)u_i = U_{sum} \cdot (t_2 - t_1) \leq m(t_2 - t_1). \tag{2}$$

We are now ready to define lag and LAG, which play a pivotal role in this paper. In the scheduling literature, the *lag* of task T_i at t in an arbitrary schedule $\mathcal{S}$ for τ is defined as the difference between the allocations to T_i in PS_τ and $\mathcal{S}$ in interval $[0, t)$. This difference, which is denoted $\mathsf{lag}(T_i, t, \mathcal{S})$ in this paper, is given by

$$\mathsf{lag}(T_i, t, \mathcal{S}) = \mathsf{A}(\mathsf{PS}_\tau, T_i, 0, t) - \mathsf{A}(\mathcal{S}, T_i, 0, t). \tag{3}$$

If $\mathsf{lag}(T_i, t, \mathcal{S})$ is positive, then schedule $\mathcal{S}$ has performed less work on the jobs of T_i until t than PS_τ (or T_i is under-allocated in $\mathcal{S}$), and more work if $\mathsf{lag}(T_i, t, \mathcal{S})$ is negative (or T_i is over-allocated in $\mathcal{S}$). The total lag of a task system τ at t, denoted $\mathsf{LAG}(\tau, t, \mathcal{S})$, is given by the sum of the lags of all tasks in τ. That is,

$$\mathsf{LAG}(\tau, t, \mathcal{S}) = \sum_{T_i \in \tau} \mathsf{lag}(T_i, t, \mathcal{S}) = \mathsf{A}(\mathsf{PS}_\tau, \tau, 0, t) - \mathsf{A}(\mathcal{S}, \tau, 0, t). \tag{4}$$

Note that $\mathsf{LAG}(\tau, 0, \mathcal{S})$ and $\mathsf{lag}(T_i, 0, \mathcal{S})$ are both zero, and that by (3) and (4), we have the following for $t_2 > t_1$.

$$\begin{aligned} \mathsf{lag}(T_i, t_2, \mathcal{S}) &= \mathsf{lag}(T_i, t_1, \mathcal{S}) + \mathsf{A}(\mathsf{PS}_\tau, T_i, t_1, t_2) - \mathsf{A}(\mathcal{S}, T_i, t_1, t_2) & (5) \\ \mathsf{LAG}(\tau, t_2, \mathcal{S}) &= \mathsf{LAG}(\tau, t_1, \mathcal{S}) + \mathsf{A}(\mathsf{PS}_\tau, \tau, t_1, t_2) - \mathsf{A}(\mathcal{S}, \tau, t_1, t_2) & (6) \end{aligned}$$

Finally, one more definition before beginning the derivation proper.

Definition 3 (busy interval): A time interval $[t_1, t_2)$, where $t_2 > t_1$, is said to be *busy* for τ if all m processors are executing some job of a task in τ throughout the interval, *i.e.*, no processor is ever idle in the interval or executes a job that is not of a task in τ. An interval $[t_1, t_2)$ that is not busy for τ, is said to be *non-busy* for τ, and is said to be *maximally non-busy* if every time instant in $[t_1, t_2)$ is non-busy, and either $t_1 = 0$ or $t_1{}_-$ is busy.

If $[t_1, t_2)$ is a busy interval in a schedule $\mathcal{S}$ for τ, then the tasks in τ receive a total allocation of $m(t_2 - t_1)$ time in $\mathcal{S}$ in that interval. By (2), the total allocation to τ in $[t_1, t_2)$ cannot exceed $m(t_2 - t_1)$ in PS_τ. Therefore, by (6), the LAG of τ at t_2 cannot exceed that at t_1, and we have the following lemma.

Lemma 1 *If* $\mathsf{LAG}(\tau, t + \delta, \mathcal{S}) > \mathsf{LAG}(\tau, t, \mathcal{S})$*, where* $\delta > 0$ *and* $\mathcal{S}$ *is a schedule for* τ*, then* $[t, t + \delta)$ *is a non-busy interval for* τ*.*

3.2 Deriving a Tardiness Bound for Preemptive EDF

Our goal is to determine a tardiness bound for non-concrete sporadic task systems. For this, we first determine a lower bound on the LAG at $d_{i,j}$ of an arbitrary concrete task system τ that is necessary for the tardiness of an arbitrary job $T_{i,j}$ of a task T_i in τ to exceed a given value in an EDF schedule. Next, we determine an upper bound on the LAG possible for τ at $d_{i,j}$, in terms of the parameters of the tasks in τ. Using these two values, it is then possible to determine a tardiness bound for τ. As we shall see, this tardiness bound that we derive is independent of release times of jobs in τ, and hence applies to all concrete task systems that are specified by the same non-concrete task system as τ, *i.e.*, to non-concrete task systems (Theorem 1). The lemma that follows determines a lower bound on LAG necessary for a given tardiness.

Lemma 2 *Let the deadline of every job of every task in a sporadic task system τ be at most t_d and let the tardiness of every job of T_k with a deadline less than t_d be at most $x + e_k$, where $x \geq 0$, for all $1 \leq k \leq n$, in an* EDF *schedule $\mathcal{S}$ for τ on m processors. Let $\mathsf{LAG}(\tau, t_d, \mathcal{S}) \leq mx + e_i$, where $t_d = d_{i,j}$, for some $1 \leq i \leq n$. Then, tardiness$(T_{i,j}, \mathcal{S})$ is at most $x + e_i$.*

Proof: Note that because the deadline of every job of a task in τ is at most t_d, $\mathsf{LAG}(\tau, t_d, \mathcal{S})$ denotes the amount of work that needs to be done on jobs of tasks in τ after t_d in $\mathcal{S}$. Because there are no new job arrivals at or after t_d, there can be no preemptions, either, at or after t_d. Let $\delta_i \leq e_i$ denote the amount of time that $T_{i,j}$ has executed for before time t_d and let $y = x + \delta_i/m$. We consider two cases depending on whether $[t_d, t_d + y)$ is busy.

Case 1: $[t_d, t_d + y)$ is busy. In this case, the amount of work completed in $[t_d, t_d + y)$ is exactly $my = mx + \delta_i$, and hence, the amount of work pending at $t_d + y$ is at most $e_i - \delta_i$. Thus, the latest time that $T_{i,j}$ resumes execution after t_d is $t_d + y$, and because there are no preemptions at or after t_d, $T_{i,j}$ completes execution at or before $t_d + y + e_i - \delta_i \leq t_d + x + e_i$. Hence, *tardiness*$(T_{i,j}, \mathcal{S}) \leq t_d + x + e_i - d_{i,j} = x + e_i$. This is illustrated in Fig. 1(a).

Case 2: $[t_d, t_d + y)$ is not busy. Let t' denote the first (earliest) non-busy instant in $[t_d, t_d + y)$. Then, because every job has its release time before t_d, and hence is ready at t' if its predecessor job has completed executing by t', we have the following.

(J) At most $m - 1$ tasks have pending jobs at or after t'.

Therefore, if $T_{i,j}$ has not completed executing before $t_d + y$, then either $T_{i,j}$ or a prior job of T_i should be executing at t'. If $T_{i,j}$ is executing at t', then because there are no preemptions after t_d and $t' < t_d + y$ holds, $T_{i,j}$ will complete executing before $t_d + y + e_i - \delta_i \leq t_d + x + e_i$. Thus, the tardiness of $T_{i,j}$ is less than $x + e_i$, which establishes the lemma. The remaining possibility is that $j > 1$ holds, and that a job of T_i that is prior to $T_{i,j}$ is executing at t'. In this case, $T_{i,j}$ could not have executed before t_d, and hence $\delta_i = 0$ and $y = x$ holds.

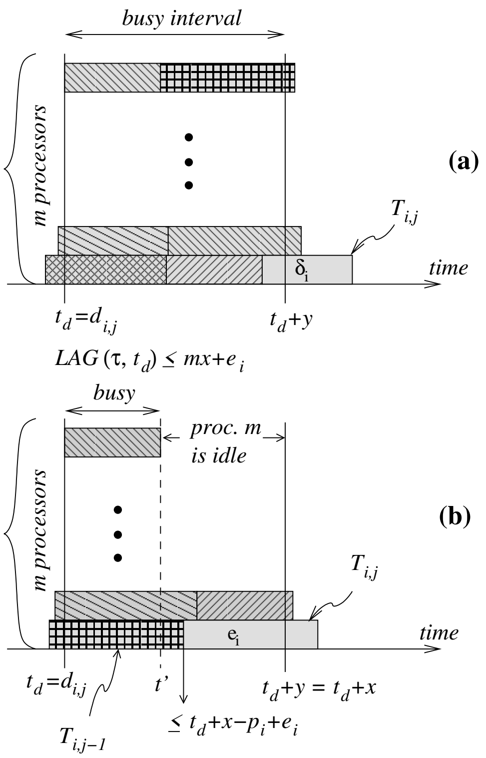

Figure 1: Lemma 2. **(a)** $[t_d, t_d + y)$ is busy. $T_{i,j}$ commences execution at or before $t_d + y$. **(b)** $[t_d, t_d + y)$ is not busy.

