Generalized Tardiness Bounds for Global Multiprocessor Scheduling

Hennadiy Leontyev
James H. Anderson
Department of Computer Science, The University of North Carolina at Chapel Hill

Abstract

We consider the issue of deadline tardiness under global multiprocessor scheduling algorithms. We present a general tardiness-bound derivation that is applicable to a wide variety of such algorithms (including some whose tardiness behavior has not been analyzed before). Our derivation is very general: job priorities may change rather arbitrarily at runtime, capacity restrictions may exist on certain processors, and, under certain conditions, non-preemptive regions are allowed. Our results show that, with the exception of static-priority algorithms, most global algorithms considered previously have bounded tardiness. In addition, our results provide a simple means for checking whether tardiness is bounded under newly-developed algorithms.

1 Introduction

Most major chip manufacturers are investing in multicore technologies to continue performance improvements in their product lines in the face of fundamental limitations of single-core chip designs. To date, several manufacturers have released dual-core chips, Intel and AMD each have quad-core chips on the market, and Sun's Niagara and more recent Niagara 2 systems have eight-core chips with multiple hardware threads per core. In the future, per-chip core counts are expected to increase significantly. Indeed, Intel has announced plans to release chips with as many as 80 cores within five years (Farivar, 2006).

The advent of multicore technologies is a profound development that is impacting software design processes across a wide range of application domains. An important category of such applications is those with *real-time constraints*. The range of real-time applications already being deployed on multicore platforms is quite varied, ranging from simple streaming applications to computationally-intensive applications for which multiprocessor designs are a *necessity*. A good example of the latter is Azul System's *Vega2* system, which is a Java-based appliance with up to 768 cores (on several chips) for processing time-sensitive business transactions (Bisson, 2006).

To support such applications on either a multicore or a conventional (non-multicore) multiprocessor platform, an appropriate multiprocessor scheduling algorithm must be used. This paper is directed at issues concerning such algorithms. Our specific focus is multiprocessor algorithms for scheduling soft real-time workloads specified as sporadic tasks.

In the sporadic task model, tasks repeatedly generate sequential *jobs* subject to deadlines. A sporadic task has a specified *period*, which defines the minimum spacing between its jobs, and a *relative deadline*, which defines the length of the time interval in which each of its jobs is allowed to complete. Job deadlines can be either *hard* — in which case they should always be met — or *soft* — in which case misses can occur, provided the extent of violation is constrained in some way. The soft real-time constraint considered in this paper requires that deadline tardiness be bounded; if a job misses its deadline, then its *tardiness* is defined as the difference between its completion time and deadline (see Section 2).

Bounded tardiness is a sufficient property in many soft real-time applications (provided the bounds are not too large). In particular, such bounds ensure that the long-term processor share of each task is in accordance with its specified utilization. For example, in a video decoding application, it is desirable to decode a frame every 33.3 ms in order to achieve a rate of 30 frames per second. However, a tardiness of a few milliseconds will not compromise the video quality if the decoding rate remains at 30 frames per second over reasonably long intervals of time.

In devising multiprocessor scheduling algorithms, two basic approaches exist: partitioning and global scheduling. Under partitioning, tasks are statically assigned to processors, and each processor schedules its assigned tasks using a uniprocessor scheduling algorithm. Under global scheduling, tasks are scheduled from a single run queue and may migrate among processors. For soft real-time systems, global algorithms have the advantage of being able to ensure bounded deadline tardiness, as long as the available processing capacity is not exceeded (something we assume throughout this paper). The same is not true of partitioned scheduling. For example, no partitioning scheme can schedule three tasks with utilization $2/3$ each on two processors without overloading a processor, and on an overloaded processor, tardiness will grow unboundedly. On the the other hand, if a task system can be partitioned without overloading a processor, then bounded tardiness can be ensured, provided an appropriate uniprocessor scheduler is used on each processor (in particular, a scheduler that uses window-constrained prioritizations, as discussed later in this section).

The main focus of this paper is global algorithms that are capable of ensuring bounded tardiness (without restrictions on overall utilization).

Motivation and prior work. The first tardiness bounds to be established for a global scheduling algorithm pertained to the earliest-pseudo-deadline-first (EPDF) Pfair algorithm (Devi and Anderson, 2004). This analysis was later extended to establish tardiness bounds for several variants of the global earliest-deadline-first (EDF) algorithm, wherein jobs with earlier deadlines have higher priority. These include preemptive and non-preemptive EDF and two variants that slightly alter EDF prioritizations and allow a small number of special tasks to be guaranteed lower tardiness (Devi and Anderson, 2006) or cause temporary overloads (Leontyev and Anderson, 2007b). (The latter variant arises in an approach for scheduling multi-speed multiprocessor systems.) Tardiness bounds have also been established for the global first-in first-out (FIFO) algorithm (Leontyev and Anderson, 2007c), wherein jobs with earlier release times have higher priority. Given that tardiness is bounded under such disparate algorithms, several questions come to mind. Do other widely-studied global algorithms have bounded tardiness? Is there a singular characteristic of such algorithms that results in bounded tardiness? Can the class of algorithms for which tardiness is bounded be generally characterized?

Contributions. In this paper, we present a generalized tardiness result that answers these questions. This result implies that the singular characteristic needed for tardiness to be bounded is that a pending job's priority eventually (in bounded time) is higher than that of any future job. Global algorithms that do *not* have this characteristic (and for which tardiness can be unbounded) include static-priority algorithms such as the rate-monotonic (RM) algorithm, and impractical dynamic-priority algorithms such as the earliest-deadline-*last* algorithm, wherein jobs with earlier deadlines have *lower* priority. (This algorithm is different from the earliest-deadline-late (EDL) scheduler from the literature (Chetto and Chetto, 1989).) Global algorithms that *do* have this property include the EDF, FIFO, EDF-until-zero-laxity (EDZL), and least-laxity-first (LLF) algorithms. (EDZL and LLF are described later.)

We establish a generalized tardiness result by considering a generic scheduling algorithm where job priorities are defined by points in time that may vary as time progresses. All of the algorithms mentioned above can be seen as special cases of this generic algorithm in which priorities are further constrained. Even the PD^2 Pfair algorithm (Anderson and Srinivasan, 2004), which uses a rather complex notion of priority, is a special case. The main result of this paper is a derivation of a tardiness bound that applies to the generic algorithm if priorities are *window-constrained*: a job's priority at any time is a point in time within a time window that includes the job's release time and deadline. For example, under EDF, this time point is simply the job's deadline. We also show that if this window constraint is violated, then tardiness can be unbounded. It is possible to define window-constrained prioritizations not only for EDF, but also for FIFO, EDZL, LLF, EPDF, and PD^2, so these algorithms have bounded tardiness. (For EDF, FIFO, EPDF, and PD^2, this was previously known.) For any other algorithm that may be devised in the future, our results enable tardiness bounds to be established by simply showing that prioritizations can be expressed in a window-constrained way (instead of laboriously devising a new proof).

The notion of a window-constrained priority is very general. For example, it is possible to describe hybrid scheduling policies by combining different prioritizations, *e.g.*, using a combination of EDF and FIFO in the same system. Priority rules can even change dynamically (subject to the window constraint). For example, if a task has missed too many deadlines, then its job priorities can be boosted for some time so that it receives special treatment. Or, if a single job is in danger of being tardy, then its prioritization may be changed so that it completes execution non-preemptively (provided certain restrictions hold — see Section 5.5). Tardiness also remains bounded if early-release behavior is allowed or if the capacity of each processor that is available to the (soft) real-time workload is restricted. In simplest terms, the main message of this paper is that, *for global scheduling algorithms, bounded tardiness is the common case, rather than the exception* (at least, ignoring clearly impractical algorithms such as earliest-deadline-last). For the widely-studied EDZL and LLF algorithms, and for several of the variants of existing algorithms just discussed, this paper is the first to show that tardiness is bounded. Although we assume in our analysis that relative deadlines are equal to periods, the analysis can also be applied to task systems where relative deadlines differ from periods as discussed in Section 5.1.

The rest of this paper is organized as follows. In Sections 2–3, we present our task model and scheduling framework. Then, in Section 4, we present our tardiness-bound derivation. In Section 5, we discuss some special cases and possible extensions to the analysis. As discussed later, tardiness may be different under different scheduling algorithms. In Section 6, we present results from experiments conducted to assess such differences. Section 7 concludes the paper.

2 System Model

We consider the problem of scheduling on m processors a set of sporadic soft real-time tasks $\tau = \{T_1, \ldots, T_n\}$. All time quantities considered in this paper are assumed to be real numbers.

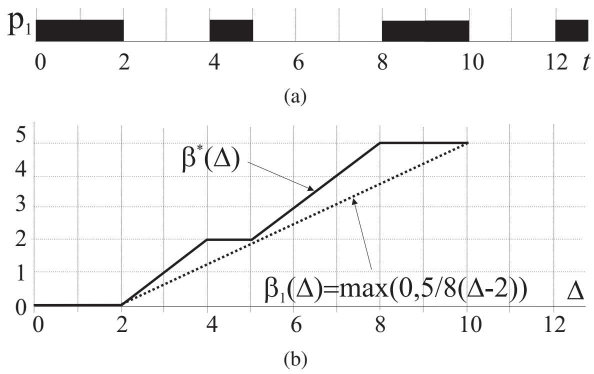


Figure 1: **(a)** Unavailable time instants and **(b)** service functions for processor 1 (denoted P_1) in Example 1.

2.1 Processor Model

Our main result is very general and can be applied in settings where the full capacity of one or more processors is not available for the soft real-time workload (Brandenburg and Anderson, 2007; Devi and Anderson, 2006; Leontyev and Anderson, 2007b). We assume that such capacity restrictions are specified using *service functions* (Chakraborty et al, 2003). Specifically, the minimum guaranteed time that processor k can provide to the tasks in τ in any time interval of length $\Delta \geq 0$ is characterized by the service function

$$\beta_k(\Delta) = \max(0, \widehat{u_k} \cdot (\Delta - \sigma_k)), \tag{1}$$

where $\widehat{u_k} \in (0, 1]$ and $\sigma_k \geq 0$. In the above definition, $\widehat{u_k}$ is the total long-term utilization available to the tasks in τ on processor k and σ_k is the maximum length of time when the processor can be unavailable over an interval of length Δ. We require $\beta_k(\Delta)$ and σ_k to be specified for each k. Note that, if (unit-speed) processor k is fully available to the tasks in τ, then $\beta_k(\Delta) = \Delta$.

Limited processor availability arises in many contexts. For example, real-time tasks may be partially deprived of processing capacity due to interrupt service routines (Jeffay and Stone, 1993) or in hierarchically scheduled systems (Shin et al, 2008). Allowing processors to have limited availability enables our results to be applied in these and other contexts in a uniform fashion.

Example 1. Consider a system with one processor ($m = 1$) that is not fully available for the soft real-time workload. The availability pattern, which repeats every eight time units, is shown in Figure 1(a); intervals of unavailability are shown as shaded regions. For processor 1, the minimum amount of time that is guaranteed to soft real-time tasks over any interval of length Δ is zero if $\Delta \leq 2$, $\Delta - 2$ if $2 \leq \Delta \leq 4$, and so on. Figure 1(b) shows the minimum amount of time $\beta^*(\Delta)$ that is available on processor 1 for soft real-time tasks over any interval $[t, t + \Delta]$. It also shows a service curve $\beta_1(\Delta) = max(0, \widehat{u_1}(\Delta - \sigma_1))$, where $\widehat{u_1} = \frac{5}{8}$ and $\sigma_1 = 2$, which bounds $\beta^*(\Delta)$ from below. $\beta_1(\Delta)$ can be used to reflect the minimum service guarantee for soft real-time tasks on processor 1.

2.2 Task Model

Using the available processor time, the sporadic tasks $T_1, \ldots, T_n$ in τ are scheduled. Each task is invoked or *released* repeatedly, with each such invocation called a *job*.

Associated with each task T_i are two parameters, e_i and p_i: e_i gives the maximum *execution time* of one job of T_i, while, p_i, called the *period* of T_i, is the minimum time between consecutive job releases. For brevity, we often use the notation $T_i = (e_i, p_i)$ to specify task parameters.

The j^{th} job of T_i, where $j \geq 1$, is denoted $T_{i,j}$. A task's first job may be released at any time $t \geq 0$. The actual execution requirement of job $T_{i,j}$ is $e_{i,j} \leq e_i$. The release time of job $T_{i,j}$ is denoted $r_{i,j}$ and its (absolute) deadline $d_{i,j}$ is defined as $r_{i,j} + p_i$. If $T_{i,j}$ completes at time t, then its *tardiness* is $\max(0, t - d_{i,j})$. A *task*'s tardiness is the maximum of the tardiness of any of its jobs. When a job of a task misses its deadline, the release time of the next job of that task is not altered. However, at most one job of a task may execute at any time, even if deadlines are missed.

The *utilization of task* T_i is defined as $u_i = e_i/p_i$, and the *utilization of the task system* τ as $U_{sum} = \sum_{T_i \in \tau} u_i$. We assume

$$U_{sum} \leq \sum_{k=1}^{m} \widehat{u_k}. \tag{2}$$

Otherwise, tardiness may grow unboundedly (Devi, 2006).

For each job $T_{i,j}$, we define an *eligibility time* $\epsilon_{i,j}$ such that $\epsilon_{i,j} \leq r_{i,j}$ and $\epsilon_{i,j-1} \leq \epsilon_{i,j}$. The eligibility time of $T_{i,j}$ denotes the earliest time when it may be scheduled. A job $T_{i,j}$ is said to be *early-released* if $\epsilon_{i,j} < r_{i,j}$. Job $T_{i,j}$ is said to be *eligible* at time t if $\epsilon_{i,j} \leq t$. The early-release task model was first proposed in work on Pfair scheduling (Anderson and Srinivasan, 2004). Allowing early releases can reduce job response times as the following example illustrates.

Example 2. Consider the (Earliest Pseudo-Deadline First) EPDF Pfair algorithm (Devi and Anderson, 2004). Under it, task periods and execution times are assumed to be integral, and each task T_i is represented by a sequence of unit-length schedulable entities called *subtasks*, denoted T_i^j, where $j \geq 1$. Each subtask T_i^j has two attributes associated with it, a *release time* r_i^j and a *deadline* d_i^j. The interval $[r_i^j, d_i^j)$ is called the *window* of T_i^j. [1] Subtask T_i^j becomes available for execution at time r_i^j and has higher priority than subtask T_x^y if $d_i^j < d_x^y$. Deadline ties are resolved arbitrarily but consistently. In considering EPDF scheduling examples, we assume (for simplicity) that jobs are released in a synchronous periodic fashion, in which case $r_i^j = \lfloor \frac{i-1}{u_i} \rfloor$ and $d_i^j = \lceil \frac{i}{u_i} \rceil$ (see (Anderson and Srinivasan, 2004)). Figure 2 shows two EPDF schedules of a task $T_1 = (3, 8)$. Inset (a) shows a schedule in which early releases are not allowed. In this schedule, each subtask executes within its respective window which is shown in bold. The time between $T_{1,1}$'s release and completion is six time units. A schedule in which early releases are allowed is shown in Figure 2(b). In this schedule, each subtask commences execution immediately after its predecessor completes. In this schedule, the response time of $T_{1,1}$ is three time units.

We assume that eligible jobs are placed into a single global ready queue. When choosing a new job to schedule, the scheduler selects (and dequeues) the ready job of highest priority. As reiterated in Definition 3 in Section 4, a job is *ready* if it is eligible and its predecessor (if any) has completed execution. Job priorities are determined as follows.

[1] This usage of term "window" should not be confused with that arising in our window constraints.

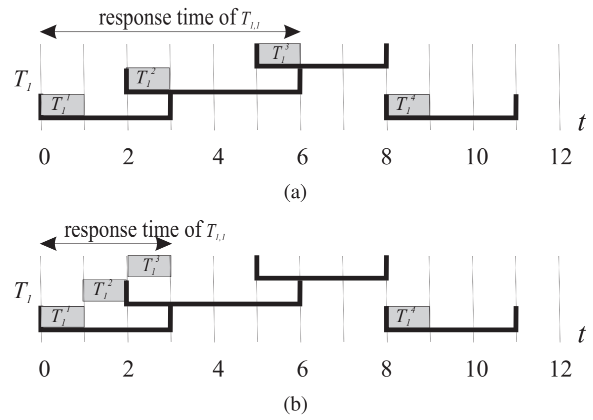


Figure 2: EPDF schedules from Example 2 **(a)** without and **(b)** with early releases.

Definition 1. (prioritization functions) Associated with each job $T_{i,j}$ is a function of time $\chi(T_{i,j}, t)$ defined for $t \geq 0$ and called its *prioritization function*. If $\chi(T_{i,j}, t) < \chi(T_{k,h}, t)$, then the priority of $T_{i,j}$ is higher than the priority of $T_{k,h}$ at time t. We assume that, when comparing priorities, any ties are broken arbitrarily but consistently. That is, if, $\chi(T_{i,j}, t) = \chi(T_{k,h}, t)$ and $\chi(T_{i,j}, t') = \chi(T_{k,h}, t')$, where $t \neq t'$, then the tie is broken in favor of $T_{i,j}$ at time t iff it is broken in favor of $T_{i,j}$ at time t'.

3 Example Mappings

We now show how to describe several well-known scheduling policies in our framework, using the two-processor task set
$\tau = \{T_1 = (1,3), T_2 = (2,3), T_3 = (1,4), T_4 = (3,4)\}$ executing on two fully-available processors as an example. Unless stated otherwise, we assume $e_{i,j} = e_i$ and $\epsilon_{i,j} = r_{i,j}$ in these examples, for each job $T_{i,j}$. In depicting example schedules, we use up (down) arrows to depict job releases (deadlines).

Example 3. Figure 3(a) shows a schedule for τ under the global EDF algorithm. In this case, since jobs are prioritized by deadline, it suffices to define $\chi(T_{i,j}, t) = d_{i,j}$ for each $T_{i,j}$. In Figure 3(a), the value of $\chi(T_{i,j}, t)$ is shown for each job $T_{i,j}$ using a black circle labeled $\chi_{i,j}$.

Example 4. Figure 3(b) shows a schedule for τ under the global RM algorithm. In this case, $T_{i,j}$ should have priority over $T_{k,h}$ if $i < k$ (since the tasks in τ are ordered by increasing periods). Thus, we can simply define $\chi(T_{i,j}, t) = i$ for each job $T_{i,j}$, as shown.

Example 5. Figure 3(c) shows a schedule for τ under the global FIFO algorithm (which, by definition, schedules jobs non-preemptively). In this case (assuming no early releases), it suffices to define $\chi_{i,j}(t) = r_{i,j}$ for each job $T_{i,j}$, as shown. (Note that, if early releases are allowed, then this prioritization may not reflect the actual job arrival order.)

Example 6. Interestingly, the definition of $\chi(T_{i,j}, t)$ is flexible enough to allow *combinations* of scheduling policies to be specified. For example, we can prioritize the jobs of $T_1, \ldots, T_3$ on an

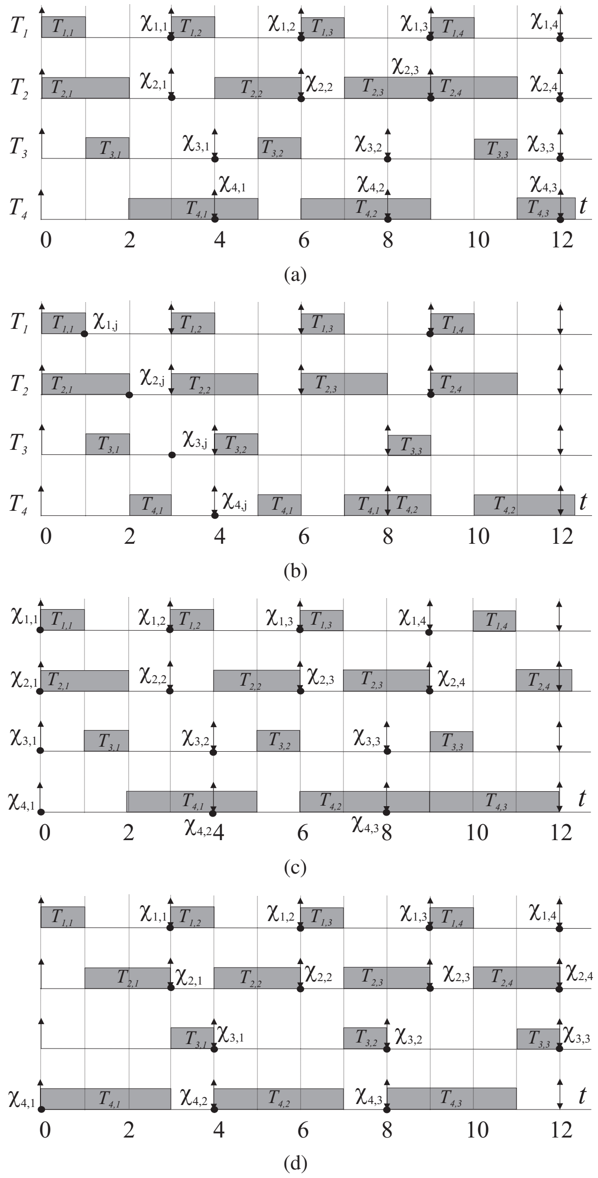


Figure 3: **(a)** Example 3 (global EDF). **(b)** Example 4 (global RM). **(c)** Example 5 (global FIFO). **(d)** Example 6 (hybrid global scheduler).

EDF basis and those of T_4 on a FIFO basis by defining $\chi(T_{i,j}, t) = d_{i,j}$ for $1 \le i \le 3$, and $\chi(T_{4,j}, t) = r_{4,j}$. A schedule for this hybrid policy is shown in Figure 3(d). It is also possible to mix RM and EDF prioritizations (even though such a scheme would not have window-constrained priorities). For example, if task T_1 needs to be statically prioritized over all other tasks, then we can

Table 1: χ-values in Example 7.

Time t	$\chi(T_{1,j}, t)$	$\chi(T_{2,j}, t)$	$\chi(T_{3,j}, t)$	$\chi(T_{4,j}, t)$
0	2	1	3	1
1	2	2	3	2
2	2	–	3	3
3	5	4	3	–
4	5	5	7	5
5	5	–	7	6
6	8	7	7	7
7	8	8	7	–
8	8	–	11	9
9	11	10	11	10
10	11	11	11	11
11	11	–	11	–

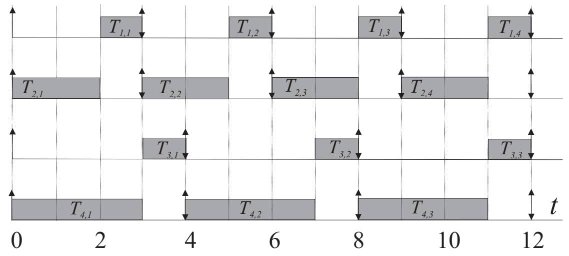


Figure 4: Example 7 (global preemptive LLF).

set $\chi(T_{1,j}, t) = -1$ for all jobs of T_1 and $\chi(T_{i,j}, t) = d_{i,j}$ for all jobs of other tasks.

Example 7. So far we have considered only fixed job-priority algorithms, wherein the priority $\chi(T_{i,j}, t)$ is constant during job $T_{i,j}$'s execution. We now consider a slightly more complicated example, namely the global LLF scheduling algorithm (Liu, 2000). The *laxity* or *slack* of a job $T_{i,j}$ at time t is defined as

$$slack_{i,j}(t) = d_{i,j} - t - (e_i - \delta_{i,j}(t)), \tag{3}$$

where $\delta_{i,j}(t)$ is the amount of time for which $T_{i,j}$ has executed before t. If a job does not miss its deadline, then its slack is always non-negative; if it does miss its deadline, then its slack becomes negative at some time prior to its deadline. According to LLF, $T_{i,j}$ has higher priority than $T_{k,h}$ at time t if $slack_{i,j}(t) < slack_{k,h}(t)$. To capture this, we can simply define $\chi(T_{i,j}, t) = d_{i,j} - (e_i - \delta_{i,j}(t))$ for each job $T_{i,j}$. Because this definition depends on $\delta_{i,j}(t)$, $\chi(T_{i,j}, t)$ is not constant, as in the prior examples, but is time-dependent. Assuming that it is updated only at integral points in time, $\chi(T_{i,j}, t+1) := \chi(T_{i,j}, t) + 1$, if $T_{i,j}$ executes during the interval $[t, t+1)$, and $\chi(T_{i,j}, t+1) := \chi(T_{i,j}, t)$, otherwise.

Figure 4 shows an LLF schedule for τ where ties are broken in favor of jobs currently executing. Because χ-values change with time, they are not shown in the schedule, as earlier, but are depicted separately in Table 1. The table shows the value of $\chi(T_{i,j}, t)$ for the earliest pending job $T_{i,j}$ of each task T_i where $0 \le t \le 11$.

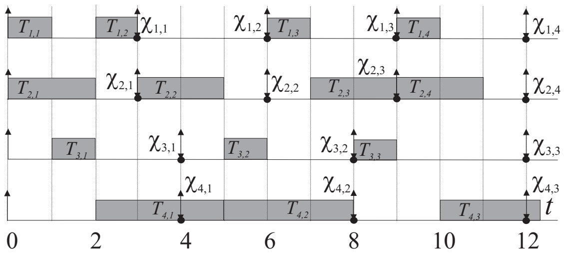


Figure 5: Early releasing under global EDF.

Example 8. The EDZL algorithm (Piao et al, 2006), which is a hybrid of EDF and LLF, can be specified as well. In this case, $\chi(T_{i,j}, t)$ is set to $d_{i,j}$ (as in EDF) when $T_{i,j}$ is released, and is reset to $d_{i,j} - (e_i - \delta_{i,j}(t)) \leq d_{i,j}$ (as in LLF) when $T_{i,j}$'s slack becomes zero, where $\delta_{i,j}(t)$ is as defined earlier. To our knowledge, EDZL has not been considered previously in systems where deadlines can be missed. However, if no deadlines are missed, then our definition yields priority comparisons that match exactly how EDZL has been specified in prior work. It is possible that other variants could be defined that prioritize jobs differently when deadlines are missed.

In the examples above, we assumed $\epsilon_{i,j} = r_{i,j}$, i.e., jobs are not released early. The idea of early releasing is illustrated in Figure 5, which shows a global EDF schedule for the task set in Example 3 with early releases allowed. In particular, for job $T_{1,2}$, $\epsilon_{1,2} = 1$. Thus, $T_{1,2}$ begins its execution one time unit earlier than its actual release time. Note also that jobs $T_{4,1}$ and $T_{4,3}$ miss their deadlines by one time unit.

Example 9. The PD^2 (Anderson and Srinivasan, 2004) and EPDF (Devi and Anderson, 2004) Pfair algorithms can also be modeled using our framework. Consider the EPDF algorithm introduced in Example 2. Again, we illustrate assuming jobs are released in a synchronous periodic fashion. First, we represent each task $T_i = (e_i, p_i)$ by a task T'_i with $e'_i = 1$ and $p'_i = \frac{1}{u_i}$. The EPDF subtask T_i^j then corresponds to the job $T'_{i,j}$. Second, we define the eligibility time of $T'_{i,j}$ as $\epsilon_{i,j} = r_i^j$. Third, we define the prioritization function for job $T'_{i,j}$ as $\chi(T_{i,j}, t) = d_i^j$. Note, that $\chi(T_{i,j}, t)$ is always an integral number.

This mapping is illustrated in Figure 6 using the task set $\tau = \{T_1 = (3, 8), T_2 = (3, 7), T_3 = (3, 6), T_4 = (1, 2)\}$ scheduled on two fully-available processors. Inset (a) shows an EPDF schedule for τ. Subtask windows are shown in bold. Inset (b) shows a schedule for τ', which is constructed from τ in the way described above. In this figure, the release time of each job $T'_{i,j}$ is denoted by an up arrow and its deadline is denoted by a down arrow. χ-values are depicted as black circles.

PD^2 differs from EPDF in that two special tie-breaking rules are used in the event of a deadline tie. We can capture the effects of these tie breaks by slightly shifting the value of a job's prioritization function and letting it be non-integral.

4 Tardiness Bound

In this section, we show that any scheduling algorithm (specified according to Definition 1) has bounded tardiness if its prioritization functions are "window-constrained," as defined below in Definition 4. This definition imposes two separate constraints on χ-values. We show that if either is

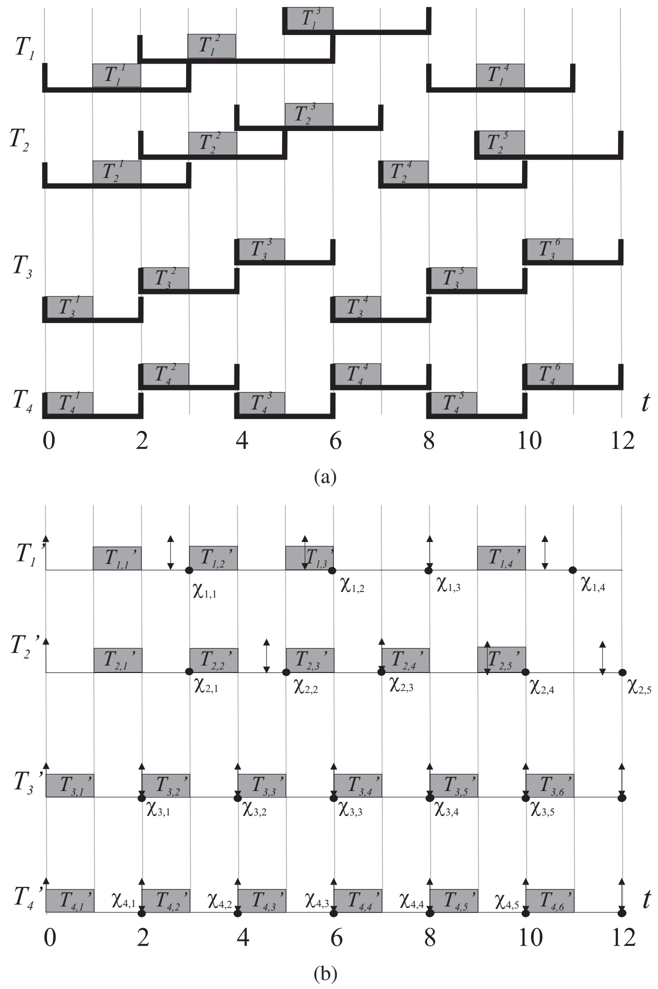


Figure 6: **(a)** An EPDF schedule for the task set τ from Example 9. **(b)** Equivalent schedule obtained using prioritization functions.

violated, then tardiness may become unbounded. In this section, we consider a system with partially available processors; later, in Section 5, we consider the special case when all processors are fully available as well as some other extensions to the analysis.

4.1 Definitions

The system start time is assumed to be zero. For any time $t > 0$, t^- denotes the time $t - v$ in the limit $v \to 0+$.

Definition 2. (pending jobs) $T_{i,j}$ is *pending* at time t in a schedule $\mathcal{S}$ if $T_{i,j}$ is eligible at time t and $T_{i,j}$ has not completed execution by t in $\mathcal{S}$.

Definition 3. (ready jobs) A pending job $T_{i,j}$ is *ready* at time t in a schedule $\mathcal{S}$ if all prior jobs of T_i have completed execution by t in $\mathcal{S}$.

Definition 4. (window-constrained priorities) A scheduling algorithm's prioritization functions are *window-constrained* iff, for each task T_i, there exist constants ϕ_i and ψ_i such that, for each job $T_{i,j}$ of T_i and time t,

$$r_{i,j} - \phi_i \leq \chi(T_{i,j}, t) \leq d_{i,j} + \psi_i. \tag{4}$$

Note that (4) requires a job's χ-values to lie within a window $[r_{i,j} - \phi_i, d_{i,j} + \psi_i]$ that is defined with respect to its release time and deadline. Note also that the constants ϕ_i and ψ_i may be positive or negative; however, if negative, the interval $[r_{i,j} - \phi_i, d_{i,j} + \psi_i]$ cannot be empty.

It is easy to see that, other than RM, all of the algorithms considered in Section 3 have prioritization functions that satisfy (4). In contrast, the prioritization function specified for RM fails to be window-constrained because it violates the required lower bound: as new jobs of each task T_i are released, $\chi(T_{i,j}, t) < r_{i,j} - \phi_i$ will eventually hold for some job $T_{i,j}$ for any choice of the constant ϕ_i. It can be shown that the task system in Example 4 has unbounded tardiness. In particular, if the job-release pattern in Figure 3(b) recurs repeatedly, then the processing capacity available to T_4 every 12 time units is the same as is depicted in Figure 3(b). This capacity is less than the amount of work generated by T_4 during the same interval. As a result, more and more work shifts to future intervals, causing tardiness for T_4 to grow unboundedly. (The fact that tardiness can be unbounded under RM was also established by Devi (Devi, 2006).)

It is possible to "fix" the prioritization functions for RM so that the required lower bounds are adhered to, but then the upper bounds will be violated. For example, we could simply define $\chi(T_{i,j}, t) = i + t'$, where t' is the time where the most recent job release occurred at or before t. This definition simply shifts the χ-values defined earlier to future points in time as new jobs are released. However, we know that tardiness for T_4 in Example 4 is unbounded, so eventually $\chi(T_{4,j}, t) > d_{4,j} + \psi_4$ will hold for some pending job $T_{4,j}$ of T_4 for any choice of the constant ψ_4. Intuitively, Inequality (4) ensures that any job $T_{i,j}$ eventually becomes the highest-priority job in the system and will execute until completion. We summarize this discussion as follows. (Recall that any task set considered in this paper is assumed to satisfy (2).)

Theorem 1. *If either the lower or upper bound given in* (4) *is eliminated, then there exists a prioritization scheme that satisfies the remaining condition for which tardiness is unbounded for some task set.*

Definition 5. A task system is *concrete* if the release times and eligibility times of all jobs are specified, and *non-concrete*, otherwise.

Each of the schedules presented in Examples 3–9 was generated for a concrete task system in which jobs are released synchronously and periodically. However, the task set considered in each example is a non-concrete task system because (being sporadic) job release times are not specified.

Most of the rest of this paper is devoted to showing that any scheduling algorithm $\mathcal{A}$ with window-constrained prioritization functions has bounded tardiness. The tardiness bound established for $\mathcal{A}$ is derived by comparing the allocations to a concrete task system τ in an ideal processor-sharing (PS) schedule to those in a schedule produced by $\mathcal{A}$. In a PS *schedule*, each job of a task T_i is executed at a constant rate of $u_{i,j} = \frac{e_{i,j}}{p_i} \leq u_i$ between its release and deadline (Stoica et al,

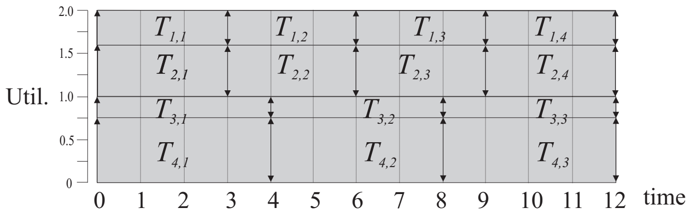


Figure 7: PS schedule for τ in Example 3.

1996). Figure 7 depicts an example. In this figure, the execution of each job $T_{i,j}$ is represented as a rectangle of length $p_i = d_{i,j} - r_{i,j}$ and height $u_{i,j}$. Therefore, the allocation of each job between its release time and deadline in this schedule is $u_{i,j} \cdot p_i = e_{i,j}$.

Note that a PS schedule does not depend on processor availability. Also, in such a schedule, each job completes exactly at its deadline. Thus, if a job misses its deadline, then it is "lagging behind" the corresponding PS schedule — this concept of "lag" is instrumental in the analysis and is formalized below. (A similar lag-based analysis was used by Devi and Anderson to establish tardiness bounds for preemptive and non-preemptive global EDF (Devi and Anderson, 2008)).

Let $\mathsf{A}(T_{i,j}, t_1, t_2, \mathcal{S})$ be the total allocation to the job $T_{i,j}$ in an arbitrary schedule $\mathcal{S}$ in $[t_1, t_2)$. Then, the difference between the allocations to $T_{i,j}$ up to time t in a PS schedule $\mathcal{PS}$ and an arbitrary schedule $\mathcal{S}$, termed the *lag of $T_{i,j}$ at time t in schedule $\mathcal{S}$*, is given by

$$\mathsf{lag}(T_{i,j}, t, \mathcal{S}) = \mathsf{A}(T_{i,j}, 0, t, \mathcal{PS}) - \mathsf{A}(T_{i,j}, 0, t, \mathcal{S}). \tag{5}$$

Task lags can be similarly defined:

$$\mathsf{lag}(T_i, t, \mathcal{S}) = \sum_{j \geq 1} \mathsf{lag}(T_{i,j}, t, \mathcal{S}) = \sum_{j \geq 1} \mathsf{A}(T_{i,j}, 0, t, \mathcal{PS}) - \mathsf{A}(T_{i,j}, 0, t, \mathcal{S}). \tag{6}$$

Finally, the *lag for a finite job set Φ at time t in the schedule $\mathcal{S}$* is defined by

$$\mathsf{LAG}(\Phi, t, \mathcal{S}) = \sum_{T_{i,j} \in \Phi} \mathsf{lag}(T_{i,j}, t, \mathcal{S}) = \sum_{T_{i,j} \in \Phi} (\mathsf{A}(T_{i,j}, 0, t, \mathcal{PS}) - \mathsf{A}(T_{i,j}, 0, t, \mathcal{S})). \tag{7}$$

Since $\mathsf{LAG}(\Phi, 0, \mathcal{S}) = 0$, the following holds for $t' \leq t$.

$$\mathsf{LAG}(\Phi, t, \mathcal{S}) = \mathsf{LAG}(\Phi, t', \mathcal{S}) + \mathsf{A}(\Phi, t', t, \mathcal{PS}) - \mathsf{A}(\Phi, t', t, \mathcal{S}) \tag{8}$$

The concept of lag is important because, if lags remain bounded, then tardiness is bounded as well.