Because successive jobs of T_i are separated by at least p_i time units, $d_{i,j-1} \leq t_d - p_i$ holds. Hence, by the statement of the lemma, $T_{i,j-1}$ completes executing by $t_d - p_i + x + e_i$. Thus, the latest time that a prior job of T_i could complete executing is $t_d - p_i + x + e_i \leq t_d + x$. Because $t' < t_d + y = t_d + x$, by (J), $T_{i,j}$ commences execution at or before $t_d + x$, and hence, completes executing by $t_d + x + e_i$. This is illustrated in Fig. 1(b). ■

By Lemma 1, the LAG of τ can increase only across an interval that is non-busy. Thus, if t' is the end of the latest maximally non-busy interval in $[0, t_d)$ in a schedule $\mathcal{S}$ for τ (*i.e.*, t'^- is non-busy and $[t', t_d)$ is busy), then $\mathsf{LAG}(\tau, t) \leq \mathsf{LAG}(\tau, t')$, for all $t' \leq t < t_d$. (If every instant in $\mathcal{S}$ is busy, then $\mathsf{LAG}(\tau, t, \mathcal{S}) \leq 0$, for all t.) Hence, an upper bound on the LAG of τ at the end of the latest maximally non-busy interval at or before t_d would serve as an upper bound on the LAG of τ at t_d. The next lemma derives such an upper bound on LAG.

Lemma 3 *Let the deadline of every job of every task in a sporadic task system τ with $U_{sum} \leq m$ be at most t_d and let $\mathcal{S}$ be an* EDF *schedule for τ on m processors. Let the tardiness in $\mathcal{S}$ of every job of T_i with a deadline less than t_d be at most $x + e_i$, where $x \geq 0$, for all $1 \leq i \leq n$. Let $[t, t')$, where $0 \leq t < t' \leq t_d$, be a maximally non-busy interval in $[0, t_d)$, and let k denote the number of jobs of tasks in τ that are executing at the end of the interval. Then, $k \leq m - 1$ and $\mathsf{LAG}(\tau, t', \mathcal{S})$ is at most $x \cdot \sum_{T_i \in \mathcal{U}_{\max}(k)} u_i + \sum_{T_i \in \mathcal{E}_{\max}(k)} e_i$.*

Proof: Recall that t'^- denotes a time instant before t' that is arbitrarily close to t'. By the statement of the lemma, t'^- is non-busy, and by (4), the LAG of τ at t' is given by the sum of the the lags at t' of tasks in τ. Therefore, an upper bound on the LAG of τ at t' can be determined by determining an upper bound on the lag at t' of each task in τ. For this, we partition the tasks in τ into subsets α, γ, β, and ρ, as defined below, and then determine an upper bound on the lag of a task in each subset. This partitioning is illustrated in Fig. 2.

$\alpha \stackrel{\text{def}}{=}$ subset of all tasks in τ executing throughout $[t, t')$

$\gamma \stackrel{\text{def}}{=}$ subset of all tasks in τ not executing at least in part of $[t, t')$, but executing at t'^-

$\beta \stackrel{\text{def}}{=}$ subset of all tasks in τ not executing at t'^-, but are active at t'^-

$\rho \stackrel{\text{def}}{=}$ subset of all tasks in τ not executing at t'^- and inactive at 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 at t. Let δ_i denote the amount of time that $T_{i,j}$ has executed before t in $\mathcal{S}$. We first determine the lag of T_i at t by considering two cases depending on $d_{i,j}$. (We consider t' afterwards.)

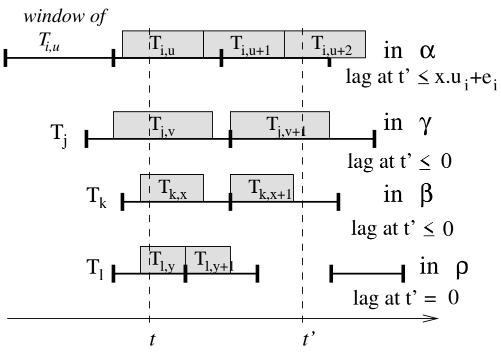

Figure 2: Lemma 3. Partitioning of tasks in τ. Jobs of a sample task in each subset are shown. In this figure, all jobs except those of T_i, which is in α, complete executing by their deadlines.

Case 1: $d_{i,j} < t$. Because $T_{i,j}$ is executing at t and a newly arriving job, whose deadline would be greater than t, cannot preempt $T_{i,j}$, the latest time that $T_{i,j}$ completes executing is $t + e_i - \delta_i$. (It could complete earlier if its actual execution time is less than e_i.) By the statement of the lemma, the tardiness of every job of T_i with a deadline less than t_d is at most $x + e_i$. Therefore, $d_{i,j} \geq t + e_i -$

$\delta_i - (x + e_i) = t - \delta_i - x$ holds. (Note that $d_{i,j} \geq t - \delta_i - x$ holds even if $T_{i,j}$ completes early, because the tardiness bound should hold for the worst case, when $T_{i,j}$ executes for a full e_i time units.) In PS_τ, $T_{i,j}$ completes execution at $d_{i,j}$, and the jobs of T_i succeeding $T_{i,j}$ are allocated a share of u_i in every instant in their windows. Because windows of no two jobs overlap, T_i is allocated a share of u_i in every instant in $[d_{i,j}, t)$ in which it is active. Thus, the under-allocation to T_i in $\mathcal{S}$ in $[0, t)$ is equal to the sum of the under-allocation to $T_{i,j}$ in $\mathcal{S}$, which is $e_i - \delta_i$, and the allocation to later jobs of T_i in $[d_{i,j}, t)$ in PS_τ. Hence, $\mathsf{lag}(T_i, t, \mathcal{S})$ is at most $e_i - \delta_i + (t - d_{i,j}) \cdot u_i \leq e_i - \delta_i + (x + \delta_i) \cdot u_i$.

Case 2: $d_{i,j} \geq t$. In this case, the amount of work done by PS_τ on $T_{i,j}$ up to time t is given by $e_i - (d_{i,j} - t) \cdot u_i$. Because all prior jobs of T_i complete execution by t in both $\mathcal{S}$ and PS_τ, and $T_{i,j}$ has executed for δ_i time units before t, $\mathsf{lag}(T_i, t, \mathcal{S}) = e_i - (d_{i,j} - t) \cdot u_i - \delta_i \leq e_i - \delta_i \leq e_i - \delta_i + (x + \delta_i) \cdot u_i$.

Thus, in both cases, we have

$$\mathsf{lag}(T_i, t, \mathcal{S}) \leq e_i - \delta_i + (x + \delta_i) \cdot u_i. \tag{7}$$

Next, to determine the lag of T_i at t', we determine the allocations to T_i in $[t, t')$ in $\mathcal{S}$ and in PS_τ. In PS_τ, T_i is allocated a share of at most u_i at every instant in $[t, t')$, for a total allocation of at most $(t' - t) \cdot u_i$. Because T_i executes throughout $[t, t')$ in $\mathcal{S}$, its total allocation in $\mathcal{S}$ in $[t, t')$ is $t' - t$. Hence, $\mathsf{A}(\mathsf{PS}_\tau, T_i, t, t') - \mathsf{A}(\mathcal{S}, T_i, t, t') \leq (u_i - 1) \cdot (t' - t)$, and so, by (5) and (7),

$$\mathsf{lag}(T_i, t', \mathcal{S}) \leq e_i - \delta_i + (x + \delta_i) \cdot u_i + (u_i - 1) \cdot (t' - t) \leq e_i + x \cdot u_i \quad \{\text{because } t' > t,\ u_i \leq 1, \text{ and } \delta_i \geq 0\}. \tag{8}$$

Upper bound on the lag of a task in γ. Let T_i be a task in γ and let t'', where $t < t'' < t'$, denote the latest time at or before t' that T_i transitions from a non-executing to an executing state. Because $[t, t')$ is maximally non-busy and T_i is not executing at t''_-, jobs of T_i released before t'' complete execution in $\mathcal{S}$ before t'', and a job of T_i is released at t''. Hence, the total allocations to T_i in $[0, t'')$ are equal in both $\mathcal{S}$ and PS_τ, and so, $\mathsf{lag}(T_i, t'', \mathcal{S}) = 0$. Therefore, $\mathsf{lag}(T_i, t', \mathcal{S}) = \mathsf{A}(\mathsf{PS}_\tau, t'', t') - \mathsf{A}(\mathcal{S}, t'', t') \leq (t' - t'')u_i - (t' - t'') \leq 0$.

Upper bound on the lag of a task in β. Let $T_{i,j}$ be the active job at t'^- of a task T_i in β. Then, $T_{i,j}$ is not executing at t'^- in $\mathcal{S}$ and $d_{i,j} > t'^-$ holds. By the definition of t'^-, $d_{i,j} \geq t'$ holds. Thus, since EDF is a work-conserving algorithm that does not idle a processor when there is pending work, and at least one processor is idle throughout $[t, t')$, $T_{i,j}$ completes execution before t'^-. However, in PS_τ, $T_{i,j}$ would not complete execution until $d_{i,j}$, and hence, T_i is not under-allocated in $[0, t')$ in $\mathcal{S}$. Thus, T_i's lag at t' is at most zero.

Upper bound on the lag of a task in ρ. Reasoning here is similar to that used for tasks in β. Let T_i be a task in ρ. Then, T_i is not executing at t'^- in $\mathcal{S}$, no job of T_i is active at t'^-, and the deadline of every job of T_i released before t'^- is at most t'^-. Hence, because EDF is work-conserving and at least one processor is idle at t'^-, all jobs of T_i released before t'^- complete execution by t'^- in $\mathcal{S}$. Because their deadlines are at most t'^-, these jobs complete execution by t'^- in PS_τ also. There are no new job releases for T_i in $[t'^-, t')$, and thus, the total allocations to T_i in $\mathcal{S}$ and PS_τ in $[0, t')$ are equal. Therefore, $\mathsf{lag}(T_i, t', \mathcal{S}) = 0$.

By (4), the LAG of τ at t' is given by the sum of the lags of tasks in subsets α, β, γ, and ρ. As shown above, only tasks in α may have a positive lag, and thus, $\mathsf{LAG}(\tau, t', \mathcal{S}) = \sum_{T_i \in \alpha \cup \gamma \cup \beta \cup \rho} \mathsf{lag}(T_i, t', \mathcal{S}) \leq \sum_{T_i \in \alpha} \mathsf{lag}(T_i, t', \mathcal{S})$, and by (8), $\mathsf{LAG}(\tau, t', \mathcal{S}) \leq \sum_{T_i \in \alpha}(e_i + x \cdot u_i)$. Since $[t, t')$ is maximally non-busy and k tasks are executing at the end of the interval, $k \leq m - 1$ holds. Hence, since a task in α executes throughout $[t, t')$, there could be at most k tasks in α. Therefore, $\mathsf{LAG}(\tau, t', \mathcal{S}) \leq x \cdot \sum_{T_i \in \mathcal{U}_{\max}(k)} u_i + \sum_{T_i \in \mathcal{E}_{\max}(k)} e_i$. ■

Finally, to determine a tardiness bound for EDF, we are left with determining as small an x as possible such that the upper bound given by Lemma 3 is at most the lower bound required in Lemma 2. We do this next.

Theorem 1 *The tardiness of every task T_k of a sporadic task system τ, where $U_{sum}(\tau) \leq m$, is at most*

$$\frac{(\sum_{T_i \in \mathcal{E}_{\max}(m-1)} e_i) - e_{\min}}{m - \sum_{T_i \in \mathcal{U}_{\max}(m-1)} u_i} + e_k \tag{9}$$

in any EDF *schedule for τ on m processors.*

Proof: Suppose the theorem does not hold. Then, there exists a concrete task system $\hat{\tau}$ whose task parameters are the same as τ such that the tardiness of some job of $\hat{\tau}$ exceeds the bound in (9) in an EDF schedule $\mathcal{S}$ for $\hat{\tau}$. Let $T_{k,\ell}$ denote one such job such that the deadline of $T_{k,\ell}$ is not later than that of any such job. Let τ' denote the task system obtained from $\hat{\tau}$ by removing all jobs with deadlines exceeding $d_{k,\ell}$, and let $\mathcal{S}'$ be an EDF schedule for τ'. Assuming an EDF scheduler that resolves ties among jobs consistently, every job in τ' will have identical schedules in both $\mathcal{S}$ and $\mathcal{S}'$, and hence, the tardiness of every job in τ' will be the same in both schedules.

Let $t_d = d_{k,\ell}$. Then, since $T_{k,\ell}$ misses its deadline, $\mathsf{lag}(T_{k,\ell}, t_d, \mathcal{S}') > 0$. Also, since no job in τ' has a deadline exceeding t_d, no task of τ' is over-allocated in $[0, t_d)$ in $\mathcal{S}'$, *i.e.*, no task of τ' has a negative lag at t_d. Thus, by (4), $\mathsf{LAG}(\tau', t_d, \mathcal{S}') > 0$. Since $\mathsf{LAG}(\tau', 0, \mathcal{S}') = 0$, by Lemma 1, there exists a non-busy interval in $[0, t_d)$ in $\mathcal{S}'$. Let t' be the end of the latest maximally non-busy interval before t_d. Then, as discussed earlier, by Lemma 1, $\mathsf{LAG}(\tau', t_d, \mathcal{S}') \leq \mathsf{LAG}(\tau', t', \mathcal{S}')$. By our definition of $T_{k,\ell}$, the tardiness of every job of task T_j with deadline less than t_d is at most $x + e_j$ for all $1 \leq j \leq n$, where

$$x = \frac{\sum_{T_i \in \mathcal{E}_{\max}(m-1)} e_i - e_{\min}}{m - \sum_{T_i \in \mathcal{U}_{\max}(m-1)} u_i}. \tag{10}$$

Hence, by Lemmas 1 and 3,

$$\mathsf{LAG}(\tau', t_d, \mathcal{S}') \leq \mathsf{LAG}(\tau', t', \mathcal{S}') \leq x \cdot \sum_{T_i \in \mathcal{U}_{\max}(m-1)} u_i + \sum_{T_i \in \mathcal{E}_{\max}(m-1)} e_i. \tag{11}$$

By our assumption, the tardiness of $T_{k,\ell}$ exceeds $x + e_k$. Hence, by Lemma 2, $\mathsf{LAG}(\tau', t_d, \mathcal{S}') > mx + e_k$, which by (11) implies $mx + e_k < x \cdot \sum_{T_i \in \mathcal{U}_{\max}(m-1)} u_i + \sum_{T_i \in \mathcal{E}_{\max}(m-1)} e_i$, *i.e.*, $x < \frac{\sum_{T_i \in \mathcal{E}_{\max}(m-1)} e_i - e_{\min}}{m - \sum_{T_i \in \mathcal{U}_{\max}(m-1)} u_i}$. This contradicts (10), and hence the theorem follows. ■

A sufficient condition on the individual task utilizations for a given tardiness bound, that places no restriction on the total utilization can be derived from the above theorem, as given by the following corollary.

Corollary 1 EDF *ensures a tardiness of at most $x+e_k$ for every task T_k of a sporadic task system τ on m processors if the sum of the utilizations of the tasks in $\mathcal{U}_{\max}(m-1)$, $\sum_{T_i \in \mathcal{U}_{\max}(m-1)} u_i$, is at most $\frac{mx - \sum_{T_i \in \mathcal{E}_{\max}(m-1)} e_i + e_{\min}}{x}$ and $U_{sum}(\tau) \leq m$.*

Proof: By Theorem 1, the tardiness for task T_k of τ in an EDF schedule is at most $\frac{(\sum_{T_i \in \mathcal{E}_{\max}(m-1)} e_i) - e_{\min}}{m - \sum_{T_i \in \mathcal{U}_{\max}(m-1)} u_i} + e_k$. Therefore, a tardiness not exceeding $x + e_k$ can be guaranteed if $\frac{(\sum_{T_i \in \mathcal{E}_{\max}(m-1)} e_i) - e_{\min}}{m - \sum_{T_i \in \mathcal{U}_{\max}(m-1)} u_i} \leq x$ holds. On rearranging the terms, we arrive at the condition in the corollary. ■

3.3 Improving Accuracy and Speed

In this subsection, we discuss possible improvements to the accuracy of the tardiness bound in Theorem 1 and the time required to compute it.

Improving the bound. Recall that in determining an upper bound on LAG in Lemma 3, we assumed that $m-1$ tasks are executing in the maximally non-busy interval, and that each such task has the highest possible lag. However, using some additional reasoning, it can be shown that for LAG to increase across a non-busy interval, there should be at least one task that is active in the interval, but had completed execution before the start of the interval (*i.e.*, β is non-empty). This fact can then be used to argue that at least one job executing at the end of the non-busy interval has its deadline at or beyond the end of the interval, and hence that the lag of its task is at most e_k, from which $x \sum_{T_k \in \mathcal{U}_{\max}(m-2)} u_k + \sum_{T_k \in \mathcal{E}_{\max}(m-1)} e_k$ follows as an upper bound on LAG. Therefore, an improved tardiness bound can be obtained by using $\sum_{T_k \in \mathcal{U}_{\max}(m-2)} u_k$ instead of $\sum_{T_k \in \mathcal{U}_{\max}(m-1)} u_k$ in (9). Similarly, it is sufficient to apply the utilization condition of Corollary 1 to tasks in $\mathcal{U}_{\max}(m-2)$. We will refer to the bound evaluated with $\mathcal{U}_{\max}(m-2)$ in (9) as EDF-BASIC.