Definition 6. A time interval $[t_1, t_2)$ is *busy* for a job set Φ in schedule $\mathcal{S}$ if, at each time $t \in [t_1, t_2)$, all m processors execute jobs from Φ in this schedule, and is *non-busy* for Φ otherwise.

When using the above terminology, we will omit "for Φ" if the job set under consideration is clear. According to the lemma below, the lag for a job set Φ cannot increase across a busy interval for Φ. This fact was proved in the context of global EDF in (Devi et al, 2006). However, since the proof relies only on the fact that the interval in question is busy, and not on how jobs are scheduled, it applies in our context as well. Later, we will examine the behavior of the LAG function over an interval where some processors are unavailable.

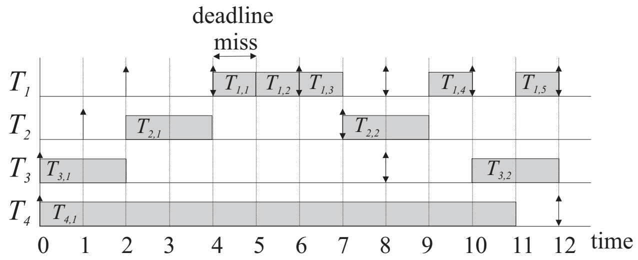


Figure 8: A schedule for τ in Example 10.

Lemma 1. *For any interval* $[t_1, t_2)$ *that is busy for* Φ, $\mathsf{LAG}(\Phi, t_2, \mathcal{S}) \leq \mathsf{LAG}(\Phi, t_1, \mathcal{S})$.

Proof. By (8),

$$\mathsf{LAG}(\Phi, t_2, \mathcal{S}) = \mathsf{LAG}(\Phi, t_1, \mathcal{S}) + \mathsf{A}(\Phi, t_1, t_2, \mathcal{PS}) - \mathsf{A}(\Phi, t_1, t_2, \mathcal{S}). \tag{9}$$

Because the interval $[t_1, t_2)$ is busy, m processors execute jobs from Φ throughout the interval, and thus $\mathsf{A}(\Phi, t_1, t_2, \mathcal{S}) = m \cdot (t_2 - t_1)$. In the ideal PS schedule $\mathcal{PS}$, each job $T_{i,j}$ executes with a constant rate $u_{i,j} \leq u_i$ from its release to its deadline, and thus

$$\mathsf{A}(\Phi, t_1, t_2, \mathcal{PS}) \leq \sum_{T_i \in \tau} \sum_{j>0} \mathsf{A}(T_{i,j}, t_1, t_2, \mathcal{PS}) \leq \sum_{T_i \in \tau} u_i \cdot (t_2 - t_1) = U_{sum} \cdot (t_2 - t_1).$$

Setting this inequality and $\mathsf{A}(\Phi, t_1, t_2, \mathcal{S}) = m \cdot (t_2 - t_1)$ into (9) and applying $U_{sum} \leq \sum_{k=1}^{m} \widehat{u_k} \leq m$, we get

$$\begin{aligned}
\mathsf{LAG}(\Phi, t_2, \mathcal{S}) &= \mathsf{LAG}(\Phi, t_1, \mathcal{S}) + \mathsf{A}(\Phi, t_1, t_2, \mathcal{PS}) - \mathsf{A}(\Phi, t_1, t_2, \mathcal{S}) \\
&\leq \mathsf{LAG}(\Phi, t_1, \mathcal{S}) + U_{sum}(t_2 - t_1) - m \cdot (t_2 - t_1) \\
&\leq \mathsf{LAG}(\Phi, t_1, \mathcal{S}).
\end{aligned}$$

□

We are interested in non-busy intervals (for a job set) because total lag (for that job set) can increase only across such (non-busy) intervals, and such increases may lead to deadline misses. The following example illustrates how lag can change across busy and non-busy intervals.

Example 10. Consider a two-processor system upon which a task set $\tau = \{T_1 = (1, 2), T_2 = (2, 6), T_3 = (2, 8), T_4 = (11, 12)\}$ is to be scheduled, where the first jobs of T_1, T_2, T_3, and T_4 are released at times 2, 1, 0, and 0 respectively. The total utilization of the system is $U_{sum} = 1/2 + 2/6 + 2/8 + 11/12 = 2$. Assume that both processors are always available, i.e., $\widehat{u_1} = \widehat{u_2} = 1$ and $\sigma_1 = \sigma_2 = 0$, and $\mathcal{A}$ is the FIFO algorithm, i.e., jobs are prioritized using $\chi(T_{i,j}, t) = r_{i,j}$ (assume there are no early releases). Consider the schedule for τ in Figure 8. Under $\mathcal{A}$, $T_{1,1}$ misses its deadline at time 4 by one time unit because it cannot preempt $T_{2,1}$ and $T_{4,1}$, which have earlier release times and later deadlines.

Let $\Phi = \{T_{1,1}, \ldots, T_{1,5}, T_{2,1}, T_{3,1}, T_{4,1}\}$ be the set of jobs with deadlines at most 12. The interval $[4, 7)$ in Figure 8 is a busy interval for Φ, because all processors execute jobs from Φ throughout the interval. By (8), $\mathsf{LAG}(\Phi, 7, \mathcal{S}) = \mathsf{LAG}(\Phi, 4, \mathcal{S}) + \mathsf{A}(\Phi, 4, 7, \mathcal{PS}) - \mathsf{A}(\Phi, 4, 7, \mathcal{S})$, where $\mathcal{S}$ is the schedule under $\mathcal{A}$. The allocation of Φ in the PS schedule $\mathcal{PS}$ during the interval $[4, 7)$ is $\mathsf{A}(\Phi, 4, 7, \mathcal{PS}) = 3 \cdot (u_1 + u_2 + u_3 + u_4) = 3/2 + 6/6 + 6/8 + 33/12 = 6$. The allocation of Φ in $\mathcal{S}$ throughout $[4, 7)$ is also 6. Thus, $\mathsf{LAG}(\Phi, 7, \mathcal{S}) = \mathsf{LAG}(\Phi, 4, \mathcal{S})$.

Now let $\Phi = \{T_{1,1}\}$ be the set of jobs with deadlines at most 4. Because the jobs $T_{2,1}$ and $T_{4,1}$, which have deadlines after time 4, execute within the interval $[2, 4)$ in Figure 8, this interval is non-busy for Φ in $\mathcal{S}$. By (7), $\mathsf{LAG}(\Phi, 4, \mathcal{S}) = \mathsf{A}(\Phi, 0, 4, \mathcal{PS}) - \mathsf{A}(\Phi, 0, 4, \mathcal{S})$. The allocation of Φ in the PS schedule $\mathcal{PS}$ throughout the interval $[0, 4)$ is $\mathsf{A}(\Phi, 0, 4, \mathcal{PS}) = 2 \cdot 1/2 = 1$. The allocation of Φ in $\mathcal{S}$ is $\mathsf{A}(\Phi, 0, 4, \mathcal{S}) = 0$. Thus, $\mathsf{LAG}(\Phi, 4, \mathcal{S}) = 1 - 0 = 1$. Figure 8 shows that at time 4, $T_{1,1}$ from Φ is pending. This job has unit execution cost, which is equal to the amount of pending work given by $\mathsf{LAG}(\Phi, 4, \mathcal{S})$.

4.2 Tardiness Bound for $\mathcal{A}$

Given an arbitrary non-concrete task system τ^N (where the eligibility times and release times of jobs are not specified – see Definition 5), we want to determine the maximum tardiness of any job of any task in any concrete instantiation of τ^N scheduled on m processors. The approach for doing this is based on techniques from (Devi and Anderson, 2008). Let τ be a concrete instantiation of τ^N. First, we order the jobs in the concrete instantiation using the following rule: $T_{i,j} \prec T_{a,b}$ iff $d_{i,j} < d_{a,b}$ or $(d_{i,j} = d_{a,b}) \wedge i < a$.

Let

$$\rho = \max\left(0, \max_{i \neq a}(\psi_a + \phi_i)\right) \quad \text{and} \quad \mu = \max\left(0, \max_{i \neq a}(p_a + \psi_a + \phi_i)\right) \tag{10}$$

Let $T_{\ell,q}$ be a job of a task T_ℓ in τ, let $t_d = d_{\ell,q}$, and let $\mathcal{S}$ be a schedule, produced for τ by the scheduling algorithm $\mathcal{A}$. We assume that the schedule $\mathcal{S}$ has the following property.

(P) The tardiness of every job $T_{k,h}$ such that $T_{k,h} \prec T_{\ell,q}$ is at most $x + e_k$, where $x \geq \rho \geq 0$.

Our goal is to determine the smallest $x \geq \rho$ such that the tardiness of $T_{\ell,q}$ remains at most $x + e_\ell$. Such a result would by induction imply a tardiness of at most $x + e_k$ for all jobs of every task $T_k \in \tau$. Because τ is arbitrary, the tardiness bound will hold for every concrete instantiation of τ^N.

The objective is easily met if $T_{\ell,q}$ completes by its deadline, t_d, so assume otherwise. The completion time of $T_{\ell,q}$ then depends on the demand of the jobs that can compete with $T_{\ell,q}$ after t_d and on the amount of available processor time after t_d. Hence, a value for x can be determined via the following steps.

1. Compute an upper bound on the demand for jobs (including $T_{\ell,q}$) that can compete with $T_{\ell,q}$ after t_d.
2. Determine the amount of such demand necessary for the tardiness of $T_{\ell,q}$ to exceed $x + e_\ell$.
3. Determine the smallest $x \geq \rho$ such that the tardiness of $T_{\ell,q}$ is at most $x + e_\ell$ using the upper bound in Step 1 and the necessary condition in Step 2.

To reason about the tardiness of $T_{\ell,q}$, we need to determine how other jobs delay its execution. To do that, we first define a boolean function of two jobs $T_{i,k}$ and $T_{a,b}$ that will allow us to exclude certain jobs from consideration:

$$\mathsf{LP}(T_{i,k}, T_{a,b}) = (\forall\, t \;:\; d_{a,b} + \psi_a < \chi(T_{i,k}, t)). \tag{11}$$

Claim 1. *If* $\mathsf{LP}(T_{i,k}, T_{a,b})$ *holds for jobs* $T_{i,k}$ *and* $T_{a,b}$*, then* $\chi(T_{a,b}, t) < \chi(T_{i,k}, t)$ *for any time* t.

Proof. We upper bound $\chi(T_{a,b}, t)$ as follows.

$$
\begin{aligned}
&\chi(T_{a,b}, t)\\
&\qquad \{\text{by (4)}\}\\
&\leq d_{a,b} + \psi_a\\
&\qquad \{\text{by the condition of the claim and (11)}\}\\
&< \chi(T_{i,k}, t)
\end{aligned}
$$

□

Claim 1 provides a sufficient condition for a job $T_{i,k}$ to have lower priority (a larger χ-value) than that of $T_{a,b}$ at any time and therefore not compete with $T_{a,b}$ for processor time. In the rest of the proof, four job sets, **d**, **DH**, **DLH**, and **DLL**, are considered. **d** and **DH** are defined as follows.

$$\mathbf{d} = \{T_{i,k} :: d_{i,k} \leq d_{\ell,q} = t_d\} \tag{12}$$

$$\mathbf{DH} = \{T_{i,k} :: (d_{i,k} > t_d) \wedge (i \neq \ell) \wedge (\exists\, T_{a,b} \in \mathbf{d} : (a \neq i) :: \neg\mathsf{LP}(T_{i,k}, T_{a,b}))\} \tag{13}$$

In this notation, **d** and **D** denote, respectively, jobs with deadlines at most and greater than t_d. The letter **H** in **DH** denotes that $T_{i,k}$'s priority at some time *may be higher* than that of a job of different task in **d** (refer to Claim 1). Note that, because $d_{\ell,y} \leq d_{\ell,q} = t_d$,

$$(\forall y : y \leq q :: T_{\ell,y} \in \mathbf{d}). \tag{14}$$

The remaining two job sets are defined as follows.

$$
\begin{aligned}
\mathbf{DLH} = \{T_{i,k} :: (d_{i,k} > t_d) \wedge (i \neq \ell) \wedge (\forall\, T_{a,b} \in \mathbf{d} : (a \neq i) :: \mathsf{LP}(T_{i,k}, T_{a,b}))\\
\wedge (\exists\, T_{a,b} \in \mathbf{DH} : (a \neq i) :: \neg\mathsf{LP}(T_{i,k}, T_{a,b}))\}
\end{aligned} \tag{15}
$$

$$
\begin{aligned}
\mathbf{DLL} = \{T_{i,k} :: (d_{i,k} > t_d) \wedge (i \neq \ell) \wedge (\forall\, T_{a,b} \in \mathbf{d} : (a \neq i) :: \mathsf{LP}(T_{i,k}, T_{a,b}))\\
\wedge (\forall\, T_{a,b} \in \mathbf{DH} : (a \neq i) :: \mathsf{LP}(T_{i,k}, T_{a,b}))\}
\end{aligned} \tag{16}
$$

If $T_{i,k}$ is in **DLH** or **DLL**, then, for each job $T_{a,b} \in \mathbf{d}$ such that $a \neq i$, $\mathsf{LP}(T_{i,k}, T_{a,b})$ holds, and hence, $T_{i,k}$'s priority *is always lower* than that of any job in **d** of a different task. The second letter **L** in **DLH** and **DLL** is intended to denote this. Similarly, the third letter **H** in **DLH** denotes that job $T_{i,k}$'s priority may be higher than that of a job of a different task T_a that belongs to **DH**. Finally, the third letter **L** in **DLL** denotes that job $T_{i,k}$'s priority is always lower than that of any job of a different task T_a that belongs to **DH**.

Example 11. Consider the task set $\tau = \{T_1 = (1, 2),\, T_2 = (1.5, 3), T_3 = (5, 5)\}$ and the PS schedule for it in Figure 9. Job $T_{1,1}$ is released at time 1, and jobs $T_{2,1}$ and $T_{3,1}$ are released at time 0. Consider the job $T_{\ell,q} = T_{1,1}$, which has a deadline at time 3. Assume that there are no early releases and jobs are prioritized as follows. For task T_1, $\chi(T_{1,j}, t) = d_{1,j}$ for all j. For task T_2, $\chi(T_{2,j}, t) = r_{2,j}$ if j is even and $\chi(T_{2,j}, t) = d_{2,j}$ if j is odd. For task T_3, $\chi(T_{3,j}, t) = r_{3,j}$ for all j.

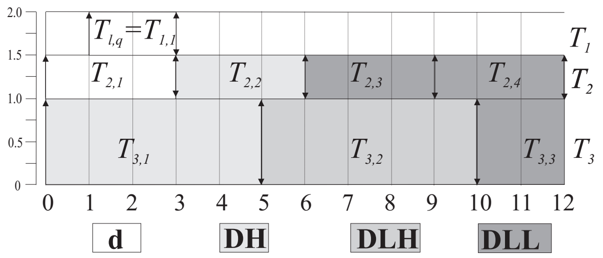


Figure 9: Job set partitioning.

We thus have, $\phi_1 = -p_1$, $\phi_2 = \phi_3 = 0$, $\psi_1 = 0$, $\psi_2 = 0$, and $\psi_3 = -p_3$. With respect to $T_{1,1}$, the four sets mentioned above are **d** $= \{T_{1,1}, T_{2,1}\}$, **DH** $= \{T_{3,1}, T_{2,2}\}$, **DLH** $= \{T_{3,2}\}$, and **DLL** $= \{T_{2,3}, T_{2,4}, T_{3,3}\}$. The job $T_{2,2} \in$ **DH** because $\chi(T_{2,2}, t) = r_{2,2} = 3 \leq d_{1,1} = 3$, and hence, $\mathsf{LP}(T_{2,2}, T_{1,1})$ does not hold. The job $T_{3,2} \in$ **DLH** because $\chi(T_{3,2}, t) = r_{3,2} = 5 \leq d_{2,2} = 6$, and hence, $\mathsf{LP}(T_{3,2}, T_{2,2})$ does not hold. **DLL** would also include any jobs of tasks other than T_1 released after time 12.

We now prove some important relationships between the priorities of jobs in the four sets mentioned above.

Lemma 2. *If $T_{a,b} \in$ **DH** and $T_{i,k} \in$ **DLL**, where $a \neq i$, then $\chi(T_{a,b}, t) < \chi(T_{i,k}, t)$ for any time t.*

Proof. If $T_{i,k}$ in **DLL**, then, by (16), $(\forall T_{a,b} \in \mathbf{DH} : (a \neq i) :: \mathsf{LP}(T_{i,k}, T_{a,b}))$. By the condition of the lemma, this implies that $\mathsf{LP}(T_{i,k}, T_{a,b})$ holds. The required result follows from Claim 1. □

Lemma 3. *If $T_{a,b} \in$ **d** and $T_{i,k} \in$ **DLL** $\cup$ **DLH**, where $a \neq i$, then $\chi(T_{a,b}, t) < \chi(T_{i,k}, t)$ for any time t.*

Proof. If $T_{i,k} \in$ **DLL** $\cup$ **DLH**, then, by (15) and (16), $(\forall T_{a,b} \in \mathbf{d} : (a \neq i) :: \mathsf{LP}(T_{i,k}, T_{a,b}))$ holds. By the condition of the lemma, this implies that $\mathsf{LP}(T_{i,k}, T_{a,b})$ holds. The required result follows from Claim 1. □

Lemma 4. *If a job $T_{i,k} \in$ **DLL** is scheduled at time t or there is an idle available processor at time t, and $T_{a,b} \in$ **d** $\cup$ **DH** is ready at time t, where $a \neq i$, then $T_{a,b}$ is scheduled at time t.*

Proof. The case when an available processor is idle at time t is trivial so suppose that this is not the case. If $T_{i,k}$ and $T_{a,b}$ are defined as in the statement of the lemma, and $T_{i,k}$ is scheduled at time t, then $T_{a,b}$ is scheduled at time t as well since, by Lemmas 2 and 3, $\chi(T_{a,b}, t) < \chi(T_{i,k}, t)$. □

Lemma 5. *If a job $T_{i,k} \in$ **DLH** $\cup$ **DLL** is scheduled at time t and $T_{a,b} \in$ **d** is ready at time t, where $a \neq i$, then $T_{a,b}$ is scheduled at time t.*

Proof. If $T_{i,k}$ and $T_{a,b}$ are defined as in the statement of the lemma, and $T_{i,k}$ is scheduled at time t, then $T_{a,b}$ is scheduled as well, since by Lemma 3, $\chi(T_{a,b}, t) < \chi(T_{i,k}, t)$. □

Corollary 1. *If a job $T_{i,k} \in$ **DLH** $\cup$ **DLL** is scheduled at time $t \geq t_d$ and job $T_{\ell,q}$ is pending at time t, then T_ℓ is scheduled at t.*

Proof. If $T_{\ell,q}$ is pending at time $t \geq t_d$, then the earliest pending job of T_ℓ, $T_{\ell,y}$, where $y \leq q$ is ready at time t. The required result follows from (14) and Lemma 5. $\square$

Determining an upper bound on competing demand. We are now ready to establish the upper bound mentioned in the first step of the proof outline given earlier as a function of job sets **d**, **DH**, **DLH**, and **DLL**.