Two-processor systems. With the reasoning described above, the upper bound on LAG given by Lemma 3 reduces to $e_{\max}$ for $m = 2$, which is independent of x. Hence, unlike in the general case (*i.e.*, arbitrary m), we are not constrained to determine an x that would apply for all tasks, and it can be shown that tardiness for task T_i is at most $(e_{\max} - e_i)/2 + e_i$, which is at most $e_{\max}$.

If the time taken to compute a tardiness bound or sufficient task utilizations is not a concern, then some more improvement may be possible by relaxing another pessimistic assumption that we make in Lemma 3. We not only assume that the tasks that are executing at the end of the non-busy interval have the highest utilizations, but also that they have the highest execution costs. This can be relaxed by sorting tasks by non-increasing order of $x \cdot u_k + e_k$ (where as defined in Lemma 3, the tardiness of T_k is at most $x + e_k$) and by using the top $m-2$ tasks, denoted $\mathcal{E}$-$\mathcal{U}(m-2)$, in the expressions. (The $(m-1)^{st}$ execution cost should be taken as the maximum of the execution costs of the remaining tasks.) If x is known, as in applying Corollary 1, then this procedure is straightforward. Nevertheless, even when seeking x, as in the proof of Theorem 1, an iterative procedure could be used. In this iterative procedure, the bound given by EDF-BASIC will be used as the initial value for x. This initial value will then be used to construct $\mathcal{E}$-$\mathcal{U}(m-2)$, whose utilizations and execution costs should be used in improving x. The procedure should be continued until the task set $\mathcal{E}$-$\mathcal{U}(m-2)$ converges. (It is easy to show that convergence is guaranteed.) We will refer to the bound computed using such an iterative procedure as EDF-ITER. This procedure is illustrated in the example below.

Example. Let τ be a task set comprised of the following 8 tasks scheduled under EDF on $m = 4$ processors: $T_1(15, 150)$–$T_4(15, 150)$, $T_5(9, 10)$–$T_8(9, 10)$. For this task set, $\mathcal{U}_{\max}(\tau, m-2) = \{T_5, T_6\}$, $\mathcal{E}_{\max}(\tau, m-1) = \{T_1, T_2, T_3\}$ (where ties are resolved by task indices), and $e_{\min} = 9$. Therefore, a tardiness bound for T_k obtained using EDF-BASIC is $(45-9)/(4-18/10)+e_k$, which equals $360/22+e_k$. Thus, $x \approx 16.36$. Computing $x \cdot u_k + e_k$ for all the tasks, we obtain values of 16.636 for tasks T_1–T_4, and 23.727 for T_5–T_8. Hence, $\mathcal{E}$-$\mathcal{U}(m-2) = \{T_5, T_6\}$. Using the execution costs and utilizations of tasks in $\mathcal{E}$-$\mathcal{U}(m-2)$ and 15 as the $(m-1)^{st}$ execution cost, we obtain an improved value of 10.9 for x, which is more than 5 units less than the initial value. The tasks in $\mathcal{E}$-$\mathcal{U}(m-2)$ are not altered in the next iteration, and so, the procedure terminates.

Improving computation time. Computing the tardiness bound or determining utilization restrictions on tasks for a given bound involves selecting $m-1$ tasks with highest utilizations, and/or $m-1$ tasks with highest execution costs, and summing the values. Hence, these are $O(n+m)$ computations. If speed is a concern, as in online admission tests, and if $u_{\max}$ and $e_{\max}$ are known, then $O(1)$ computations, which assume a utilization of $u_{\max}$ and an execution cost of $e_{\max}$ for each of the $m-1$ tasks, and hence yield more pessimistic values, can be used. Under this assumption, tardiness under EDF is at most $((m-1)e_{\max} - e_{\min})/(m-(m-2)u_{\max}) + e_i$. Similarly, for a tardiness of at most $x + e_i$ for every T_i, it is sufficient that $u_{\max}$ be at most $(mx-(m-1)e_{\max} + e_{\min})/((m-2)x)$. We will refer to the bound computed using $u_{\max}$ and $e_{\max}$ as EDF-FAST.

Tightness of the bound. As discussed in the introduction, we do not believe that our result for EDF is tight. However, we can show that our result is off by at most $e_{\max}$ for small values of $u_{\max}$. For this, consider a task system τ comprised of a primary task $T_1(e_1, p_1)$ and $(m-u_1)p_1/\delta$ auxiliary tasks. Let $\delta \ll e_1$ and p_1 be the execution cost and period, respectively, of each auxiliary task. Let δ divide $(m-u_1)$ evenly and let p_1 be a multiple of m. Suppose the first job of each task is released at time 0 and suppose the first job of every auxiliary task is scheduled before $T_{1,1}$ and executes for a full δ time units. In such a schedule, the auxiliary jobs will execute continuously until time $(\frac{(m-u_1)p_1}{\delta} \times \delta)/m = p_1 - e_1/m$ on each processor. Hence, T_1 will not begin executing until time $p_1 - e_1/m$, and hence would not complete until $p_1 + e_1(1-1/m)$ for a tardiness of $e_1(1-1/m)$. By choosing $e_1 = m$, this tardiness can be made to equal $e_1 - 1$. In this example, $e_{\max} = e_1$, and hence tardiness is at least $e_{\max} - 1$. Note that when $u_{\max}$ is arbitrarily low, tardiness under EDF computed using EDF-FAST is at most $((m-1) \cdot e_{\max})/m + e_{\max}$, and since e_1/p_1 can be arbitrarily low in the above example, our result is off by at most $e_{\max}$.

Earlier in this section, we noted that tardiness is at most $e_{\max}$ for two-processor systems. The following is an example that shows that this result is tight. Let $T_1(1, 2)$, $T_2(1, 2)$, and $T_3(2k+1, 2k+1)$, where $k \geq 1$, be three periodic tasks all of whose first jobs are released at time zero. If deadline ties are resolved in favor

T_1 and T_2 over T_3, then on two processors, tardiness for jobs of T_3 can be as high as $2k$ time units. (If T_3 is favored, then its jobs can miss by $2k - 1$.) For this task set, estimated tardiness is $e_{\max} = 2k + 1$, and for large k, the difference between estimated and actual tardiness is negligible.

An empirical evaluation of the accuracy of the bounds is available in Sec. 4. Interestingly, we have found that tardiness can exceed $e_{\max}$ even for task systems with $u_{\max}$ near 1/2. The following is one such task set: $T_1(1,2)$–$T_4(1,2)$, $T_5(1,5)$–$T_7(1,5)$, $T_8(1,11)$, $T_9(34,110)$, $T_{10}(23,63)$, $T_{11}(7,18)$–$T_{12}(7,18)$, $T_{13}(3,7)$–$T_{14}(3,7)$. The total utilization of this task set is five. When scheduled on five processors with deadline ties resolved using task indices, a job of T_9 misses its deadline at time 7295 by $35 > 34 = e_{\max}$ time units. (The EDF-BASIC and EDF-ITER bounds for this task set are 54 and 51.78, respectively.) Hence, the best bound that can be hoped for an arbitrary task set definitely exceeds $e_{\max}$.

3.4 Deriving a Tardiness Bound for NP-EDF

In this subsection, a tardiness bound is derived for NP-EDF. The analysis required for NP-EDF differs slightly from that of EDF due to *priority inversions*.

Priority inversions, blocked jobs, and blocking jobs. Under NP-EDF, once scheduled, each job continues to execute until completion without any interruption. Therefore, a job arriving at time t, even if ready at t and has a deadline that is earlier than some other job executing at t, 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 as a *blocking job*. 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. Fig. 3 gives an example. In this example, because job $T_{4,2}$ is executing in $[4, 5)$, task T_4 is not blocked in $[4, 5)$ even though job $T_{4,3}$, which has a higher priority than $T_{2,1}$, is not executing. (Similarly, $T_{4,3}$ is not blocked in $[4, 5)$ because it is not ready.)

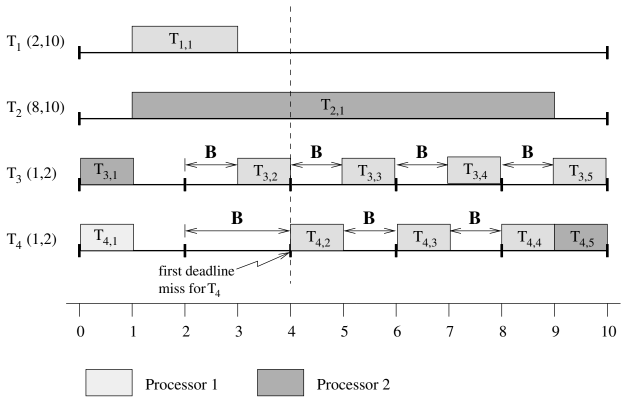

Figure 3: Priority inversion under NP-EDF. Intervals in which tasks T_3 and T_4 are blocked are indicated using a "B." T_3 and T_4 are blocked by jobs $T_{1,1}$ and $T_{2,1}$ in $[2, 3)$. In addition, T_3 is blocked in $[4, 5)$, $[6, 7)$, and $[8, 9)$, while T_4 is blocked in $[3, 4)$, $[5, 6)$, and $[7, 8)$.

Under EDF, jobs with deadlines later than t_d do not interfere with jobs with deadlines at most t_d. Hence in Sec. 3.2 (Theorem 1), while determining an upper bound on LAG at t_d in order to derive a tardiness bound for an arbitrary job with deadline at t_d under EDF, it was sufficient to consider a concrete task system comprised of jobs with deadlines at most t_d. However, under NP-EDF, due to the possibility of priority inversions, not all jobs with deadlines later than t_d can be ignored from the concrete task systems that we work with. Let τ be a concrete task system and let Ψ denote the set of all jobs of tasks in τ with deadlines at most t_d.Then, to determine a tardiness bound for a job with deadline at t_d, we need an estimate of the amount of work pending at t_d for jobs in Ψ. However, because not every job in τ has deadline at most t_d, using an upper bound on LAG of τ at t for this does not suffice. To see this, let $\mathcal{S}$ be the schedule in Fig. 3. Letting $t_d = 4$ in this example, we have $\Psi = \{T_{3,1}, T_{3,2}, T_{4,1}, T_{4,2}\}$. The amount of work pending for Ψ at t_d is equal to the amount of work pending for $T_{4,2}$, which is one. However, since the lags of T_1 through T_4 are -1.2, 0.2, 0, and 1, in that order, $\mathsf{LAG}(\tau, t_d, \mathcal{S}) = \sum_{i=1}^{n} \mathsf{lag}(T_i, t_d, \mathcal{S}) = 0$ holds. Thus, because tasks of blocking jobs *may* have negative lags, the LAG of τ may be an underestimate of the amount of work pending for Ψ. To deal with this, we extend the notion of lag to apply to jobs and job sets, and determine the lag of jobs in Ψ at t_d. Finally, since a job not in Ψ that commences execution before t_d may execute after t_d, an estimate on the amount of such blocking work pending at t_d is also required. To properly handle this, we formally define intervals in which jobs not in Ψ may be executing and the amount of work pending for such blocking jobs at any time t. We begin by formally defining lag for jobs.

Lag for jobs. The notion of lag can be applied to a job or a set of jobs in an obvious manner. Just as with tasks, we will use the terms lag and LAG to denote the lag of an individual job and that of a set of jobs, respectively. Let τ denote a concrete task system and Ψ a subset of jobs in τ. Let $\mathsf{A}(\mathsf{PS}_\tau, T_{i,j}, t_1, t_2)$ and $\mathsf{A}(\mathcal{S}, T_{i,j}, t_1, t_2)$ denote the allocations to $T_{i,j}$ in the interval $[t_1, t_2)$ in PS_τ and $\mathcal{S}$ respectively. (These values will depend on t_1, t_2, $r_{i,j}$, and $d_{i,j}$.) Then, $\mathsf{lag}(T_{i,j}, t, \mathcal{S}) = \mathsf{A}(\mathsf{PS}_\tau, T_{i,j}, r_{i,j}, t) - \mathsf{A}(\mathcal{S}, T_{i,j}, r_{i,j}, t)$, and LAG of Ψ at t can be defined analogously to the definition in (4) as $\mathsf{LAG}(\Psi, t, \mathcal{S}) = \sum_{T_{i,j} \in \Psi} \mathsf{lag}(T_{i,j}, t, \mathcal{S})$. The total allocation in $[0, t)$, where $t > 0$, to a job that is neither pending at t^- in $\mathcal{S}$ nor is active at t^- is the same in both $\mathcal{S}$ and PS_τ, and hence, its lag at t is zero. Therefore, for $t > 0$, we have the following.

$$\mathsf{LAG}(\Psi, t, \mathcal{S}) = \sum_{\substack{\{T_{i,j} \text{ is in } \Psi, \text{ and is} \\ \text{pending or active at} \\ t^-\}}} \mathsf{lag}(T_{i,j}, t, \mathcal{S}).$$

The above expression for LAG of Ψ can be expressed in terms of lags of tasks in τ as follows.

$$\mathsf{LAG}(\Psi, t, \mathcal{S}) \leq \sum_{\substack{\{T_i \in \tau \,:\, T_{i,j} \text{ is in } \Psi, \text{ and} \\ \text{is pending or active at } t^-\}}} \mathsf{lag}(T_i, t, \mathcal{S}). \tag{12}$$

The next definition identifies intervals in which jobs in Ψ are blocked. In this definition and in the rest of the paper, Ψ is to be taken as the set of all jobs of tasks in τ with deadlines at most t_d.

Definition 4 (blocking interval): An interval $[t_1, t_2)$ in $\mathcal{S}$, where $0 \leq t_1 < t_2 \leq t_d$ and $\mathcal{S}$ is a schedule for τ, is said to be a *blocking interval* for Ψ, if at least one job of Ψ is blocked in $[t_1, t_2)$ by a job not in Ψ, *i.e.*, a job with a deadline exceeding t_d. $[t_1, t_2)$ is said 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)): The set of all jobs, of tasks in τ, not in Ψ, that block one or more jobs of Ψ at some time before t and may continue to execute at t in $\mathcal{S}$ is denoted $\mathcal{B}(\tau, \Psi, t, \mathcal{S})$ and the total amount of time that the jobs in $\mathcal{B}(\tau, \Psi, t, \mathcal{S})$ will execute beyond t, *i.e.*, the total amount of work pending for those jobs at t, is denoted $\mathsf{B}(\tau, \Psi, t, \mathcal{S})$. The set of all jobs, of tasks in τ, not in Ψ, that can block some job in Ψ under NP-EDF at some time is denoted $\mathcal{B}(\tau, \Psi,$NP-EDF$)$, or simply $\mathcal{B}$, when the parameters are obvious.

Non-busy interval categories. With respect to 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 Lemma 1 applies to Ψ also in a schedule for τ. 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. For example, in Fig. 3, if $t_d = 4$, then $T_{1,1}$ and $T_{2,1}$ are not in Ψ. Therefore, $[1, 2)$ is a non-busy interval for Ψ. However, $[1, 2)$ is not a blocking interval, because no job of Ψ is awaiting execution in $[1, 2)$. Thus, a non-busy interval in an NP-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*.

Differences in reasoning. Our approach for deriving a tardiness bound for NP-EDF differs from that used for EDF in the following. (In this description, every non-busy interval or a blocking interval is to be taken to be maximal, unless otherwise stated. We refrain from explicitly saying so for conciseness.)

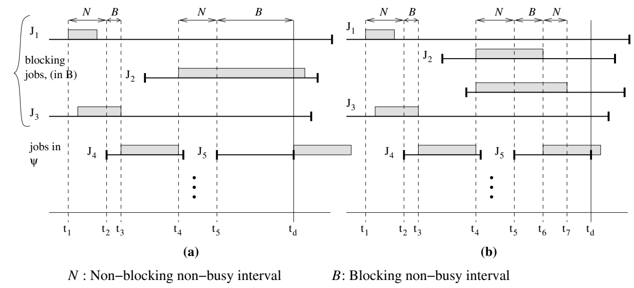

Figure 4: Illustration of the reasoning required for NP-EDF.

By Lemma 1, we know that the LAG of Ψ can increase only across a non-busy interval (not necessarily maximal), and hence, to determine an upper bound on 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-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. 4(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. 4(b), an upper bound on LAG will be determined at t_7 by considering 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 a such an interval is identical to that used for EDF in Lemma 3. (In Lemma 3, 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 that is blocking, there is at least one task whose 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. 4(a), J_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 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 execution 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.*, $\mathsf{LAG}(\Psi, t_d, \mathcal{S})$, we also need to determine an upper bound on the total amount of work pending for the jobs that are blocking those of Ψ at t_d, *i.e.*, $\mathsf{B}(\tau, t_d, \mathcal{S})$. In our example in Fig. 4(a), assuming that J_2 is the only pending blocking job at t_d, we will also need an estimate of the amount of time that J_2 will execute after t_d. In Fig. 4(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 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 deadline at most t_d to exceed a given value, and an upper bound on the maximum value for the same that is possible with a given task set. Finally, we use these to arrive at a tardiness bound. The lemma that follows parallels Lemma 2 and its proof is similar.