Definition 7. Let $W(\alpha)$ be the total allocation of jobs in the set α in schedule $\mathcal{S}$ after time t_d while job $T_{\ell,q}$ is pending.

We are interested in the allocation of jobs in $\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH}$ because these jobs may delay the execution of $T_{\ell,q}$. (By Lemma 4, jobs in **DLL** cannot delay $T_{\ell,q}$ or prior jobs of T_ℓ.) Their allocation after t_d while $T_{\ell,q}$ is pending, is

$$W(\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH}) = W(\mathbf{d}) + W(\mathbf{DH} \cup \mathbf{DLH}). \tag{17}$$

Because jobs from **d** have deadlines at most t_d, they do not execute in the PS schedule $\mathcal{PS}$ beyond t_d. Thus, the allocation of jobs in **d** after time t_d is upper-bounded by the amount of pending work due to jobs in this set at time t_d as given by $\mathsf{LAG}(\mathbf{d}, t_d, \mathcal{S})$, which must be positive in order for $T_{\ell,q}$ to miss its deadline at t_d (by (14)). Therefore,

$$W(\mathbf{d}) \leq \mathsf{LAG}(\mathbf{d}, t_d, \mathcal{S}). \tag{18}$$

From (17) and (18), we have

$$W(\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH}) \leq \mathsf{LAG}(\mathbf{d}, t_d, \mathcal{S}) + W(\mathbf{DH} \cup \mathbf{DLH}). \tag{19}$$

Thus, an upper bound on $W(\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH})$ can be obtained by determining bounds for $\mathsf{LAG}(\mathbf{d}, t_d, \mathcal{S})$ and $W(\mathbf{DH} \cup \mathbf{DLH})$ individually.

Upper bound on $\mathsf{LAG}(\mathbf{d}, t_d, \mathcal{S})$. In deriving this bound, we assume that all busy and non-busy intervals considered are with respect to **d** and the schedule $\mathcal{S}$ is produced by the scheduling algorithm $\mathcal{A}$.

To begin, note that, by Lemma 1, if no non-busy interval exists in $[0, t_d)$, then $\mathsf{LAG}(\mathbf{d}, t_d, \mathcal{S}) \leq \mathsf{LAG}(\mathbf{d}, 0, \mathcal{S}) = 0$. In that which follows, we consider the more interesting case wherein some non-busy interval exists in $[0, t_d)$. An interval could be non-busy for two reasons:

1. There are not enough ready jobs in **d** to occupy all available processors, so it is immaterial whether jobs from **DH**, **DLH**, or **DLL** execute during the interval.

2. There are tasks with ready jobs in **d** that cannot execute because, within certain sub-intervals, some processors are not available (because of capacity restrictions) or jobs in **DH** occupy one or more processors because they have higher priority. Note that, by Lemma 5, jobs in **DLH** and **DLL** cannot execute at time instants when there are ready unscheduled jobs in **d**.

Jobs with deadlines after time t_d may prevent the execution of jobs in **d** before time t_d (if such jobs become eligible before t_d) and hence increase the LAG for **d**.

Definition 8. **($\tau_{\mathbf{DH}}$)** Let $\tau_{\mathbf{DH}}$ be the set of tasks that have jobs in **DH**.

Definition 9. **(δ_i)** Let δ_i be the total allocation of task T_i's jobs in **DH** in the schedule $\mathcal{S}$ by time t_d.

In much of the rest of the analysis, we focus on a time t_n defined as follows.

Definition 10. If there exists a time instant t such that there are at most $m-1$ tasks with ready jobs in **d** at time t^- and all these tasks execute at time t^-, then define t_n to be the latest such time instant at or before t_d; if no such t exists, then let $t_n = 0$.

We express a bound on $\mathsf{LAG}(\mathbf{d}, t_d, \mathcal{S})$ in terms of individual task parameters and processor availability functions using Lemmas 6, 7, and 8, which are proved in an appendix. Lemma 7 establishes a relationship between $\mathsf{LAG}(\mathbf{d}, t_n, \mathcal{S})$ and $\mathsf{LAG}(\mathbf{d}, t_d, \mathcal{S})$. Lemmas 6 and 8 were initially proved in (Devi et al, 2006) in the context of global EDF, for the case where all processors are fully available. The proof of each lemma relies only on Property (P) and, for Lemma 7, the definition of t_n. In particular, the exact way in which jobs are scheduled does not arise.

Lemma 6: $\mathsf{lag}(T_k, t, \mathcal{S}) \le x \cdot u_k + e_k$ for any task T_k and $t \in [0, t_d]$.

Lemma 7: $\mathsf{LAG}(\mathbf{d}, t_d, \mathcal{S}) \le \mathsf{LAG}(\mathbf{d}, t_n, \mathcal{S}) + \sum_{T_i \in \tau_{\mathbf{DH}}} \delta_i + \sum_{k=1}^{m} \widehat{u_k} \cdot \sigma_k$.

Definition 11. **($U(\tau, y)$ and $E(\tau, y)$)** Let $U(\tau, y)$ ($E(\tau, y)$) be the set of at most $\min(|\tau|, y)$ tasks from τ of highest utilization (execution cost), where $|\tau|$ is the number of tasks in τ, and let

$$E_L = \sum_{T_i \in E(\tau, m-1)} e_i \quad \text{and} \quad U_L = \sum_{T_i \in U(\tau, m-1)} u_i.$$

Lemma 8: $\mathsf{LAG}(\mathbf{d}, t_n, \mathcal{S}) \le E_L + x \cdot U_L$.

Using Lemmas 7 and 8, we can upper bound $\mathsf{LAG}(\mathbf{d}, t_d, \mathcal{S})$ in (19).

Lemma 9: $\mathsf{LAG}(\mathbf{d}, t_d, \mathcal{S}) \le E_L + x \cdot U_L + \sum_{T_i \in \tau_{\mathbf{DH}}} \delta_i + \sum_{k=1}^{m} \widehat{u_k} \cdot \sigma_k$.

Upper bound on W(DH$\cup$DLH). The jobs in **DH**$\cup$**DLH** may delay the execution of $T_{\ell,q}$ because some of these jobs may have higher priority than $T_{\ell,q}$ at some time. We now upper-bound the total execution demand due to jobs in **DH**$\cup$**DLH**. Lemmas 10 and 11, which are proved in the appendix, upper-bound the release times of jobs in **DH** $\cup$ **DLH** using ρ and μ from (10).

Lemma 10. *If $T_{i,k} \in$ **d** $\cup$ **DH**, then $r_{i,k} \le t_d + \rho$.*

Lemma 11. *If $T_{i,k} \in$ **DLH**, then $r_{i,k} \le t_d + \rho + \mu$.*

Similarly to Definition 8, we define the following task set.

Definition 12. **($\tau_{\mathbf{DLH}}$)** Let $\tau_{\mathbf{DLH}}$ be the set of tasks that have jobs in **DLH**.

Lemma 12. *Task $T_i \in \tau_{\mathbf{DH}}$ can have at most $\left\lceil \frac{\rho}{p_i} \right\rceil$ jobs in* **DH** *with release times after t_d. Task $T_i \in \tau_{\mathbf{DLH}}$ can have at most $\left\lceil \frac{\rho+\mu}{p_i} \right\rceil$ jobs in* **DLH** *with release times after t_d.*

Proof. Suppose that $T_{i,k} \in$ **DH** $\cup$ **DLH** and $r_{i,k} > t_d$. If $T_{i,k} \in$ **DH**, then, by Lemma 10, $r_{i,k} \le t_d + \rho$. If $T_{i,k} \in$ **DLH**, then, by Lemma 11, $r_{i,k} \le t_d + \rho + \mu$. Because task T_i's consecutive job releases are separated by at least p_i time units, the lemma follows. $\square$

Lemma 13: $W(\mathbf{DH} \cup \mathbf{DLH}) \leq \sum_{T_i \in \tau_{\mathbf{DH}} \cup \tau_{\mathbf{DLH}}} \left(\left(\left\lceil \frac{\rho+\mu}{p_i} \right\rceil + 1\right) \cdot e_i\right) - \sum_{T_i \in \tau_{\mathbf{DH}}} \delta_i$

Proof. Consider $T_i \in \tau_{\mathbf{DH}} \cup \tau_{\mathbf{DLH}}$. Each job $T_{i,k}$ in $\mathbf{DH} \cup \mathbf{DLH}$ is released either at or before t_d or after t_d. Because each job in $\mathbf{DH} \cup \mathbf{DLH}$ has a deadline after t_d, each T_i has at most one job in $\mathbf{DH} \cup \mathbf{DLH}$ with a release time at or before t_d. The demand due to this job is at most e_i. By Lemma 12, the demand of jobs of T_i in $\mathbf{DH} \cup \mathbf{DLH}$ released after t_d is at most $\left\lceil \frac{\rho+\mu}{p_i} \right\rceil \cdot e_i$. The allocation of task T_i's jobs in $\mathbf{DH}$ in schedule $\mathcal{S}$ before time t_d is δ_i, by Definition 9. Thus, the allocation of all jobs in $\mathbf{DH} \cup \mathbf{DLH}$ after time t_d in schedule $\mathcal{S}$ while $T_{\ell,q}$ is pending is

$$\begin{aligned}
&W(\mathbf{DH} \cup \mathbf{DLH}) \\
&\quad \leq \sum_{T_i \in \tau_{\mathbf{DH}} \cup \tau_{\mathbf{DLH}}} \left(\left\lceil \frac{\rho+\mu}{p_i} \right\rceil \cdot e_i + e_i\right) - \sum_{T_i \in \tau_{\mathbf{DH}}} \delta_i \\
&\quad = \sum_{T_i \in \tau_{\mathbf{DH}} \cup \tau_{\mathbf{DLH}}} \left(\left(\left\lceil \frac{\rho+\mu}{p_i} \right\rceil + 1\right) \cdot e_i\right) - \sum_{T_i \in \tau_{\mathbf{DH}}} \delta_i
\end{aligned}$$

□

Upper bound on $W(\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH})$.

Definition 13. Let $\alpha(\tau, \ell) \geq \sum_{T_i \in \tau_{\mathbf{DH}} \cup \tau_{\mathbf{DLH}}} \left(\left(\left\lceil \frac{\rho+\mu}{p_i} \right\rceil + 1\right) \cdot e_i\right)$ be a scheduling-algorithm-dependent bound on the competing demand due to jobs in **DH** and **DLH**.

From (19), Lemma 9, and Lemma 13 we have

$$\begin{aligned}
&W(\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH}) \\
&\qquad \{\text{by (19)}\} \\
&\quad \leq \mathsf{LAG}(\mathbf{d}, t_d, \mathcal{S}) + W(\mathbf{DH} \cup \mathbf{DLH}) \\
&\qquad \{\text{by Lemmas 9 and 13}\} \\
&\quad \leq E_L + x \cdot U_L + \sum_{T_i \in \tau_{\mathbf{DH}}} \delta_i + \sum_{k=1}^{m} \widehat{u_k} \cdot \sigma_k \\
&\qquad + \sum_{T_i \in \tau_{\mathbf{DH}} \cup \tau_{\mathbf{DLH}}} \left(\left(\left\lceil \frac{\rho+\mu}{p_i} \right\rceil + 1\right) \cdot e_i\right) - \sum_{T_i \in \tau_{\mathbf{DH}}} \delta_i \\
&\quad = E_L + x \cdot U_L + \sum_{k=1}^{m} \widehat{u_k} \cdot \sigma_k + \sum_{T_i \in \tau_{\mathbf{DH}} \cup \tau_{\mathbf{DLH}}} \left(\left(\left\lceil \frac{\rho+\mu}{p_i} \right\rceil + 1\right) \cdot e_i\right) \\
&\qquad \{\text{by Definition 13}\} \\
&\quad \leq E_L + x \cdot U_L + \sum_{k=1}^{m} \widehat{u_k} \cdot \sigma_k + \alpha(\tau, \ell) \qquad (20)
\end{aligned}$$

Claim 2. *The expression* $\sum_{T_i \in \tau \setminus T_\ell} \left(\left(\left\lceil \frac{\rho+\mu}{p_i} \right\rceil + 1\right) \cdot e_i\right)$ *(conservatively) upper-bounds* $\alpha(\tau, \ell)$ *for any window-constrained scheduler.*

Proof. The claim follows from $\tau_{\mathbf{DH}} \cup \tau_{\mathbf{DLH}} \subseteq \tau \setminus T_\ell$. □

In Section 5, we will discuss how to compute tighter bounds for $\alpha(\tau, \ell)$ for GEDF and FIFO schedulers.

Necessary condition for tardiness to exceed $x + e_\ell$. We now find the amount of competing work that is necessary for $T_{\ell,q}$ to miss its deadline by more than $x + e_\ell$ time units. Job $T_{\ell,q}$'s tardiness depends on the amount of competing demand $W(\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH})$ and on the amount of processor time available to τ after time t_d.

Definition 14. Let $\beta_k^* \geq \beta_k(x + e_\ell)$ be the amount of processor time available to tasks in τ during the interval $[t_d, t_d + x + e_\ell)$ on processor k in schedule $\mathcal{S}$. Let $R = \sum_{k=1}^{m}(x + e_\ell - \beta_k^*)$ be the total amount of processor time that is *not available* to τ during $[t_d, t_d + x + e_\ell)$.

Definition 15. Let F be the number of processors that could be unavailable, i.e., $F = |k :: \beta_k(\Delta) < \Delta|$.

Lemma 14. *If at most $m - F$ tasks with ready jobs in* $\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH}$ *are scheduled at time $t^* \in [t_d + \rho, t_d + x + e_\ell)$, $T_{\ell,q}$ is pending at t^*, and there is an idle available processor at time t^* or a job from* **DLL** *is scheduled at time t^*, then* **(i)** *task T_ℓ is scheduled at t^*, and* **(ii)** *T_ℓ is guaranteed uninterrupted execution until the job $T_{\ell,q}$ completes.*

Proof. **(i)** follows from Corollary 1. To prove **(ii)**, assume that the antecedent of the lemma holds. Let $A(t)$ ($B(t)$) be the number of tasks that have ready jobs in **d** (**DH**) at time $t \geq t^*$. By Lemma 4, all tasks with ready jobs in $\mathbf{d} \cup \mathbf{DH}$ are scheduled at time t^*, and hence,

$$A(t^*) + B(t^*) \leq m - F. \tag{21}$$

Suppose, contrary to the statement of the lemma, that T_ℓ executes uninterruptedly within $[t^*, t')$ but is preempted at time t' so that $T_{\ell,q}$ is pending at t'. By Lemma 5, no job in $\mathbf{DLH} \cup \mathbf{DLL}$ can be scheduled at time t' (since $T_{\ell,q} \in \mathbf{d}$). Therefore, at time t', all available processors are occupied by tasks with ready jobs in $\mathbf{d} \cup \mathbf{DH}$, and T_ℓ has ready job (in **d**) at time t' that is not scheduled. This implies $A(t') + B(t') > m - F$, and, by (21),

$$A(t') + B(t') > A(t^*) + B(t^*). \tag{22}$$

By Lemma 10, all jobs in $\mathbf{d} \cup \mathbf{DH}$ are released at or before $t_d + \rho$. Therefore, the number of tasks with ready jobs in $\mathbf{d} \cup \mathbf{DH}$ at time $t' > t^*$, $A(t') + B(t')$, cannot be higher than $A(t^*) + B(t^*)$, i.e., $A(t') + B(t') \leq A(t^*) + B(t^*)$. This contradicts (22). ☐

The following lemma establishes a lower bound on the competing demand for $T_{\ell,q}$.

Lemma 15. *If the tardiness of $T_{\ell,q}$ exceeds $x + e_\ell$, where $x \geq \rho$, then*

$$W(\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH}) + R > (m - (m - a) \cdot u_\ell) \cdot x + (1 - a) \cdot \rho + e_\ell, \tag{23}$$

where $a = \min(m, m - F + 1)$.

Proof. Assume that

$$W(\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH}) + R \leq (m - (m - a) \cdot u_\ell) \cdot x + (1 - a) \cdot \rho + e_\ell \tag{24}$$

holds and suppose, contrary to the statement of the lemma, that

(T) the tardiness of $T_{\ell,q}$ exceeds $x + e_\ell$.

In the rest of the proof, we say that a time instant $t \geq t_d$ (or an interval) is *WR-occupied* if each processor either executes a job from $\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH}$ or is unavailable; otherwise, we say that t is *WR-free*. The prefix *"WR"* denotes that all processors contribute to the allocation $W(\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH}) + R$. If the time instant $t \geq t_d$ is *WR*-free, then either at least one available processor is idle at t, or a job from **DLL** is scheduled at time t. Because, by (T), $T_{\ell,q} \in \mathbf{d}$ is pending throughout the interval $[t_d, t_d + x + e_\ell)$, the following property holds by Corollary 1:

(E) task T_ℓ executes at each *WR*-free instant within $[t_d, t_d + x + e_\ell)$.

By (P), the preceding job $T_{\ell,q-1}$ (if it exists) completes by time

$$t' \leq t_d - p_\ell + e_\ell + x \leq t_d + x. \tag{25}$$

Thus, $t_d + x$ is the latest time at which $T_{\ell,q}$ may become ready. If the latest *WR*-occupied instant in the interval $[t_d, t_d + x + e_\ell)$ is at or before $t_d + x$, then, by (E), $T_{\ell,q}$ executes uninterruptedly after $t_d + x$ and its tardiness is at most $x + e_{\ell,q} \leq x + e_\ell$, contrary to (T). In the rest of the proof, we assume that the latest *WR*-occupied instant in the interval $[t_d, t_d + x + e_\ell)$ is after $t_d + x$.

Suppose that at most $m - F$ processors execute jobs from $\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH}$ at some *WR*-free instant $t^* \in [t_d + \rho, t_d + x)$. In this case, because t^* is *WR*-free, some processor is idle or a job in **DLL** is scheduled there. Thus, by Lemma 14, T_ℓ is guaranteed uninterrupted execution at or after time t^* until $T_{\ell,q}$ finishes. By (25), $T_{\ell,q-1}$ (if it exists) finishes its execution by time $t' \leq t_d + x$, so $T_{\ell,q}$ finishes by time $t' + e_{\ell,q} \leq t_d + x + e_{\ell,q} \leq t_d + x + e_\ell$, thereby having tardiness at most $x + e_\ell$, contrary to (T).

In the rest of the proof, we assume the following:

(N) at least $a = \min(m, m - F + 1)$ processors execute jobs from $\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH}$ at each *WR*-free instant in $[t_d + \rho, t_d + x)$.

Let B_1, B_2, and B_3 be the total length of *WR*-occupied intervals within $[t_d, t_d + \rho)$, $[t_d + \rho, t_d + x)$, and $[t_d + x, t_d + x + e_\ell)$, respectively. (Recall, from (P), that $x \geq \rho$.) Let $B = B_1 + B_2 + B_3$. This is illustrated in Figure 10.