Lemma 4 *Let τ be a concrete sporadic task system and let Ψ denote the set of all jobs with deadlines at most t_d of tasks in τ. Let $\mathcal{B}$ and* B *be as defined in Def. 5. Let $U_{sum} \leq m$ and let the tardiness of every job of T_k with deadline less than t_d be at most $x + e_k$, where $x \geq 0$, for all $1 \leq k \leq n$ in an* NP-EDF *schedule $\mathcal{S}$ for τ on m processors. Let $\mathsf{LAG}(\Psi, t_d, \mathcal{S}) + \mathsf{B}(\tau, \Psi, t_d, \mathcal{S}) \leq mx + e_i$, where $t_d = d_{i,j}$, for some $1 \leq i \leq n$. Then, $tardiness(T_{i,j}, \mathcal{S}) \leq x + e_i$.*

An upper bound on $\mathsf{LAG}(\Psi, t', \mathcal{S}) + \mathsf{B}(\tau, \Psi, t', \mathcal{S})$, where t' is the end of a maximally non-busy interval, is given by the next lemma. Its proof is only slightly different from that of Lemma 3, and is available in an appendix.

Lemma 5 *Let τ, Ψ, and $\mathcal{B}$ be as defined in Lemma 4. Let $\mathcal{S}$ be an* NP-EDF *schedule for τ on m processors and 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 of T_i with a deadline less than t_d be at most $x + e_i$, where $x \geq 0$ for all $1 \leq i \leq n$. Then, $\mathsf{LAG}(\Psi, t', \mathcal{S}) + \mathsf{B}(\tau, \Psi, t', \mathcal{S})$ is at most $x \cdot \sum_{T_i \in \mathcal{U}_{\max}(m-1)} u_i + \sum_{T_i \in \mathcal{E}_{\max}(m)} e_i$.*

Lemmas 4 and 5 can be used to establish the following.

Theorem 2 NP-EDF *ensures a tardiness of at most $\frac{(\sum_{T_i \in \mathcal{E}_{\max}(m)} e_i) - e_{\min}}{m - \sum_{T_i \in \mathcal{U}_{\max}(m-1)} u_i} + e_k$ on m processors for every task T_k of a sporadic task system τ with $U_{sum}(\tau) \leq m$.*

Corollary 2 NP-EDF *ensures a tardiness of at most $x + e_k$ for every task T_k of a sporadic task system τ on m processors, if the sum of the utilizations of the tasks in $\mathcal{U}_{\max}(m-1)$ is at most $\frac{mx - \sum_{T_i \in \mathcal{E}_{\max}(m)} e_i + e_{\min}}{x}$ and $U_{sum}(\tau) \leq m$.*

We denote the tardiness bound given by Theorem 2 as NP-EDF-BASIC. As with EDF, a less-pessimistic bound, denoted NP-EDF-ITER, can be obtained using an iterative procedure, and a more-pessimistic bound, denoted NP-EDF-FAST, can be computed in constant time.

It is known that the tardiness bound for a task system with $U_{sum} = 1$ is $e_{\max}$ under NP-EDF on a uniprocessor. However, the bound obtained using Theorem 2 is $e_{\max} - e_{\min} + e_k$ for T_k, which may be higher than $e_{\max}$. The tight bound of $e_{\max}$ can be shown to hold by treating $m = 1$ as a special case in the proof of Theorem 2.

4 Experimental Evaluation

In this section, we describe the results of two sets of simulation experiments conducted using randomly-generated task sets to evaluate the accuracy of the tardiness bounds for EDF and NP-EDF derived in Sec. 3.

The first set of experiments compares the tardiness bounds given by EDF-BASIC and NP-EDF-BASIC and their fast (EDF-FAST and NP-EDF-FAST) and iterative (EDF-ITER and NP-EDF-ITER) variants for 4, 8, and 16 processors (m). For each m, 10^6 task sets were generated. For each task set, new tasks were added as long as total utilization was less than m. For each task T_i, first its utilization u_i was generated as a uniform random number between $(0, y]$, where y was fixed at 0.1 for the first $100,000$ task sets, and was incremented in steps of 0.1 for every $100,000$ task sets. T_i's execution cost was chosen as a uniform random number in the range $(0, 20]$. For each task set, the maximum tardiness of any task and the average of the maximum tardiness of all tasks as given by the six bounds (three each for EDF and NP-EDF) were computed. The mean maximum tardiness is plotted for $m = 4$ and $m = 8$ in Fig. 5 as a functions of u_{avg} and e_{avg}, where u_{avg} denotes the average of the $m - 2$ highest utilizations and e_{avg} that of the $m - 1$ highest execution costs. (Mean average tardiness is around $e_{avg}/2$ time units less.) The descriptions of the plots can be found in the caption of the figure. The rest of this section discusses the results in some detail.

Comparison of BASIC and FAST. For $m = 4$, the difference between BASIC and FAST is negligible for EDF, but is quite considerable for NP-EDF at high values of u_{avg} and e_{avg}. This is due to an additional $e_{\max}$ term in the numerator and a negative $u_{\max}$ term in the denominator of NP-EDF-FAST. The difference widens with m for both EDF and NP-EDF for high u_{avg}. For $m = 8$, NP-EDF-FAST is almost twice as much as NP-EDF-BASIC. For EDF, the difference seems to be tolerable at $m = 8$, but EDF-FAST increases to close to 100% higher than EDF-BASIC for $m = 16$ (not shown here). Overall, FAST appears to yield good results for small m and small u_{avg} or e_{avg}.

Comparison of BASIC and ITER. The difference between ITER and BASIC is almost the same as that between BASIC and FAST for EDF for $m = 4$, but is lower for NP-EDF. The difference increases with increasing m for both EDF and NP-EDF. This is because there is not much increase in ITER with increasing m.

Comparison of EDF and NP-EDF. While there is a large difference between the FAST versions of EDF and NP-EDF, which widens with increasing m, the difference between EDF-ITER and NP-EDF-ITER is much less and