We now find a lower bound on B. Suppose first that $B \leq x - x \cdot u_\ell$. In this case, the total length of *WR*-free intervals during $[t_d, t_d + x + e_\ell)$ is $x + e_\ell - B \geq x + e_\ell - (x - x \cdot u_\ell) \geq x \cdot u_\ell + e_\ell$. Thus, by (E), T_ℓ executes for at least $x \cdot u_\ell + e_\ell$ time units after time t_d within the interval $[t_d, t_d + x + e_\ell)$. By Lemma 6, the total amount of pending work for T_ℓ at time t_d, including work due to $T_{\ell,q}$, is at most $x \cdot u_\ell + e_\ell$, and thus $T_{\ell,q}$ completes by time $t_d + x + e_\ell$ and its tardiness is at most $x + e_\ell$. This contradicts (T). In the rest of the proof, we consider the other possibility, i.e.,

$$B = x - x \cdot u_\ell + v, \tag{26}$$

where $v > 0$.

By (E), at least one processor executes a job from $\mathbf{d}$ at each *WR*-free instant within $[t_d, t_d + \rho)$ (because T_ℓ executes at each such instant). The total length of all *WR*-free intervals within $[t_d, t_d + \rho)$ is

$$L_1 = \rho - B_1. \tag{27}$$

By (N), at least a processors execute jobs from $\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH}$ at each *WR*-free instant in $[t_d + \rho, t_d + x)$. The total length of all *WR*-free intervals within $[t_d + \rho, t_d + x)$ is $x - \rho - B_2 =$

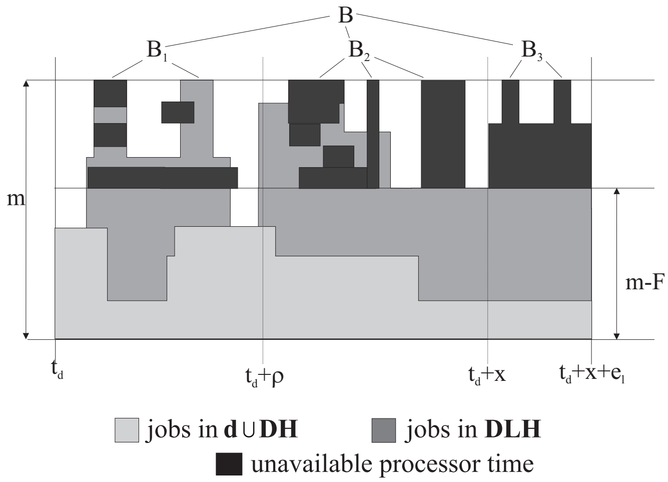


Figure 10: Structure of *WR*-occupied intervals in Lemma 15.

$x - \rho - (B - B_1 - B_3)$. Thus, the total processor allocation to jobs in **d** $\cup$ **DH** $\cup$ **DLH** in *WR*-free intervals within $[t_d + \rho, t_d + x)$ is at least

$$L_2 = a \cdot (x - \rho - B + B_1 + B_3). \tag{28}$$

By (E), at least one processor executes a job from **d** at each *WR*-free instant within $[t_d + x, t_d + x + e_\ell)$ (again, because T_ℓ executes at each such instant). The total length of all *WR*-free intervals in $[t_d + x, t_d + x + e_\ell)$ is

$$L_3 = e_\ell - B_3. \tag{29}$$

By (26), the sum of the total allocation to jobs in **d** $\cup$ **DH** $\cup$ **DLH** and the unavailable processor time in all *WR*-occupied intervals in $[t_d, t_d + x + e_\ell)$ is

$$L_b = m \cdot B = m \cdot (x - x \cdot u_\ell + v). \tag{30}$$

Let Z be the total allocation to jobs in **d** $\cup$ **DH** $\cup$ **DLH** within $[t_d, t_d + x + e_\ell)$. Because each processor is either unavailable or executes a job from **d** $\cup$ **DH** $\cup$ **DLH** at every *WR*-occupied instant and at least one processor executes T_ℓ at every *WR*-free instant, summing the lengths of all *WR*-free intervals in $[t_d, t_d + \rho)$ and $[t_d + x, t_d + x + e_\ell)$, given by (27) and (29), the allocation of jobs in **d** $\cup$ **DH** $\cup$ **DLH** in *WR*-free intervals within $[t_d + \rho, t_d + x)$, given by (28), and the total processor allocation and the unavailable processor time in *WR*-occupied intervals in $[t_d, t_d + x + e_\ell)$, given by (30), we have

$$Z + R \geq L_1 + L_2 + L_3 + L_b,$$

where R is defined earlier in Definition 14. From the inequality above, we have

$$\begin{aligned}
&Z + R \\
&\quad \geq L_1 + L_2 + L_3 + L_b \\
&\qquad \{\text{by (27), (28), (29), and (30)}\} \\
&\quad = \rho - B_1 + a \cdot (x - \rho - B + B_1 + B_3) + e_\ell - B_3 + m \cdot (x - x \cdot u_\ell + v) \\
&\qquad \{\text{setting } B' = B_1 + B_3 \text{ and } B = x - x \cdot u_\ell + v, \text{ which follows from (26)}\} \\
&\quad = e_\ell + \rho - B' + a \cdot (x \cdot u_\ell - v - \rho + B') + m \cdot (x - x \cdot u_\ell + v) \\
&\quad = e_\ell + \rho - B' + a \cdot x \cdot u_\ell - a \cdot v - a \cdot \rho + a \cdot B' + m \cdot x - m \cdot x \cdot u_\ell + m \cdot v \\
&\quad = e_\ell + (m - (m - a) \cdot u_\ell) \cdot x + (m - a) \cdot v + (a - 1) \cdot B' + (1 - a) \cdot \rho. && (31)
\end{aligned}$$

By our assumption at the beginning of the proof, $T_{\ell,q}$'s tardiness exceeds $x + e_\ell$. Because $T_{\ell,q} \in$ **d**, at time $t_d + x + e_\ell$, there is therefore unfinished work on jobs in $\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH}$. Let $Z' > 0$ be this remaining work. To find Z', we subtract $Z + R$ from $W(\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH}) + R$.

$$\begin{aligned}
&Z' \\
&= W(\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH}) + R - Z - R \\
&\qquad \{\text{by (24)}\} \\
&\leq (m - (m - a) \cdot u_\ell) \cdot x + (1 - a) \cdot \rho + e_\ell - Z - R \\
&\qquad \{\text{by (31)}\} \\
&\leq (m - (m - a) \cdot u_\ell) \cdot x + (1 - a) \cdot \rho + e_\ell - e_\ell \\
&\qquad - (m - (m - a) \cdot u_\ell) \cdot x - (m - a) \cdot v - (a - 1) \cdot B' - (1 - a) \cdot \rho \\
&= (1 - a) \cdot B' - (m - a) \cdot v.
\end{aligned}$$

By (N), $1 - a = 1 - \min(m, m - F + 1) = \max(F - m, 1 - m) \leq 0$ and $m - a = m - \min(m, m - F + 1) = \max(F - 1, 0) \geq 0$, and thus $Z' \leq 0$. Therefore, there is no work pending at time $t_d + x + e_\ell$ for jobs in $\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH}$, which implies that $T_{\ell,q}$'s tardiness is at most $x + e_\ell$, contrary to (T). $\square$

Deriving a tardiness bound. In that which follows, it is more convenient to use the following form of (23):

$$W(\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH}) + R > (m - \max(F - 1, 0) \cdot u_\ell) \cdot x + \max(F - m, 1 - m) \cdot \rho + e_\ell. \quad (32)$$

This expression is obtained from (23) by replacing $1 - a$ by $\max(F - m, 1 - m)$ and $m - a$ by $\max(F - 1, 0)$.

Earlier, in (19), we established an upper bound on $W(\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH})$. Using Definition 14, we can upper-bound R as follows.

$$
\begin{aligned}
& R \\
& \quad \{\text{by Definition 14}\} \\
& = \sum_{k=1}^{m}(x + e_\ell - \beta_k^*) \\
& \quad \{\text{by Definition 14}\} \\
& \leq \sum_{k=1}^{m}(x + e_\ell - \beta_k(x + e_\ell)) \\
& \quad \{\text{ by (1) }\} \\
& \leq \sum_{k=1}^{m}(x + e_\ell - \widehat{u_k} \cdot (x + e_\ell - \sigma_k))
\end{aligned} \tag{33}
$$

To this point, x has only been constrained to be at least ρ. We now show that if x is further constrained according to the definition below, then the tardiness of $T_{\ell,q}$ is at most $x + e_\ell$.

Definition 16. Let $x = \max(\rho, z)$, where

$$z = \frac{E_L + \max(A(\ell))}{\sum_{k=1}^{m} \widehat{u_k} - \max(F-1, 0) \cdot \max(u_\ell) - U_L}, \tag{34}$$

and $A(\ell) = e_\ell \cdot (\sum_{k=1}^{m}(1 - \widehat{u_k}) - 1) + 2\sum_{k=1}^{m} \widehat{u_k} \cdot \sigma_k + \alpha(\tau, \ell) + \min(m - F, m - 1) \cdot \rho$.

Lemma 16. *With x as defined in Definition 16, the tardiness of $T_{\ell,q}$ is at most $x + e_\ell$ provided the denominator of (34) is positive.*

Proof. Suppose that the denominator of (34) is positive and, contrary to the statement of the lemma, that the tardiness of $T_{\ell,q}$ exceeds $x + e_\ell$. By (20) and (33),

$$
\begin{aligned}
& W(\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH}) + R \\
& \quad \leq E_L + x \cdot U_L + \sum_{k=1}^{m} \widehat{u_k} \cdot \sigma_k + \alpha(\tau, \ell) + \sum_{k=1}^{m}(x + e_\ell - \widehat{u_k} \cdot (x + e_\ell - \sigma_k)) \\
& \quad = E_L + x \cdot U_L + 2\sum_{k=1}^{m} \widehat{u_k} \cdot \sigma_k + \alpha(\tau, \ell) \\
& \qquad + x \cdot \left(m - \sum_{k=1}^{m} \widehat{u_k}\right) + e_\ell \cdot \sum_{k=1}^{m}(1 - \widehat{u_k}).
\end{aligned} \tag{35}
$$

Since, by our assumption, $T_{\ell,q}$'s tardiness is greater than $x + e_\ell$ and $x \geq \rho$, by Lemma 15, (32) holds. From (35) and (32), we have

$$
\begin{aligned}
&(m - \max(F-1,0)\cdot u_\ell)\cdot x + \max(F-m, 1-m)\cdot\rho + e_\ell \\
&\quad < E_L + x\cdot U_L + 2\sum_{k=1}^{m}\widehat{u_k}\cdot\sigma_k + \alpha(\tau,\ell) \\
&\qquad + x\cdot\left(m - \sum_{k=1}^{m}\widehat{u_k}\right) + e_\ell\cdot\sum_{k=1}^{m}(1-\widehat{u_k}).
\end{aligned}
$$

Rearranging, we have

$$
\begin{aligned}
&(m - \max(F-1,0)\cdot u_\ell)\cdot x - m\cdot x + x\cdot\sum_{k=1}^{m}\widehat{u_k} - x\cdot U_L \\
&\quad < E_L + 2\sum_{k=1}^{m}\widehat{u_k}\cdot\sigma_k + \alpha(\tau,\ell) \\
&\qquad - \max(F-m, 1-m)\cdot\rho + e_\ell\cdot\left(\sum_{k=1}^{m}(1-\widehat{u_k}) - 1\right),
\end{aligned}
$$

which implies

$$
\begin{aligned}
&x\cdot\left(\sum_{k=1}^{m}\widehat{u_k} - \max(F-1,0)\cdot u_\ell - U_L\right) \\
&< E_L + 2\sum_{k=1}^{m}\widehat{u_k}\cdot\sigma_k + \alpha(\tau,\ell) + \min(m-F, m-1)\cdot\rho \\
&\quad + e_\ell\cdot\left(\sum_{k=1}^{m}(1-\widehat{u_k}) - 1\right).
\end{aligned}
$$

From this, we have

$$
\begin{aligned}
x &< \frac{E_L + A(\ell)}{\sum_{k=1}^{m}\widehat{u_k} - \max(F-1,0)\cdot u_\ell - U_L} \\
&\le \max\left(\rho, \frac{E_L + \max(A(\ell))}{\sum_{k=1}^{m}\widehat{u_k} - \max(F-1,0)\cdot\max(u_\ell) - U_L}\right),
\end{aligned}
$$

where $A(\ell)$ is defined as in Definition 16. However, this contradicts the definition of x in Definition 16. □

From the above reasoning, we have the following theorem.

Theorem 2. *The tardiness of any task T_k under a window-constrained scheduling algorithm $\mathcal{A}$ is at most $x + e_k$, where x is as in Definition 16, provided the denominator of (34) is positive.*

5 Discussion

In this section, we discuss some implications of Theorem 2 and consider some extensions and improvements to the analysis given above, such as tightening the tardiness bound for specific scheduling algorithms and processor configurations.

5.1 Relative Deadlines Different from Periods

First, note that, the definition of a prioritization function we have assumed is flexible enough to allow task systems with relative deadlines different from periods to be analyzed. By Theorem 2 and the definition of tardiness, each job $T_{i,j}$ is guaranteed to complete within $p_i + e_i + x$ time units after its release time $r_{i,j}$. We thus can compute a maximum tardiness bound with respect to an arbitrary relative deadline.

5.2 Implications of Theorem 2

The requirement to have the denominator of (34) to be positive implicitly restricts the maximum per-task utilization the system is able to accommodate without having unbounded deadline tardiness. (Recall that (2) is assumed to hold, and by our task model, $|\tau| = n$.)

Corollary 2. *Bounded tardiness is guaranteed if*

(A) $n \leq m - F$*, or*

(B) $\max(u_\ell) < \frac{\sum_{k=1}^{m} \widehat{u_k}}{\max(F-1,0)+\min(m-1,n)}$*, or*

(C) $m \geq 2$ *and* $F \leq 1$.

Proof. (A) follows trivially from the fact that if tasks do not compete for available processors, then no deadlines are missed. To prove (B), suppose that

$$\max(u_\ell) < \frac{\sum_{k=1}^{m} \widehat{u_k}}{\max(F-1,0)+\min(m-1,n)}.$$

From this, we get $\sum_{k=1}^{m} \widehat{u_k} > \max(u_\ell) \cdot \max(F-1,0) + \max(u_\ell) \cdot \min(m-1,n) \geq \max(u_\ell) \cdot \max(F-1,0) + U_L$, where the last inequality follows from Definition 11. Therefore, the denominator of (34) is positive, and by Theorem 2, the tardiness of any task in τ is bounded. As for (C), if it holds, then $\max(F-1,0) = 0$. Because, by Definition 11 and (2), $U_L < \sum_{k=1}^{m} \widehat{u_k}$, this implies that the denominator of (34) is positive. Again, by Theorem 2, the tardiness of any task in τ is bounded. $\square$

The conditions of Corollary 2 are not necessary. Depending on the processor availability pattern, it may be possible to schedule a task system for which some of the conditions from Corollary 2 do not hold yet tardiness is still bounded as the following example illustrates.

Example 12. Consider a four-processor system, where the first processor is fully available, and all other processors are available for one time unit every three time units as shown in Figure 11(a). For these processors, $\widehat{u_1} = 1$, $\widehat{u_2} = \widehat{u_3} = \widehat{u_4} = 1/3$, $\sigma_1 = 0$, and $\sigma_2 = \sigma_3 = \sigma_4 = 2$. The

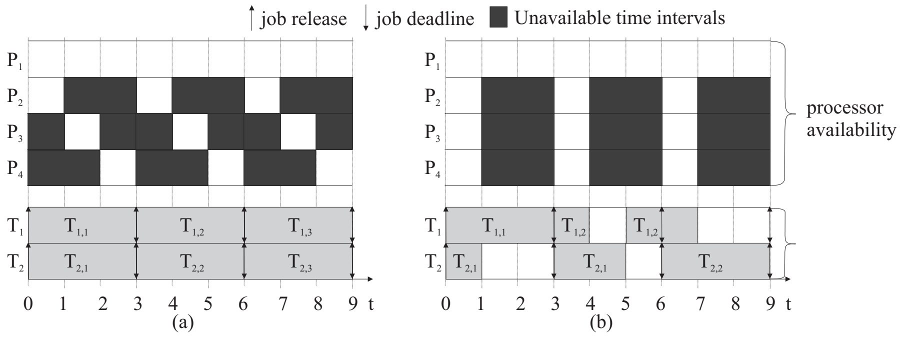


Figure 11: Task execution for different processor availability patterns.

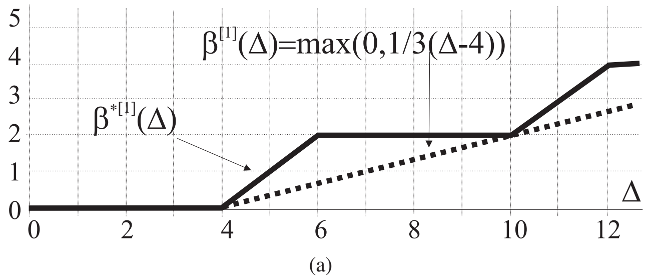
total processing capacity of the system is $\sum_{k=1}^{4} \widehat{u_k} = 1 + 3 \cdot 1/3 = 2$. Suppose that the task set $\tau = \{T_1 = (3,3), T_2 = (3,3)\}$ is scheduled. Applying Corollary 2 to this task system, we find that bounded deadline tardiness can be guaranteed if

$$\begin{aligned}\max(u_\ell) <& \frac{\sum_{k=1}^{m} \widehat{u_k}}{\max(F-1,0) + \min(m-1,n)} \\ =& \frac{2}{\max(3-1,0) + \min(3,2)} = 2/4 = 1/2.\end{aligned}$$

Though $\max(u_\ell) = 1 > 1/2$, jobs of T_1 and T_2 always meet their deadlines because at every time instant two processors are available. However, if we attempt to schedule τ on a system with the availability pattern shown in Figure 11(b), which is described by the same service functions as the pattern in Figure 11(a), we indeed will have unbounded deadline tardiness, because the arriving jobs demand six time units every three time units (assuming the job-arrival pattern continues as shown) but can utilize only four time units.

Uniform multiprocessors. Service functions as defined by (1) can also be used to describe a uniform multiprocessor platform, i.e., a platform where processors have different (constant) speeds. Particularly, a service function for which $\sigma_k = 0$ describes a processor with speed $\widehat{u_k} \leq 1$. This can be thought of as a unit-speed processor that is unavailable in infinitesimally small time intervals. The following example illustrates this approximation.