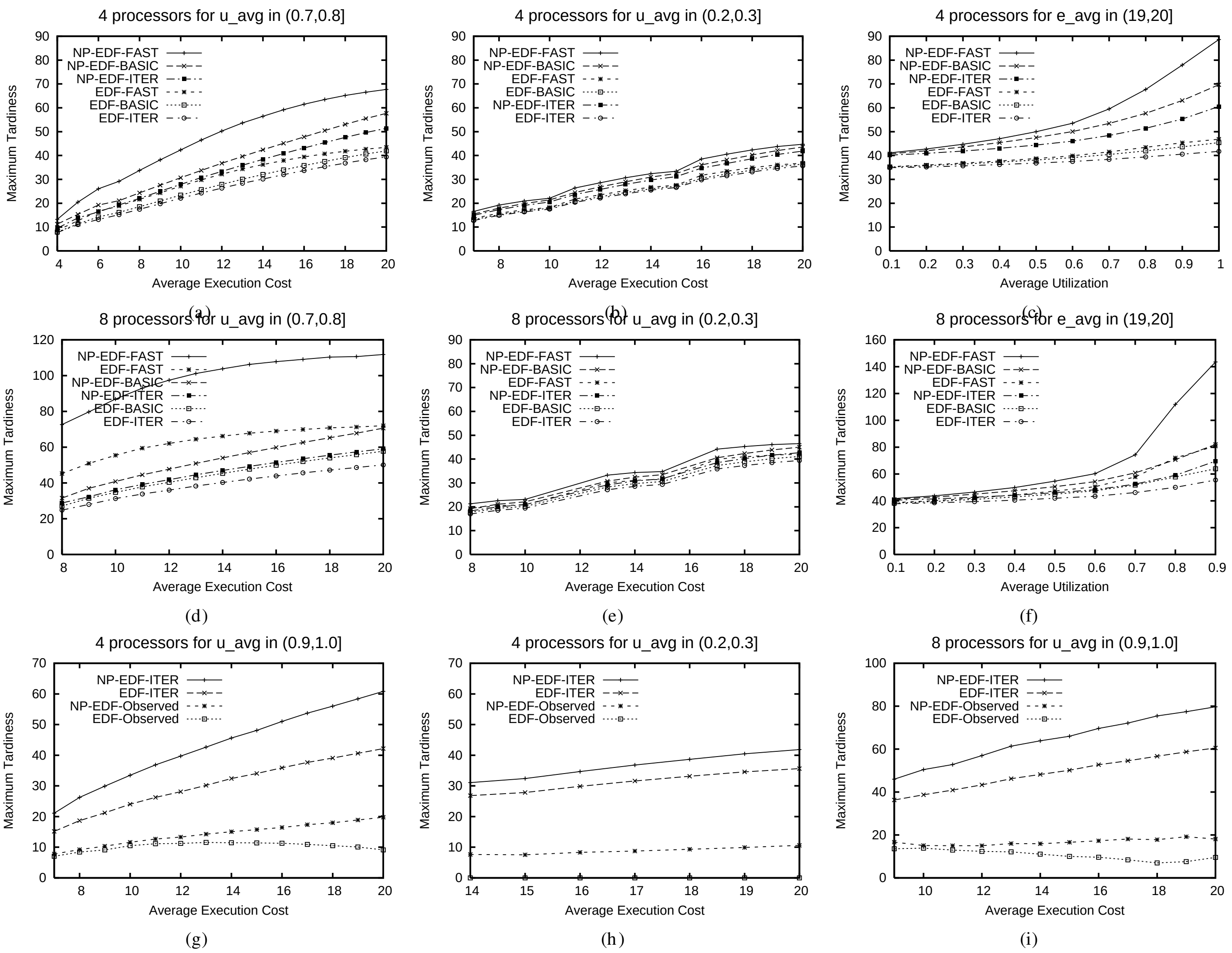

Figure 5: Comparison of the three bounds derived for EDF and NP-EDF for **(a)–(c)** $m = 4$ and **(d)–(f)** $m = 8$. Tardiness bounds as a function of e_{avg} for **(a) & (d)** $0.7 < u_{avg} \leq 0.8$, and for **(b) & (e)** $0.2 < u_{avg} \leq 0.3$, and as a function of u_{avg} for **(c) & (f)** $19 < e_{avg} \leq 20$. Comparison of EDF-ITER and NP-EDF-ITER to tardiness observed in actual EDF and NP-EDF schedules for **(g) & (h)** $m = 4$, and for **(i)** $m = 8$. (The order of the legends and curves coincide in all the graphs.)

narrows with increasing m. The peak difference between the ITER versions, which is less than 20 time units, occurs for $m = 4$, $0.9 < u_{avg} \leq 1.0$, and $19 < e_{avg} \leq 20$ (see inset (c)). At low utilizations, the difference is around five time units for $m = 4$ and three time units for $m = 8$. The difference between the BASIC versions is also not much and decreases with m. Furthermore, these results assume the same worst-case execution costs for both EDF and NP-EDF, whereas in practice the estimates for NP-EDF will be lower due to the absence of job preemptions and migrations. This should further close the gap between the two.

Though not shown here, we also plotted the tardiness bounds by u_{avg} for low and medium values of e_{avg}, and found that in comparison to the plots in insets (c) and (f), all the bounds are proportionately reduced.

Comparison of ITER to actual tardiness. The experiments in the second set compare the bounds estimated by EDF-ITER and NP-EDF-ITER, the best bounds derived, to actual tardiness observed under EDF and NP-EDF, respectively. In this case, 100,000 task sets were generated for each m. For each task set, the tardiness bounds given by EDF-ITER and NP-EDF-ITER were computed. Also, an EDF and an NP-EDF schedule were generated for each task set for 20,000 and 50,000 time units, respectively, and the maximum tardiness observed in each schedule was noted. (Though we have found tardiness to increase even after these time limits, we were forced to constrain the experiments due to restrictions on available time.) Plots of the average of the estimated and observed values for task sets grouped by e_{avg} and u_{avg} are shown in insets (g)–(i) of Fig. 5. For medium values of e_{avg}, the estimates are twice as much as the observed values, with the difference increasing with increasing execution costs. It is somewhat surprising that actual tardiness does not increase much with increasing e_{avg} and that it decreases in some cases.

5 Conclusion

We have derived tardiness bounds under preemptive and non-preemptive global EDF for sporadic real-time task systems on multiprocessors, when the total utilization of a task system is not restricted, but may equal the number of processors, m. These results should help in improving the effective system utilization when soft real-time tasks that can tolerate bounded deadline misses but require guarantees on the long-run fraction of the processing time allocated are multiplexed on a multiprocessor .

Our task model can alternatively be viewed as one in which the relative deadline of each task is greater than its period by an amount that is dependent on the parameters of the task system. Our conditions that check if a tardiness bound can be guaranteed can then be used as schedulability tests for such task systems. While it is possible to extend the EDF schedulability tests derived in prior research discussed in the introduction to task systems with relative deadlines greater than periods, it does not seem likely that such extensions would allow total utilization to equal m even when per-task utilizations are low and relative deadlines are large.

One limitation of our approach is that the tardiness bound that can be guaranteed to each task is fixed and is dependent on task parameters. The problem of guaranteeing arbitrary and different tardiness bounds to different tasks remains to be explored.

References

[1] B. Andersson, S. Baruah, and J. Jonsson. Static priority scheduling on multiprocessors. In *Proceedings of the 22nd Real-Time Systems Symposium*, pages 193–202, December 2001.

[2] T. P. Baker. Multiprocessor EDF and deadline monotonic schedulability analysis. In *Proceedings of the 24th IEEE Real-Time Systems Symposium*, pages 120–129, December 2003.

[3] S. Baruah. The non-preemptive scheduling of periodic tasks upon multiprocessors. Technical Report TR02-025, Department of Computer Science, The University of North Carolina, 2002. To appear in *Real-Time Systems*.

[4] S. Baruah. Optimal utilization bounds for the fixed-priority scheduling of periodic task systems on identical multiprocessors. *IEEE Transactions on Computers*, 53(6):781–784, June 2004.

[5] S. Baruah, N. Cohen, C.G. Plaxton, and D. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15(6):600–625, June 1996.

[6] M. Bertogna, M. Cirinei, and G. Lipari. Improved schedulability analysis of EDF on multiprocessor platforms. In *Proceedings of the 17th Euromicro Conference on Real-Time Systems*, pages 209–218, July 2005.

[7] 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.

[8] S. K. Dhall and C. L. Liu. On a real-time scheduling problem. *Operations Research*, 26(1):127–140, 1978.

[9] J. Goossens, S. Funk, and S. Baruah. Priority-driven scheduling of periodic task systems on multiprocessors. *Real-Time Systems*, 25(2-3):187–205, 2003.

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

[11] C. A. Phillips, C. Stein, E. Torng, and J. Wein. Optimal time-critical scheduling via resource augmentation. *Algorithmica*, 32:163–200, 2002.

[12] L. Sha, T. Abdelzaher, K.-E. Arzen, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A.K. Mok. Real time scheduling theory: A historical perspective. *Real-Time Systems*, 28(2/3):101–155, November/December 2004.

[13] A. Srinivasan and J. Anderson. Efficient scheduling of soft real-time applications on multiprocessors. In *Proceedings of the 15th Euromicro Conference on Real-Time Systems*, pages 51–59, July 2003.

[14] A. Srinivasan and S. Baruah. Deadline-based scheduling of periodic task systems on multiprocessors. *Information Processing Letters*, 84(2):93–98, 2002.

A Appendix — Proof of Lemma 5

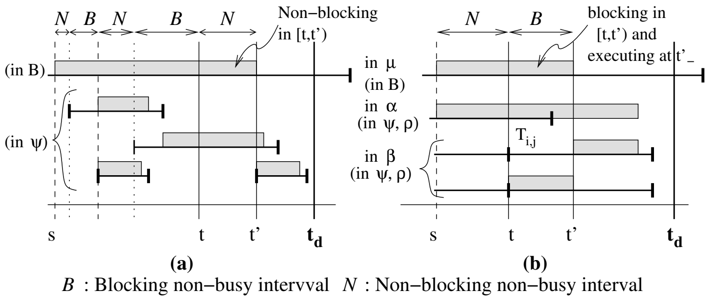

Figure 6: Lemma 5. **(a)** CASE A. **(b)** CASE B.