Example 13. Consider a processor that is available for two time units every six time units. The amount of available service $\beta^{*[1]}(\Delta)$ is shown in Figure 12(a) with a solid line. The service function for this processor is $\beta^{[1]}(\Delta) = \max(0, \widehat{u} \cdot (\Delta - \sigma))$, where $\widehat{u} = 1/3$ and $\sigma = 4$ as shown in Figure 12(a). The superscript "[1]" denotes that this is a first approximation of a processor with speed 1/3. It is possible to make processor availability more even, so that the processor is available for one time units every three time units. The respective service curves, $\beta^{*[2]}(\Delta)$ and $\beta^{[2]}(\Delta) = \max(0, 1/3 \cdot (\Delta - 2))$, are shown in Figure 12(b). Continuing this process, we can approximate a processor with speed $1/3$ by using the limiting service function $\beta^{lim}(\Delta) = \Delta/3$, shown in Figure 12(b), as the availability function.



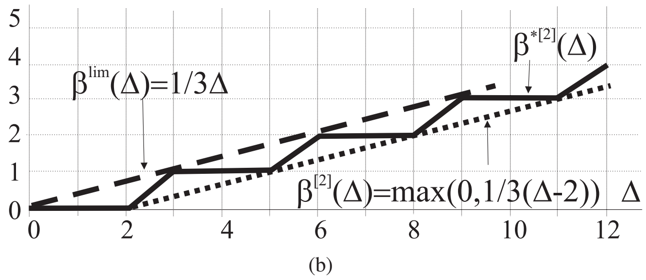


Figure 12: Approximating a slow processor with a unit-speed processor.

In order to apply Theorem 2 to a uniform multiprocessor system, task execution times have to be measured with respect to the fastest processor. The speeds of all processors must be scaled down so that the fastest processor has unit speed. When considering a system with partially-available processors in Section 4, we did not make any assumptions about the way that jobs are assigned to processors except that these processors select at most m jobs of highest priority. Therefore, Corollary 2, under which bounded tardiness is guaranteed, may be unnecessarily restrictive for uniform multiprocessors. This is because Theorem 2 treats different-speed processors and partially-available unit-speed processors in a unified fashion. In the case of a uniform multiprocessor, it may be more advantageous to assign jobs with larger utilizations or execution times or higher priorities to faster processors in order to achieve better performance. Alternatively, a partitioning scheme that restricts the set of processors where jobs may execute can be employed (e.g., see (Leontyev and Anderson, 2007b)).

5.3 Systems With Full Processor Availability

In previous work on deriving tardiness bounds for different global scheduling algorithms (Devi and Anderson, 2005; Devi et al, 2006; Leontyev and Anderson, 2007c), a system where all processors are always available for scheduling soft real-time tasks from τ was considered. In this section, we instantiate Theorem 2 for this important subcase.

If all processors are fully available to tasks in τ, then for each k, $\beta_k(\Delta) = \Delta$, $\widehat{u_k} = 1$, and $\sigma_k = F = 0$. Setting these values into Theorem 2 we have the following corollary.

Corollary 3. *If all processors are always available for scheduling the tasks in τ, then the tardiness of any task T_k under a window-constrained scheduling algorithm $\mathcal{A}$ is at most* $\max(\rho, z) + e_k$, *where*

$$z = \frac{E_L + \max(A(\ell))}{m - U_L}, \tag{36}$$

$$\textit{and } A(\ell) = -e_\ell + \alpha(\tau, \ell) + (m-1) \cdot \rho.$$

Note that the denominator of (36) is always positive since $U_L < m$ holds, by Definition 11.

5.4 Tightening the Bound for Specific Algorithms

The bounds in Theorem 2 and Corollary 3 can be improved for particular algorithms by exploiting the structure of the sets $\tau_{\mathbf{DH}}$ and $\tau_{\mathbf{DLH}}$, and the way jobs are prioritized. (Indeed, it is difficult to establish a tight bound when considering only very general properties of a scheduling algorithm.)

For example, for global EDF, $\chi(T_{i,j}, t) = d_{i,j}$, so jobs with deadlines after t_d have lower priority than $T_{\ell,q}$. Thus, $\phi_i = -p_i$, $\psi_i = 0$, and $\rho = 0$. By (13), we have $\mathbf{DH} = \emptyset$, and hence, $\mathbf{DLH} = \emptyset$, which by Definitions 7 and 13, implies $\alpha(\tau, \ell) = 0$. As a result, tardiness under global EDF for task T_k is at most

$$e_k + \max\left(0, \frac{E_L + 2\sum_{k=1}^{m} \widehat{u_k} \cdot \sigma_k + \max_{T_h \in \tau}\left(e_h \cdot \left(\sum_{k=1}^{m}(1 - \widehat{u_k}) - 1\right)\right)}{\sum_{k=1}^{m} \widehat{u_k} - \max(F-1, 0) \cdot \max(u_h) - U_L}\right), \tag{37}$$

provided the denominator of the second argument of $\max$ is positive.

If at most one processor is partially available, then $F \leq 1$, $\widehat{u_k} = 1$ for each k except one and $\sigma_k = 0$ for each k except one. From this, we have

$$\left.\begin{array}{l} \sum_{k=1}^{m} \widehat{u_k} = m - 1 + \min(\widehat{u_h}), \\ \sum_{k=1}^{m} \widehat{u_k} \cdot \sigma_k = \max(\widehat{u_h} \cdot \sigma_h), \\ \sum_{k=1}^{m}(1 - \widehat{u_k}) - 1 = -\min(\widehat{u_h}), \\ \max(F-1, 0) \cdot \max(u_h) = 0. \end{array}\right\} \tag{38}$$

Setting (38) into (37), we have a tardiness bound for task T_k under GEDF if at most one processor is partially available:

$$e_k + \frac{E_L + 2\max(\widehat{u_k} \cdot \sigma_k) - \min(\widehat{u_h}) \cdot \min_{T_h \in \tau}(e_h)}{m - 1 + \min(\widehat{u_h}) - U_L}. \tag{39}$$

Finally, if all processors are fully available, then $\max(\widehat{u_h} \cdot \sigma_h) = 0$, because $\sigma_h = 0$ for all h, and $\min(\widehat{u_h}) = 1$, and hence, by (39), the tardiness under global EDF for task T_k is at most $e_k + \frac{E_L - \min_{T_h \in \tau}(e_h)}{m - U_L}$. The latter tardiness bound was first established by Devi and Anderson in (Devi and Anderson, 2005).

Under global FIFO, jobs are prioritized by their release times, i.e., $\chi(T_{i,j}, t) = r_{i,j}$. We thus have $\phi_i = 0$ and $\psi_i = -p_i$ for each task T_i, and hence, by (10), $\rho = 0$ and $\mu = 0$. Using these values and Claim 2, we can upper-bound $\alpha(\tau, \ell)$ by $\sum_{T_i \in \tau \setminus T_\ell} e_i$. After setting these values into (36), from Corollary 3, the maximum tardiness of task T_k under global FIFO is $e_k + \frac{E_L + \max_\ell(\sum_{T_i \in \tau \setminus T_\ell} e_i - e_\ell)}{m - U_L}$. This bound is slightly worse than that obtained in (Leontyev and Anderson, 2007c), which is $e_k + \frac{E_L + \max_\ell(\sum_{T_i \,:\, p_i > p_\ell} e_i - e_\ell)}{m - U_L}$.

5.5 Non-Preemptive Execution

As shown in Section 3, the notion of window-constrained priorities allows a wide range of scheduling algorithms to be described. Some of these algorithms, e.g., global FIFO, execute jobs non-preemptively. Non-preemptivity is useful when overheads associated with rescheduling are high or when exclusive access to shared resources is needed. Some simple but efficient resource access protocols require using short non-preemptive code regions (Block et al, 2007).

Non-preemptive execution causes priority inversions when a lower-priority job is scheduled and a higher-priority job is ready but not scheduled. In this section, we show how to model non-preemptivity using window-constrained prioritization functions in a system where all processors are always available for scheduling the tasks in τ; we leave the analysis of non-preemptive execution under partial processor availability as an open problem. (Indeed, it is not clear how to deal with the situation where a processor becomes unavailable while a job is executing on it non-preemptively.) We assume some additional constraints on the task system and the scheduler.

Definition 17. We call a task system *restricted early-release* if there exists a constant $\gamma \geq 0$ such that, for each job $T_{i,j}$,

$$\epsilon_{i,j} \geq r_{i,j} - \gamma. \tag{40}$$

Definition 18. Let $\chi^A(T_{i,j}, t)$ be a prioritization function imposed by the scheduling algorithm $\mathcal{A}$. We call $\mathcal{A}$ *eventually-monotonic* if there exists a constant $M \geq 0$ such that for each job $T_{i,j}$, for all $t \geq d_{i,j} + M$ and $v \geq 0$, $\chi^A(T_{i,j}, t) \leq \chi^A(T_{i,j}, t + v)$.

From the above definition, any algorithm for which $\chi^A(T_{i,j}, t)$ is constant, e.g., global EDF, FIFO, and RM, is eventually-monotonic. Also, it is easy to verify that LLF and EDZL, as specified as in Examples 7 and 8, are eventually-monotonic. In the rest of this section, we concentrate on restricted early-release task systems scheduled under an eventually-monotonic scheduler $\mathcal{A}$ assuming that (4) holds for $\chi^A(T_{i,j}, t)$. We show how to modify the prioritization functions of $\mathcal{A}$ in a window-constrained way to ensure non-preemptive execution (if this is not ensured already).

Definition 19. Let $\phi_{max} = \max_{T_i \in \tau}(\phi_i)$, $p_{max} = \max_{T_i \in \tau}(p_i)$, and $G = \mu + \gamma + \phi_{max} + M + p_{max} + 1$.

As mentioned earlier, non-preemptive execution causes priority inversions when a low-priority job $T_{i,j}$ is scheduled and there is a ready high-priority job $T_{a,b}$ that is not scheduled. This means that $T_{i,j}$'s priority is effectively higher than that of $T_{a,b}$ for the duration of the non-preemptive region. We can explicitly model this behavior by changing prioritization functions of $\mathcal{A}$ as follows.

If a ready job $T_{i,j}$ is not executing within a non-preemptive region, then $\chi(T_{i,j}, t) = \chi^A(T_{i,j}, t)$. If $T_{i,j}$ begins executing a non-preemptive region at time t_1 and leaves that region at a later time t_2, then we "boost" its priority while it executes non-preemptively by setting $\chi(T_{i,j}, t) = r_{i,j} - G$ for all $t \in (t_1, t_2)$.

Theorem 3. (proved in the appendix) *If $\mathcal{A}$ is an eventually-monotonic scheduling algorithm and its prioritization functions are augmented as described above, then no job is preempted while executing in a non-preemptive region.*

The augmented prioritization function $\chi(T_{i,j}, t)$ remains window-constrained because $r_{i,j} - G \leq \chi(T_{i,j}, t) \leq d_{i,j} - \psi_i$ holds, where G is constant. By Corollary 3, this implies that tardiness is bounded for any restricted early-release task system under a window-constrained eventually-monotonic scheduler on m fully available processors even if the tasks in τ have non-preemptive regions.

6 Experiments

As noted in Section 5, different algorithms to which Theorem 2 applies may exhibit very different behavior in terms of tardiness. To provide a sense of how significant such differences can be, we present here the results of some experiments that we conducted to compare observed tardiness under different scheduling algorithms.

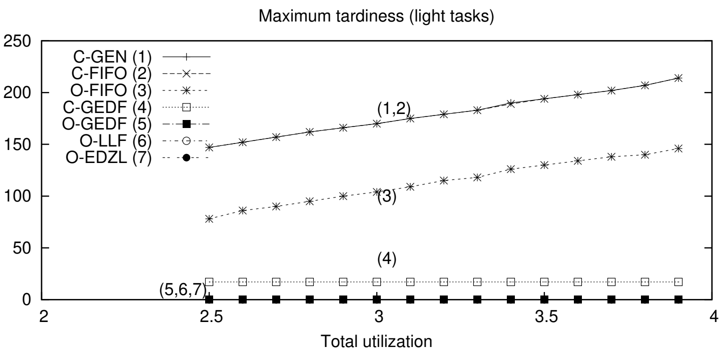
In these experiments, we examined m-procssesor systems for which task sets were randomly generated as follows. First, we generated an initial set set τ by adding tasks with integral execution times uniformly distributed over the range $[1, 10]$ and utilizations uniformly distributed over the range $[u_{min}, u_{max})$ until U_{sum} exceeded $(m + 1)/2$. We considered three utilization ranges: $[0.01, 0.05)$ (light), $[0.05, 0.5)$ (medium), and $[0.5, 0.9)$ (heavy). For each utilization range, we generated 500 independent task sets. After the initial independent task set τ was generated we incrementally added tasks to τ until its total utilization exceeded m. For each resulting task set, we produced schedules (with job releases occurring in a synchronous, periodic manner) for each of EDF, FIFO, LLF, and EDZL for $\min(20000, 20 \cdot \max(p_i))$ time units. In producing these schedules, system and scheduling overheads were taken to be negligible. For each schedule, the maximum observed tardiness was recorded.

Figure 13 shows the maximum observed tardiness values under EDF, FIFO, LLF, and EDZL as a function of U_{sum} for $m = 4$ for the light (inset (a)), medium (inset (b)), and heavy (inset (c)) utilization ranges. These observed values are denoted O-GEDF, O-FIFO, O-LLF, and O-EDZL, respectively. Additionally, for each task set, a maximum tardiness bound under LLF and EDZL was computed using Corollary 3 and assuming $\psi_i = \phi_i = 0$ for each task T_i. This bound is denoted C-GEN (it is a generalized bound, which is also applicable to FIFO and EDF). We also computed tighter bounds for EDF and FIFO, denoted C-GEDF and C-FIFO, respectively, as discussed in Section 5.4. To compute the maximum deadline tardiness under FIFO, we used the slightly improved bound mentioned earlier in Section 5.4 from (Leontyev and Anderson, 2007c). Figure 14 depicts similar data for the case $m = 8$.

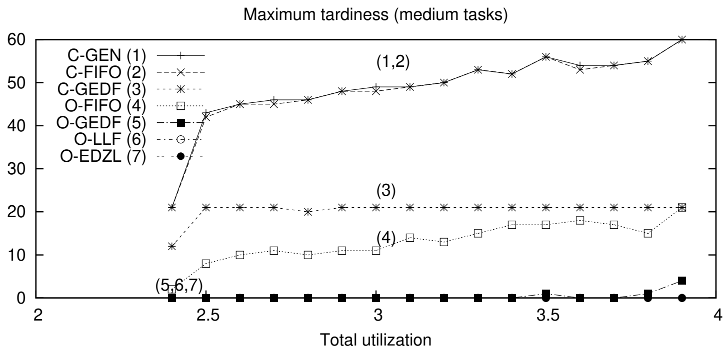
Of the four scheduling algorithms under consideration, observed tardiness under LLF and EDZL was smaller than that under FIFO and EDF (*much* smaller than under FIFO). While LLF may be impractical in reality because it preempts jobs frequently, EDZL could be a viable approach for scheduling soft real-time workloads when tardiness is allowed.

The general tardiness bound obtained using Corollary 3 is five to six times larger than the maximum task execution time, which seems quite reasonable, for the medium and heavy per-task utilization ranges (see insets (b) and (c) of Figures 13 and 14). In contrast, for the light utilization range, the maximum tardiness bound is about twenty times larger than the maximum per-task execution cost. However, the observed tardiness under FIFO for that utilization range is also quite high so it is unlikely that the general bound can be improved much (see inset (a) of Figures 13 and 14). Even though observed tardiness under LLF and EDZL is practically zero, the tardiness bound given for them by Corollary 3 (C-GEN) is very pessimistic, due to the use of a conservative estimation for $\alpha(\tau, \ell)$ (from Claim 2). Obtaining a better estimation for these algorithms is difficult, due to their dynamic nature.

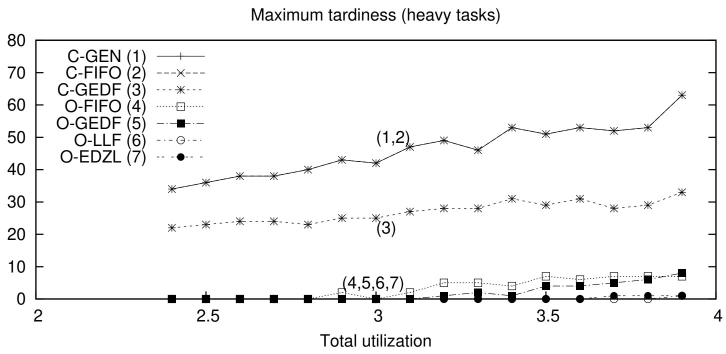
The experiments also show that the FIFO bound improvement discussed in Section 5.4 is only a slight improvement (C-GEN and C-FIFO do not differ much in any graph). In contrast, the improved bound for EDF is significantly better. (Note that the improved bound for EDF is two to three times larger than the maximum per-task execution time for all utilization ranges.) These results suggest that it might be possible to improve the tardiness bound for each algorithm (particularly



(a)



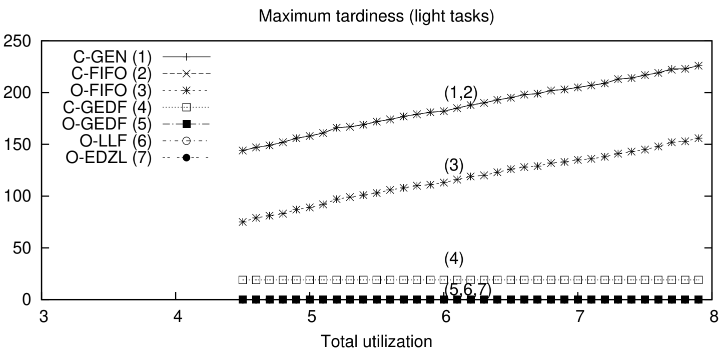
(b)



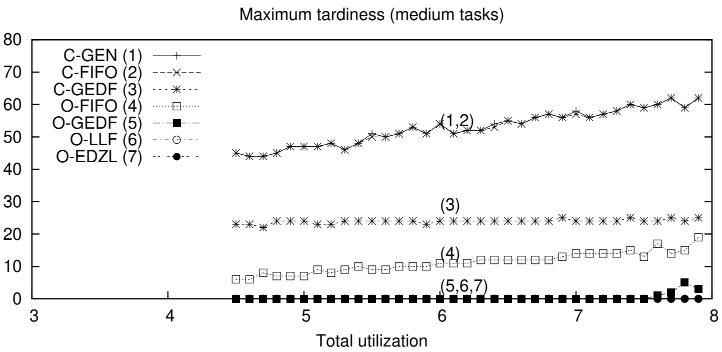
(c)

Figure 13: Maximum deadline tardiness observed and computed for **(a)** light, **(b)** medium, and **(c)** heavy per-task utilization ranges for $m = 4$ processors.

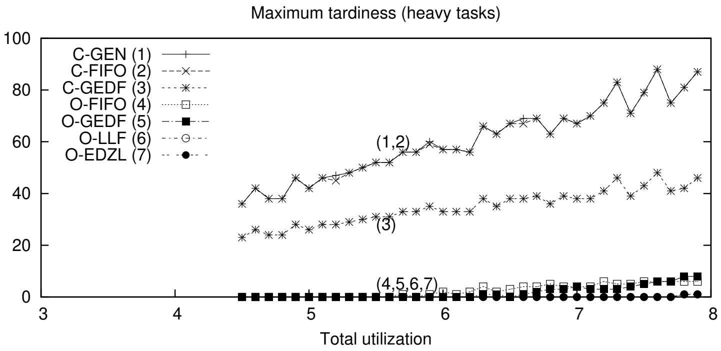
EDZL and LLF) further. We leave the development of tighter bounds for these algorithms as open problems.



(a)



(b)



(c)

Figure 14: Maximum deadline tardiness observed and computed for **(a)** light, **(b)** medium, and **(c)** heavy per-task utilization ranges for $m = 8$ processors.

7 Conclusion

We have presented a general tardiness-bound derivation that applies to a wide variety of global scheduling algorithms. Our results show that, with the exception of static-priority algorithms, most global algorithms of interest in the real-time-systems community have bounded tardiness. When considering new algorithms, the question of whether tardiness is bounded can be answered in the affirmative by simply showing that the required prioritization can be specified. Of course, a tardiness bound that is tighter than that given by our results might be possible through the use of reasoning specific to a particular algorithm. Indeed, it is difficult to obtain a very tight bound when assuming so little concerning the nature of the scheduling algorithm. Our goal in this paper was not to produce the tightest bound possible, but rather to produce a bound that could be widely applied.

Since their publication in preliminary form (Leontyev and Anderson, 2007a), the results of this paper have been used in several other efforts. For example, the ability to re-define priority points at runtime has been used in a scheduling framework that seeks to minimize cache thrashing on multicore platforms (Calandrino and Anderson, 2008). Also, the fact that per-task utilizations are not severely constrained when at most one processor is partially available (see Corollary 2 and (39)) was used in developing a hierarchical bandwidth reservation scheme for multiprocessors (Leontyev and Anderson, 2008). This scheme allows hierarchies of task groups to be scheduled with soft real-time constraints with no utilization loss (assuming overheads are negligible).

Several interesting avenues for further work exist. First, it would be interesting to investigate reactive techniques that could be applied at runtime to lessen tardiness for certain jobs by redefining priority points, as circumstances warrant. Such techniques might exploit the fact that our framework allows priority definitions to be changed rather arbitrarily at runtime. Second, our experimental results suggest that actual tardiness under EDZL is likely to be very low. It would be interesting to improve our analysis as it applies to EDZL in order to obtain a tight tardiness bound.

Acknowledgement: Work supported by a grant from Intel Corp., by NSF grants CNS 0408996, CCF 0541056, and CNS 0615197 and by ARO grant W911NF-06-1-0425.

References

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

Bisson S (2006) Azul announces 192 core Java appliance. http://www.itpro.co.uk/servers/news/99765/azul-announces-192-core-java-appliance.html

Block A, Leontyev H, Brandenburg B, Anderson J (2007) A flexible real-time locking protocol for multiprocessors. In: Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, pp 71–80

Brandenburg B, Anderson J (2007) Integrating hard/soft real-time tasks and best-effort jobs on multiprocessors. In: Proceedings of the 19th Euromicro Conference on Real-Time Systems, pp 61–70

Calandrino J, Anderson J (2008) Cache-aware real-time scheduling on multicore platforms: Heuristics and a case study. In: Proceedings of the 20th Euromicro Conference on Real-Time Systems, pp 209–308

Chakraborty S, Kunzli S, Thiele L (2003) A general framework for analysing system properties in platform-based embedded system designs. In: Proceedings of the conference on Design, Automation and Test in Europe - Volume 1

Chetto H, Chetto M (1989) Some results of the earliest deadline scheduling algorithm. IEEE Transactions on Software Engineering 15(10):1261–1269

Devi U (2006) Soft real-time scheduling on multiprocessors. PhD thesis, University of North Carolina, Chapel Hill, NC

Devi U, Anderson J (2004) Improved conditions for bounded tardiness under EPDF fair multiprocessor scheduling. In: Proceedings of 12th International Workshop on Parallel and Distributed Real-Time Systems

Devi U, Anderson J (2005) Tardiness bounds for global EDF scheduling on a multiprocessor. In: Proceedings of the 26th IEEE Real-Time Systems Symposium, pp 330–341

Devi U, Anderson J (2006) Flexible tardiness bounds for sporadic real-time task systems on multiprocessors. In: Proceedings of the 20th IEEE International Parallel and Distributed Processing Symposium, on CDROM

Devi U, Anderson J (2008) Tardiness bounds for global EDF scheduling on a multiprocessor. Real-Time Systems 38(2):133–189

Devi U, Leontyev H, Anderson J (2006) Efficient synchronization under global EDF scheduling on multiprocessors. In: Proceedings of the 18th Euromicro Conference on Real-Time Systems, pp 75–84

Farivar C (2006) Intel Developers Forum roundup: four cores now, 80 cores later. http://www.engadget.com/2006/09/26/intel-developers-forum-roundup-four-cores-now-80-cores-later/

Jeffay K, Stone D (1993) Accounting for interrupt handling costs in dynamic priority task systems. In: Proceedings of the 14^{th} IEEE Symposium on Real-Time Systems, IEEE, pp 212–221

Leontyev H, Anderson J (2007a) Generalized tardiness bounds for global multiprocessor scheduling. In: Proceedings of the 28th IEEE Real-Time Systems Symposium, pp 413–422

Leontyev H, Anderson J (2007b) Tardiness bounds for EDF scheduling on multi-speed multicore platforms. In: Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, pp 103–111

Leontyev H, Anderson J (2007c) Tardiness bounds for FIFO scheduling on multiprocessors. In: Proceedings of the 19th Euromicro Conference on Real-Time Systems, 71-80

Leontyev H, Anderson J (2008) A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees. In: Proceedings of the 20th Euromicro Conference on Real-Time Systems, pp 191–200

Liu J (2000) Real-Time Systems. Prentice Hall

Piao X, Han S, Kim H, Park M, Cho Y, Cho S (2006) Predictability of earliest deadline zero laxity algorithm for multiprocessor real-time systems. In: Proceedings of the 9th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing, pp 359–364

Shin I, Easwaran A, Lee I (2008) Hierarchical scheduling framework for virtual clustering of multiprocessors. In: Proceedings of the 20th Euromicro Conference on Real-Time Systems, pp 181–190

Stoica I, Abdel-Wahab H, Jeffay K, Baruah S, Gehrke J, Plaxton C (1996) A proportional share resource allocation algorithm for real-time, time-shared systems. In: Proceedings of the 17th IEEE Real-Time Systems Symposium, pp 288–299

Appendix

The following claim is used in proving Lemma 6 and Lemma A1.

Claim A1.
(a) *If, for job $T_{i,g}$, $r_{i,g} \geq t$, then $\mathsf{A}(T_{i,j}, 0, t, \mathcal{PS}) = 0$ for each $j \geq g$.*
(b) *If, for job $T_{i,g}$, $r_{i,g} < t \leq d_{i,g}$, then $\mathsf{A}(T_{i,j}, 0, t, \mathcal{PS}) = 0$ for each $j > g$.*

Proof. (a) follows from the fact that no job $T_{i,j}$ such that $r_{i,j} \geq t$ receives an allocation before its release time in the PS schedule $\mathcal{PS}$. If $r_{i,g} < t \leq d_{i,g}$, then $j > g$ implies that $r_{i,j} \geq r_{i,g} + p_i = d_{i,g} \geq t$, which, by (a), implies (b). □

Lemma 6: $\mathsf{lag}(T_k, t, \mathcal{S}) \leq x \cdot u_k + e_k$ *for any task* T_k *and* $t \in [0, t_d]$.

Proof. Let $d_{k,j}$ be the deadline of the earliest pending job of T_k, $T_{k,j}$, in the schedule $\mathcal{S}$ at time t. Let $\gamma_{k,j} < e_{k,j}$ be the amount of time for which $T_{k,j}$ executes before t in the schedule $\mathcal{S}$. By (6) and the selection of $T_{k,j}$,

$$\begin{aligned}
&\mathsf{lag}(T_k, t, \mathcal{S}) \\
&\quad = \sum_{h \geq 1} \mathsf{lag}(T_{k,h}, t, \mathcal{S}) \\
&\quad = \sum_{h \geq j} \mathsf{lag}(T_{k,h}, t, \mathcal{S}) \\
&\quad = \sum_{h \geq j} (\mathsf{A}(T_{k,h}, 0, t, \mathcal{PS}) - \mathsf{A}(T_{k,h}, 0, t, \mathcal{S})) \\
&\quad = \mathsf{A}(T_{k,j}, 0, t, \mathcal{PS}) - \mathsf{A}(T_{k,j}, 0, t, \mathcal{S}) + \sum_{h > j} \mathsf{A}(T_{k,h}, 0, t, \mathcal{PS}) - \sum_{h > j} \mathsf{A}(T_{k,h}, 0, t, \mathcal{S}). \qquad (41)
\end{aligned}$$

We now bound each term in the equation above. Since the earliest pending job $T_{k,j}$ executes for $\gamma_{k,j}$ time units before time t in the schedule $\mathcal{S}$,

$$\mathsf{A}(T_{k,j}, 0, t, \mathcal{S}) = \gamma_{k,j} \quad \text{and} \quad \sum_{h > j} \mathsf{A}(T_{k,h}, 0, t, \mathcal{S}) = 0. \qquad (42)$$

Bounds for the remaining terms depend on the relationship between $d_{k,j}$ and t.

Case 1: $d_{k,j} < t$. Since $T_{k,j}$ does not execute before its release time and finishes at $d_{k,j}$ in $\mathcal{PS}$, from the condition of Case 1, it follows that

$$\mathsf{A}(T_{k,j}, 0, t, \mathcal{PS}) = \mathsf{A}(T_{k,j}, r_{k,j}, d_{k,j}, \mathcal{PS}) = e_{k,j}. \qquad (43)$$

Since the job $T_{k,j+1}$ cannot commence execution in $\mathcal{PS}$ earlier than time $d_{k,j}$,

$$\sum_{h > j} \mathsf{A}(T_{k,h}, 0, t, \mathcal{PS}) \leq u_k \cdot (t - d_{k,j}). \qquad (44)$$

Setting (42), (43), and (44) into (41), we get

$$\mathsf{lag}(T_k, t, \mathcal{S}) \leq e_{k,j} - \gamma_{k,j} + u_k \cdot (t - d_{k,j}). \qquad (45)$$

Because $d_{k,j} < t \leq t_d$ holds, by Property (P), $T_{k,j}$ has tardiness at most $x + e_k$. Let $compl(T_{k,j}, t)$ be the length of the interval after time t where $T_{k,j}$ is pending. Then, $t + compl(T_{k,j}, t) \leq d_{k,j} + x + e_k$, and hence,

$$t - d_{k,j} \leq x + e_k - compl(T_{k,j}, t). \tag{46}$$

Because $T_{k,j}$ executes for $\gamma_{k,j}$ time units before time t, $compl(T_{k,j}, t) \geq e_{k,j} - \gamma_{k,j}$. Setting the last inequality into (46), we get $t - d_{i,j} \leq x + e_k - e_{k,j} + \gamma_{k,j}$. From (45), we therefore have

$$\begin{aligned}
&\mathsf{lag}(T_k, t, \mathcal{S}) \\
&\qquad \leq e_{k,j} - \gamma_{k,j} + u_k \cdot (t - d_{k,j}) \\
&\qquad \leq e_{k,j} - \gamma_{k,j} + u_k \cdot (x + e_k - e_{k,j} + \gamma_{k,j}) \\
&\qquad = e_{k,j} + u_k \cdot x + \gamma_{k,j} \cdot (u_k - 1) + u_k \cdot (e_k - e_{k,j}) \\
&\qquad \leq u_k \cdot x + e_{k,j} + u_k \cdot (e_k - e_{k,j}) \\
&\qquad = u_k \cdot x + e_{k,j} \cdot (1 - u_k) + u_k \cdot e_k \\
&\qquad \quad \{\text{maximized if } e_{k,j} = e_k\} \\
&\qquad \leq u_k \cdot x + e_k.
\end{aligned}$$

Case 2: $d_{k,j} \geq t$. In this case,

$$\mathsf{A}(T_{k,j}, 0, t, \mathcal{PS}) = \mathsf{A}(T_{k,j}, r_{k,j}, t, \mathcal{PS}) \leq u_{k,j} \cdot (t - r_{k,j}) \leq u_k \cdot (d_{k,j} - r_{k,j}) = u_k \cdot p_k = e_k. \tag{47}$$

By the condition of Case 2, for any job $T_{k,h}$ such that $h > j$, $r_{k,h} \geq t$ holds, and hence, by Claim A1,

$$\sum_{h>j} \mathsf{A}(T_{k,h}, 0, t, \mathcal{PS}) = 0. \tag{48}$$

Setting (42), (47), and (48) into (41) we get

$$\mathsf{lag}(T_k, t, \mathcal{S}) \leq e_{k,j} - \gamma_{k,j} \leq e_k + u_k \cdot x,$$

where the latter inequality trivially follows, since $x \geq \rho \geq 0$ (see (P)). The lemma follows. □

Lemma 7: $\mathsf{LAG}(\mathbf{d}, t_d, \mathcal{S}) \leq \mathsf{LAG}(\mathbf{d}, t_n, \mathcal{S}) + \sum_{T_i \in \tau_{\mathbf{DH}}} \delta_i + \sum_{k=1}^{m} \widehat{u_k} \cdot \sigma_k$.

Proof. By (8),

$$\mathsf{LAG}(\mathbf{d}, t_d, \mathcal{S}) = \mathsf{LAG}(\mathbf{d}, t_n, \mathcal{S}) + \mathsf{A}(\mathbf{d}, t_n, t_d, \mathcal{PS}) - \mathsf{A}(\mathbf{d}, t_n, t_d, \mathcal{S}). \tag{49}$$

To compute $\mathsf{A}(\mathbf{d}, t_n, t_d, \mathcal{PS}) - \mathsf{A}(\mathbf{d}, t_n, t_d, \mathcal{S})$, we split $[t_n, t_d)$ into b non-overlapping intervals $[t_{p_s}, t_{q_s})$, $1 \leq s \leq b$, such that $t_n = t_{p_1}$, $t_{q_{s-1}} = t_{p_s}$, and $t_{q_b} = t_d$. These intervals are defined so that, for each interval $[t_{p_s}, t_{q_s})$, if processor h is unavailable at time $t \in [t_{p_s}, t_{q_s})$, then it is unavailable throughout the entire interval $[t_{p_s}, t_{q_s})$. We further assume that each interval $[t_{p_s}, t_{q_s})$ is defined so that if a job $T_{k,j}$ executes at some point in the interval in schedule $\mathcal{S}$, then it executes continuously throughout the interval in $\mathcal{S}$. Note that such a job $T_{k,j}$ does not necessarily execute continuously throughout $[t_n, t_d)$. The allocation difference for $\mathbf{d}$ throughout the interval $[t_n, t_d)$ is thus

$$\mathsf{A}(\mathbf{d}, t_n, t_d, \mathcal{PS}) - \mathsf{A}(\mathbf{d}, t_n, t_d, \mathcal{S}) = \sum_{s=1}^{b} \left(\mathsf{A}(\mathbf{d}, t_{p_s}, t_{q_s}, \mathcal{PS}) - \mathsf{A}(\mathbf{d}, t_{p_s}, t_{q_s}, \mathcal{S}) \right).$$

We now bound the allocation difference in the PS schedule $\mathcal{PS}$ and the schedule $\mathcal{S}$ across each of the intervals $[t_{p_s}, t_{q_s})$. The sum of these bounds gives us a bound on the total allocation difference throughout $[t_n, t_d)$. By the definition of a PS schedule,

$$\mathsf{A}(\mathbf{d}, t_{p_s}, t_{q_s}, \mathcal{PS}) \leq U_{sum} \cdot (t_{q_s} - t_{p_s}). \tag{50}$$

For each interval $[t_{p_s}, t_{q_s})$, we let $\alpha_s \subseteq \tau_{\mathbf{DH}}$ denote those tasks that execute their jobs in **DH** continuously throughout $[t_{p_s}, t_{q_s})$ in the schedule $\mathcal{S}$. Due to selection of t_n, within each interval $[t_{p_s}, t_{q_s})$ in schedule $\mathcal{S}$ two alternatives are possible:

1. m available processors are occupied by tasks with ready jobs in **d**.
2. Some tasks with ready jobs in **d** do not execute because some processors are unavailable and/or other available processors execute tasks in α_s. (Note that, by Lemma 5, jobs in **DLH** and **DLL** cannot execute at time instants when there are ready unscheduled jobs in **d**.)