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. 6(a). Let t be the earliest instant at or after s such that every instant in $[t, t')$ is non-blocking. Hence, because $[s, t')$ is maximally non-busy, at each 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 the 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 LAG of Ψ at t', the jobs of $\mathcal{B}$ that are executing in $[t, t')$ can be ignored and the intervals in which they are executing can be taken to be idle intervals on the respective processors. Therefore, the LAG of Ψ at t' can be determined in the same manner as that used in the case of preemptive EDF in Lemma 3 to be at most

$$x \cdot \sum_{T_i \in \mathcal{U}_{\max}(\tau_\Psi, k)} u_i + \sum_{T_i \in \mathcal{E}_{\max}(\tau_\Psi, k)} e_i, \tag{13}$$

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

We next determine a bound on $\mathsf{B}(\tau, \Psi, t', \mathcal{S})$. If $t' < t_d$, then by the statement of the lemma, t' is busy. Therefore, no job of $\mathcal{B}$ that is executing at t'^{-} executes at t' or later. Hence, in this case $\mathsf{B}(\tau, \Psi, t', \mathcal{S}) = 0$. The other case is that $t' = t_d$ holds. Note that each job $T_{i,j}$ in $\mathcal{B}(\tau, \Psi, t_d, \mathcal{S})$ could execute for at most e_i time units after t_d. Because k jobs 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 $t' = t_d$, $\mathsf{B}(\tau, \Psi, t_d, \mathcal{S}) \le \sum_{T_i \in \mathcal{E}_{\max}((\tau \setminus \tau_\Psi), m-k)} e_i$. Hence, for either case, by

(13), we have $\mathsf{LAG}(\Psi, t', \mathcal{S}) + \mathsf{B}(\tau, \Psi, t', \mathcal{S}) \leq x \cdot \sum_{T_i \in \mathcal{U}_{\max}(\tau, m-1)} u_i + \sum_{T_i \in \mathcal{E}_{\max}(\tau, m)} e_i$.

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 execution at t. Rather, the blocking job should have commenced execution before t. Similarly, since every instant in $[t, t')$ 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 $\hat{t}$, where $t \leq \hat{t} < t'$, 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 τ 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

$$|\mathcal{J}| = |\mu| = b \geq 1. \tag{14}$$

By (12), the LAG of Ψ at t is at most the sum of the lags of all tasks in τ 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,

$$\mathsf{LAG}(\Psi, t, \mathcal{S}) \leq \textstyle\sum_{T_i \in \rho} \mathsf{lag}(T_i, t, \mathcal{S}), \tag{15}$$

Partitioning ρ. Our approach for determining an upper bound on the LAG of Ψ at t' is mostly similar to that used in Lemma 3. Because (15) holds, we first partition the tasks in ρ into subsets α and β, 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'.

$$\alpha \stackrel{\text{def}}{=} \text{subset of all tasks in } \rho \text{ executing at } t^{-}$$
$$\beta \stackrel{\text{def}}{=} \text{subset of all tasks in } \rho \text{ not executing at } t^{-}$$

Upper bound on the 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_i 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 $d_{i,j}$.

Case 1: $d_{i,j} < t$. Because $T_{i,j}$ cannot be preempted, the latest time that $T_{i,j}$ completes executing can be $t + e_i - \delta_i$. (It could complete earlier if its actual execution time is lower than e_i.) By the statement of the lemma, the tardiness of every job of T_i with deadline less that t_d is at most $x + e_i$. Therefore, $d_{i,j} \geq t + (e_i - \delta_i) - (x + e_i) = t - \delta_i - x$ holds. (As in Lemma 3, $d_{i,j} \geq t - \delta_i - x$ holds, even if $T_{i,j}$ completes early, because, the tardiness bound should hold even for the worst case, when $T_{i,j}$ executes for a full e_i time units.) In PS_{Ψ}, $T_{i,j}$ completes execution by $d_{i,j}$ and T_i is allocated a share of u_i in every instant in $[d(T_{i,j}), t)$ in which it is active. Thus, the under-allocation to T_i in $\mathcal{S}$ in $[0, t)$ is at most $e_i - \delta_i + (t - d_{i,j}) \cdot u_i \leq e_i - \delta_i + (x + \delta_i) \cdot u_i$. Hence, $\mathsf{lag}(T_i, t, \mathcal{S}) \leq e_i - \delta_i + (x + \delta_i) \cdot u_i \leq e_i + x \cdot u_i$.

Case 2: $d_{i,j} \geq t$. In this case, the amount of work done by PS_Ψ on $T_{i,j}$ up to time t is given by $e_i - (d_{i,j} - t) \cdot u_i$. Because all prior jobs of T_i have completed execution by t in both $\mathcal{S}$ and PS_Ψ, and $T_{i,j}$ has executed for δ_i time units before t in $\mathcal{S}$, $\mathsf{lag}(T_i, t, \mathcal{S}) = e_i - (d_{i,j} - t) \cdot u_i - \delta_i \leq e_i - \delta_i \leq e_i - \delta_i + (x + \delta_i) \cdot u_i \leq e_i + x \cdot u_i$.

Thus, in both cases, we have

$$\mathsf{lag}(T_i, t, \mathcal{S}) \leq e_i + x \cdot u_i. \tag{16}$$

Upper bound on the lag at t of a task in β. Let T_i be a task in β. 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 job $T_{i,j}$ is in Ψ and is pending at t^-. Then, $d_{i,j} \leq t_d$ holds and because T_i is in β, $T_{i,j}$ is not executing at t^-. Since $[t, t')$ is maximally-blocking, at least one 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. 6(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 PS_Ψ, and hence, the lag of T_i at t is at most zero.

Because the lag of a task in β is at most zero at t, $\sum_{T_i \in \rho} \mathsf{lag}(T_i, t, \mathcal{S}) = \sum_{T_i \in \alpha} \mathsf{lag}(T_i, t, \mathcal{S}) + \sum_{T_i \in \beta} \mathsf{lag}(T_i, t, \mathcal{S}) \leq \sum_{T_i \in \alpha} \mathsf{lag}(T_i, t, \mathcal{S})$. Hence, by (16), $\sum_{T_i \in \rho} \mathsf{lag}(T_i, t, \mathcal{S}) \leq \sum_{T_i \in \alpha} (e_i + x \cdot u_i)$. Therefore, by (15), we have

$$\mathsf{LAG}(\Psi, t, \mathcal{S}) \leq \textstyle\sum_{T_i \in \alpha} (e_i + x \cdot u_i) \tag{17}$$

Since we need to determine an upper bound on the sum of $\mathsf{LAG}(\Psi, t', \mathcal{S})$ and $\mathsf{B}(\tau, \Psi, \mathcal{S}, t')$, we also need to determine an upper bound on $\mathsf{B}(\tau, \Psi, \mathcal{S}, t)$. 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 for jobs in $\mathcal{B}$ (*i.e.*, the blocking work) at any time u in $[t, t')$, $\mathsf{B}(\tau, \Psi, \mathcal{S}, u)$, 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 e_i. Therefore, we have $\mathsf{B}(\tau, \Psi, \mathcal{S}, t) \leq \sum_{T_i \in \mu} e_i$, and hence, by (17), we have

$$\begin{aligned} \mathsf{LAG}(\Psi, t, \mathcal{S}) + \mathsf{B}(\tau, \Psi, t, \mathcal{S}) &\leq \textstyle\sum_{T_i \in \alpha} (e_i + x \cdot u_i) + \sum_{T_i \in \mu} e_i = \sum_{T_i \in \alpha \cup \mu} e_i + \sum_{T_i \in \alpha} x \cdot u_i \\ &\leq \textstyle\sum_{T_i \in \mathcal{E}_{\max}(\tau, m)} e_i + \sum_{T_i \in \mathcal{U}_{\max}(\tau, m-1)} x \cdot u_i, \end{aligned} \tag{18}$$

where the last inequality follows from (14) ($|\mu| = b \geq 1$) and $|\alpha| = m - b$. $|\alpha| = m - b$ holds because every task in μ or α is executing at t^-.

Finally, we are left with determining an upper bound on the sum of LAG and B at t'. Let $X \leq \mathsf{B}(\tau, \Psi, t, \mathcal{S})$ 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 PS_Ψ, jobs in Ψ could execute for at most $m \cdot (t' - t)$ time, *i.e.*, $\mathsf{A}(PS_\Psi, \Psi, t, t') \leq m \cdot (t' - t)$. Therefore, $\mathsf{LAG}(\Psi, t', \mathcal{S}) = \mathsf{LAG}(\Psi, t, \mathcal{S}) + \mathsf{A}(\mathsf{PS}_\Psi, \Psi, t, t') - \mathsf{A}(\mathcal{S}, \Psi, t, t') \leq \mathsf{LAG}(\Psi, t, \mathcal{S}) + X$. However, since jobs in $\mathcal{J}$ execute for a total time of X in $[t, t')$, the pending work for jobs in $\mathcal{J}$, and hence, those in $\mathcal{B}$ at t', $\mathsf{B}(\tau, \Psi, t', \mathcal{S})$, is at most $\mathsf{B}(\tau, \Psi, t, \mathcal{S}) - X$. Thus, $\mathsf{LAG}(\Psi, t', \mathcal{S}) + \mathsf{B}(\tau, \Psi, t', \mathcal{S}) \leq \mathsf{LAG}(\Psi, t, \mathcal{S}) + \mathsf{B}(\tau, \Psi, t, \mathcal{S})$, which by (18), is at most $\sum_{T_i \in \mathcal{E}_{\max}(\tau, m)} e_i + x \cdot \sum_{T_i \in \mathcal{U}_{\max}(\tau, m-1)} u_i$. ■