For each interval $[t_{p_s}, t_{q_s})$, we define κ_s to be the number of unavailable processors in that interval. The number of available processors in $[t_{p_s}, t_{q_s})$ is thus $m - \kappa_s$. Therefore,

$$\begin{aligned}
&\mathsf{A}(\mathbf{d}, t_{p_s}, t_{q_s}, \mathcal{S}) \\
&\qquad = (t_{q_s} - t_{p_s}) \cdot (m - |\alpha_s| - \kappa_s) \\
&\qquad = -(t_{q_s} - t_{p_s}) \cdot |\alpha_s| + (t_{q_s} - t_{p_s}) \cdot (m - \kappa_s).
\end{aligned} \tag{51}$$

Subtracting (51) from (50), we get

$$\begin{aligned}
&\mathsf{A}(\mathbf{d}, t_{p_s}, t_{q_s}, \mathcal{PS}) - \mathsf{A}(\mathbf{d}, t_{p_s}, t_{q_s}, \mathcal{S}) \\
&\qquad \leq (t_{q_s} - t_{p_s}) \cdot U_{sum} - (-(t_{q_s} - t_{p_s}) \cdot |\alpha_s| + (t_{q_s} - t_{p_s}) \cdot (m - \kappa_s)) \\
&\qquad = (t_{q_s} - t_{p_s}) \cdot U_{sum} + (t_{q_s} - t_{p_s}) \cdot |\alpha_s| - (t_{q_s} - t_{p_s}) \cdot (m - \kappa_s) \\
&\qquad = (t_{q_s} - t_{p_s}) \cdot U_{sum} + (t_{q_s} - t_{p_s}) \cdot \sum_{T_i \in \alpha_s} 1 - (t_{q_s} - t_{p_s}) \cdot (m - \kappa_s).
\end{aligned} \tag{52}$$

Summing (52) over all intervals $[t_{p_s}, t_{q_s})$, we have

$$\begin{aligned}
&\mathsf{A}(\mathbf{d}, t_n, t_d, \mathcal{PS}) - \mathsf{A}(\mathbf{d}, t_n, t_d, \mathcal{S}) \\
&\quad \leq \sum_{s=1}^{b} (t_{q_s} - t_{p_s}) \cdot U_{sum} + \sum_{s=1}^{b} \sum_{T_i \in \alpha_s} (t_{p_s} - t_{q_s}) - \sum_{s=1}^{b} (t_{q_s} - t_{p_s}) \cdot (m - \kappa_s) \\
&\quad = (t_d - t_n) \cdot U_{sum} + \sum_{s=1}^{b} \sum_{T_i \in \alpha_s} (t_{p_s} - t_{q_s}) - \sum_{s=1}^{b} (t_{q_s} - t_{p_s}) \cdot (m - \kappa_s).
\end{aligned} \tag{53}$$

For each task $T_i \in \tau_{\mathbf{DH}}$, the sum of the lengths of the intervals $[t_{p_s}, t_{q_s})$, in which jobs of T_i from **DH** execute continuously before time t_d is at most δ_i (see Definition 9). Thus,

$$\sum_{s=1}^{b} \sum_{T_i \in \alpha_s} (t_{p_s} - t_{q_s}) \leq \sum_{T_i \in \tau_{\mathbf{DH}}} \delta_i. \tag{54}$$

Now consider $\sum_{s=1}^{b}(t_{q_s} - t_{p_s}) \cdot (m - \kappa_s)$. Since κ_s is the number of unavailable processors within the interval $[t_{p_s}, t_{q_s})$, $(m - \kappa_s) \cdot (t_{q_s} - t_{p_s})$ is the amount of processor time available to tasks in τ within $[t_{p_s}, t_{q_s})$. The sum of these times for all the intervals $[t_{p_s}, t_{q_s})$ is at least the total processor time guaranteed within $[t_n, t_d)$, because each processor is either unavailable or executes a task from τ within $[t_{p_s}, t_{q_s})$. Thus,

$$\sum_{s=1}^{b}(m - \kappa_s) \cdot (t_{q_s} - t_{p_s}) \geq \sum_{k=1}^{m} \beta_k (t_d - t_n). \tag{55}$$

By (1) and (55), we have

$$\sum_{s=1}^{b}(m - \kappa_s) \cdot (t_{q_s} - t_{p_s}) \geq \sum_{k=1}^{m} \beta_k (t_d - t_n) \geq \sum_{k=1}^{m} \widehat{u_k} \cdot (t_d - t_n - \sigma_k). \tag{56}$$

Substituting (54) and (56) into (53), we have

$$\begin{aligned}
&\mathsf{A}(\mathbf{d}, t_n, t_d, \mathcal{PS}) - \mathsf{A}(\mathbf{d}, t_n, t_d, \mathcal{S}) \\
&\qquad \leq (t_d - t_n) U_{sum} + \sum_{T_i \in \tau_{\mathbf{DH}}} \delta_i - \sum_{k=1}^{m} \widehat{u_k} \cdot (t_d - t_n - \sigma_k) \\
&\qquad = (t_d - t_n)\left(U_{sum} - \sum_{k=1}^{m} \widehat{u_k}\right) + \sum_{T_i \in \tau_{\mathbf{DH}}} \delta_i + \sum_{k=1}^{m} \widehat{u_k} \cdot \sigma_k \\
&\qquad\quad \{ \text{ by (2) } \} \\
&\qquad \leq \sum_{k=1}^{m} \widehat{u_k} \cdot \sigma_k + \sum_{T_i \in \tau_{\mathbf{DH}}} \delta_i.
\end{aligned} \tag{57}$$

By (57) and (49), the lemma follows. □

The following definition and Lemmas A1 and A2 and used in proving Lemma 8.

Definition 20. Let $\xi = \{T_i :: \exists T_{i,j} \in \mathbf{d}$ such that $T_{i,j}$ is ready at t_n^- in schedule $\mathcal{S}\}$.

Lemma A1. *If* $T_i \notin \xi$, *then* $\sum_{T_{i,j} \in \mathbf{d}} \mathsf{lag}(T_{i,j}, t_n, \mathcal{S}) \leq 0$.

Proof. Consider task $T_i \notin \xi$ at time instant t_n^-. Let $T_{i,g}$ be the latest job such that $r_{i,g} < t_n$. Then $t_n \leq r_{i,j}$ for each $j > g$. By Claim A1 (b),

$$\sum_{T_{i,j} \,::\, T_{i,j} \in \mathbf{d} \wedge j > g} \mathsf{A}(T_{i,j}, 0, t_n, \mathcal{PS}) = 0. \tag{58}$$

Also, in the PS schedule $\mathcal{PS}$, $T_{i,g}$'s allocation cannot be larger than its actual execution time $e_{i,g}$.

$$\mathsf{A}(T_{i,g}, 0, t_n, \mathcal{PS}) \leq e_{i,g}. \tag{59}$$

Because $T_i \notin \xi$, all jobs $T_{i,j}$ such that $T_{i,j} \in \mathbf{d}$ and $j < g$ complete by time t_n^- in both schedules $\mathcal{S}$ and $\mathcal{PS}$, and hence,

$$\mathsf{A}(T_{i,j}, 0, t_n, \mathcal{PS}) = \mathsf{A}(T_{i,j}, 0, t_n, \mathcal{S}) \text{ for each } j < g \text{ and } T_{i,j} \in \mathbf{d}. \tag{60}$$

Also, all jobs with eligibility times at most t_n, including job $T_{i,g}$, for which $\epsilon_{i,g} \le r_{i,g} < t_n$, complete by t_n in schedule $\mathcal{S}$. We thus have

$$\mathsf{A}(T_{i,g}, 0, t_n, \mathcal{S}) = e_{i,g}. \tag{61}$$

By (7), we have

$$\begin{aligned}
&\sum_{T_{i,j} \in \mathbf{d}} \mathsf{lag}(T_{i,j}, t_n, \mathcal{S}) \\
&\qquad = \sum_{T_{i,j} \in \mathbf{d}} (\mathsf{A}(T_{i,j}, 0, t_n, \mathcal{PS}) - \mathsf{A}(T_{i,j}, 0, t_n, \mathcal{S})) \\
&\qquad\quad \{\text{by (60)}\} \\
&\qquad = \sum_{T_{i,j} \,::\, T_{i,j} \in \mathbf{d} \wedge j \ge g} (\mathsf{A}(T_{i,j}, 0, t_n, \mathcal{PS}) - \mathsf{A}(T_{i,j}, 0, t_n, \mathcal{S})) \\
&\qquad\quad \{\text{by (59) and (61)}\} \\
&\qquad \le \sum_{T_{i,j} \,::\, T_{i,j} \in \mathbf{d} \wedge j > g} \mathsf{A}(T_{i,j}, 0, t_n, \mathcal{PS}) - \sum_{T_{i,j} \,::\, T_{i,j} \in \mathbf{d} \wedge j > g} \mathsf{A}(T_{i,j}, 0, t_n, \mathcal{S}) \\
&\qquad\quad \{\text{by (58)}\} \\
&\qquad \le - \sum_{T_{i,j} \,::\, T_{i,j} \in \mathbf{d} \wedge j > g} \mathsf{A}(T_{i,j}, 0, t_n, \mathcal{S}) \\
&\qquad \le 0.
\end{aligned}$$

The lemma follows. □

Lemma A2. *If* $T_i \in \xi$, *then* $\sum_{T_{i,j} \in \mathbf{d}} \mathsf{lag}(T_{i,j}, t_n, \mathcal{S}) \le \mathsf{lag}(T_i, t_n, \mathcal{S})$.

Proof. Because $T_i \in \xi$, there exists a job $T_{i,g}$ such that $d_{i,g} \le t_d$ and $T_{i,g}$ is pending at t_n^-. Because jobs of T_i execute sequentially, jobs of T_i with deadlines after $d_{i,g}$ do not execute before time t_n, and hence,

$$\mathsf{A}(T_{i,j}, 0, t_n, \mathcal{S}) = 0 \text{ for each job } T_{i,j} \notin \mathbf{d}. \tag{62}$$

We therefore have,

$$\begin{aligned}
&\mathsf{lag}(T_i, t_n, \mathcal{S}) \\
&\qquad \{\text{by (6)}\} \\
&\quad = \sum_{j \ge 1} (\mathsf{A}(T_{i,j}, 0, t_n, \mathcal{PS}) - \mathsf{A}(T_{i,j}, 0, t_n, \mathcal{S})) \\
&\quad = \sum_{(j \ge 1) \wedge T_{i,j} \in \mathbf{d}} (\mathsf{A}(T_{i,j}, 0, t_n, \mathcal{PS})
\end{aligned}$$

$$
\begin{aligned}
&- \mathsf{A}(T_{i,j}, 0, t_n, \mathcal{S})) + \sum_{(j \geq 1) \wedge T_{i,j} \notin \mathbf{d}} (\mathsf{A}(T_{i,j}, 0, t_n, \mathcal{PS}) - \mathsf{A}(T_{i,j}, 0, t_n, \mathcal{S}))\\
&\{\text{by (7)}\}\\
= &\sum_{T_{i,j} \in \mathbf{d}} \mathsf{lag}(T_{i,j}, t_n, \mathcal{S}) + \sum_{T_{i,j} \notin \mathbf{d}} (\mathsf{A}(T_{i,j}, 0, t_n, \mathcal{PS}) - \mathsf{A}(T_{i,j}, 0, t_n, \mathcal{S}))\\
&\{\text{by (62)}\}\\
= &\sum_{T_{i,j} \in \mathbf{d}} \mathsf{lag}(T_{i,j}, t_n, \mathcal{S}) + \sum_{T_{i,j} \notin \mathbf{d}} \mathsf{A}(T_{i,j}, 0, t_n, \mathcal{PS})\\
\geq &\sum_{T_{i,j} \in \mathbf{d}} \mathsf{lag}(T_{i,j}, t_n, \mathcal{S}).
\end{aligned}
$$
□

Lemma 8: $\mathsf{LAG}(\mathbf{d}, t_n, \mathcal{S}) \leq E_L + x \cdot U_L$.

Proof. If $t_n = 0$, then $\mathsf{LAG}(\mathbf{d}, t_n, \mathcal{S}) = 0$ and the lemma holds trivially, so assume that $t_n > 0$. By Definition 10 and Definition 20, all tasks in ξ execute at t_n^-, and hence, $|\xi| \leq m - 1$. Therefore,

$$
\begin{aligned}
&\mathsf{LAG}(\mathbf{d}, t_n, \mathcal{S})\\
&\quad \{\text{by (7)}\}\\
&= \sum_{T_{i,j} \in \mathbf{d}} \mathsf{lag}(T_{i,j}, t_n, \mathcal{S})\\
&= \sum_{T_i \in \xi} \sum_{T_{i,j} \in \mathbf{d}} \mathsf{lag}(T_{i,j}, t_n, \mathcal{S}) + \sum_{T_i \notin \xi} \sum_{T_{i,j} \in \mathbf{d}} \mathsf{lag}(T_{i,j}, t_n, \mathcal{S})\\
&\quad \{\text{by Lemma A1}\}\\
&\leq \sum_{T_i \in \xi} \sum_{T_{i,j} \in \mathbf{d}} \mathsf{lag}(T_{i,j}, t_n, \mathcal{S})\\
&\quad \{\text{by Lemma A2}\}\\
&\leq \sum_{T_i \in \xi} \mathsf{lag}(T_i, t_n, \mathcal{S})\\
&\quad \{\text{by Lemma 6}\}\\
&\leq \sum_{T_i \in \xi} (x \cdot u_i + e_i)\\
&\quad \{\text{because } |\xi| \leq m - 1\}\\
&\leq E_L + x \cdot U_L.
\end{aligned}
$$
□

The following claim is used in proving Lemmas 10 and 11.

Claim A2. *If $T_{i,k} \in$ **DH**, then $\chi(T_{i,k}, t') \leq t_d + \psi_a$ for some $a \neq i$ and time t'.*

Proof. If $T_{i,k} \in$ **DH**, then, by (13), there exists $T_{a,b} \in \mathbf{d}$ such that $a \neq i$ and $\neg\mathsf{LP}(T_{i,k}, T_{a,b})$ holds.

By (11), there exists t' such that

$$\begin{aligned}
&\chi(T_{i,k}, t') \\
&\quad \leq d_{a,b} + \psi_a \\
&\qquad \{\text{because } T_{a,b} \in \mathbf{d}\text{, by (12), } d_{a,b} \leq t_d\ \} \\
&\quad \leq t_d + \psi_a.
\end{aligned}$$

The claim follows. □

Lemma 10. *If $T_{i,k} \in \mathbf{d} \cup \mathbf{DH}$, then $r_{i,k} \leq t_d + \rho$.*

Proof. Because sets **d** and **DH** are disjoint we consider two cases.

Case 1: $T_{i,k} \in$ **d.** In this case, $r_{i,k} \leq d_{i,k} \leq t_d \leq t_d + \rho$, since $\rho \geq 0$.

Case 2: $T_{i,k} \in$ **DH.** By the condition of Case 2 and Claim A2, there exists $a \neq i$ and t' such that $\chi(T_{i,k}, t') \leq t_d + \psi_a$. We thus have, for time t',

$$\begin{aligned}
&r_{i,k} \\
&\qquad \{\text{by (4)}\} \\
&\quad \leq \chi(T_{i,k}, t') + \phi_i \\
&\quad \leq t_d + \psi_a + \phi_i \\
&\qquad \{\text{by (10)}\} \\
&\quad \leq t_d + \rho.
\end{aligned}$$

The lemma follows. □

Lemma 11. *If $T_{i,k} \in$ **DLH**, then $r_{i,k} \leq t_d + \rho + \mu$.*

Proof. Suppose that $T_{i,k} \in$ **DLH**. Then, by (15), there exists $T_{a,b} \in$ **DH** such that $a \neq i$ and $\neg\mathsf{LP}(T_{i,k}, T_{a,b})$ holds. The latter implies that $\chi(T_{i,k}, t') \leq d_{a,b} + \psi_a$ holds for some time t'. We thus have, for time t',

$$\begin{aligned}
&r_{i,k} \\
&\qquad \{\text{by (4)}\} \\
&\quad \leq \chi(T_{i,k}, t') + \phi_i \\
&\quad \leq d_{a,b} + \psi_a + \phi_i \\
&\quad = r_{a,b} + p_a + \psi_a + \phi_i \\
&\qquad \{\text{by (10)}\} \\
&\quad \leq r_{a,b} + \mu \\
&\qquad \{\text{by Lemma 10}\} \\
&\quad \leq t_d + \rho + \mu.
\end{aligned}$$

□

Theorem 3. *If $\mathcal{A}$ is an eventually-monotonic scheduling algorithm and its prioritization functions are augmented as described above, then no job is preempted while executing in a non-preemptive region.*

Proof. Suppose, contrary to the statement of the theorem, that job $T_{k,h}$ begins executing a non-preemptive region at time t_1 and, while still within that region, is preempted at time t_p by job $T_{a,b}$ that is either ready but not scheduled at time t_p^- or becomes eligible at t_p. Because $T_{k,h}$ cannot be scheduled earlier than $\epsilon_{k,h}$, we have

$$\epsilon_{k,h} \leq t_1 < t_p^- < t_p. \tag{63}$$

According to the priority augmentation rules, $\chi(T_{a,b}, t_p) = \chi^A(T_{a,b}, t_p)$. Below, we show that either $\chi^A(T_{a,b}, t_p) > r_{k,h} - G = \chi(T_{k,h}, t_p)$ holds or the tie-breaking between jobs $T_{a,b}$ and $T_{k,h}$ at times t_p^- and t_p is not consistent, and hence, job $T_{a,b}$ cannot be scheduled at time t_p as assumed. Let

$$r_c = r_{k,h} - \mu - \gamma - p_{max} - M. \tag{64}$$

Two cases are possible, based upon the release time of $T_{a,b}$.

Case 1: $r_c \leq r_{a,b}$**.** In this case,

$$\begin{aligned}
&\chi^A(T_{a,b}, t_p) \\
&\qquad \{\text{by (4)}\} \\
&\quad \geq r_{a,b} - \phi_a \\
&\qquad \{\text{by the condition of Case 1}\} \\
&\quad \geq r_c - \phi_a \\
&\qquad \{\text{by (64)}\} \\
&\quad = r_{k,h} - \mu - \gamma - p_{max} - M - \phi_a \\
&\qquad \{\text{by Definition 19}\} \\
&\quad > r_{k,h} - G
\end{aligned}$$

Case 2: $r_{a,b} < r_c$**.** In this case, we can show that $d_{a,b} + M < \epsilon_{k,h}$ holds.

$$\begin{aligned}
&d_{a,b} + M \\
&\quad = r_{a,b} + p_a + M \\
&\qquad \{\text{by the condition of Case 2}\} \\
&\quad < r_c + p_a + M \\
&\qquad \{\text{by (64)}\} \\
&\quad = r_{k,h} - \mu - \gamma - p_{max} - M + p_a + M \\
&\quad \leq r_{k,h} - \mu - \gamma \\
&\qquad \{\text{by (40)}\} \\
&\quad \leq \epsilon_{k,h} - \mu \\
&\qquad \{\text{because } \mu \geq 0 \text{ (see (10))}\} \\
&\quad \leq \epsilon_{k,h}
\end{aligned} \tag{65}$$

Two subcases are possible, depending on whether job $T_{a,b}$ is ready at time t_p^-.

Subcase 1: *$T_{a,b}$ is not ready at time t_p^-.* In this case, by the selection of $T_{a,b}$, it becomes eligible at t_p, and hence, by (63),

$$\epsilon_{k,h} < t_p = \epsilon_{a,b}. \tag{66}$$

We can lower-bound $\chi^A(T_{a,b}, t_p)$ as follows.

$$\begin{aligned}
&\chi^A(T_{a,b}, t_p) \\
&\quad \{\text{by (4)}\} \\
&\geq r_{a,b} - \phi_a \\
&\geq \epsilon_{a,b} - \phi_a \\
&\quad \{\text{by (66)}\} \\
&> \epsilon_{k,h} - \phi_a \\
&\quad \{\text{by (40)}\} \\
&\geq r_{k,h} - \gamma - \phi_a \\
&\quad \{\text{by Definition 19}\} \\
&> r_{k,h} - G
\end{aligned}$$

Subcase 2: *$T_{a,b}$ is ready at time t_p^-.* In this case, because $T_{k,h}$ is scheduled at t_p^- and $T_{a,b}$ is not scheduled, we have

$$r_{k,h} - G = \chi(T_{k,h}, t_p^-) \leq \chi(T_{a,b}, t_p^-) = \chi^A(T_{a,b}, t_p^-). \tag{67}$$

The latter equality holds because $T_{a,b}$ is not scheduled at t_p^- and thus is not executing non-preemptively then. By (63) and (65), $d_{a,b} + M < t_p^- < t_p$. Therefore, by Definition 18, we have

$$\chi^A(T_{a,b}, t_p^-) \leq \chi^A(T_{a,b}, t_p). \tag{68}$$

By (67) and (68), we have $r_{k,h} - G \leq \chi^A(T_{a,b}, t_p)$. If $r_{k,h} - G < \chi^A(T_{a,b}, t_p)$ holds, then $T_{a,b}$ cannot preempt $T_{k,h}$. If $r_{k,h} - G = \chi^A(T_{a,b}, t_p)$, then by (67) and (68), we have $r_{k,h} - G = \chi^A(T_{a,b}, t_p^-)$, and hence, the tie-breaking between jobs $T_{a,b}$ and $T_{k,h}$ is not consistent. $\square$