Tardiness Bounds under Global EDF Scheduling on a Multiprocessor*

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

Abstract

We consider the scheduling of a sporadic real-time task system on an identical multiprocessor. Though Pfair algorithms are theoretically optimal for such task systems, in practice, their runtime overheads can significantly reduce the amount of useful work that is accomplished. On the other hand, if *all* deadlines need to be met, then every known non-Pfair algorithm requires restrictions on total system utilization that can approach approximately 50% of the available processing capacity. This may be overkill for soft real-time systems, which can tolerate occasional or bounded deadline misses (*i.e.*, bounded tardiness). In this paper we derive tardiness bounds under preemptive and non-preemptive global EDF when the total system utilization is not restricted, except that it not exceed the available processing capacity. Hence, processor utilization can be improved for soft real-time systems on multiprocessors. Our tardiness bounds depend on the total system utilization and per-task utilizations and execution costs — the lower these values, the lower the tardiness bounds. As a final remark, we note that global EDF may be superior to partitioned EDF for multiprocessor-based soft real-time systems in that the latter does not offer any scope to improve system utilization even if bounded tardiness can be tolerated.

*Work supported by NSF grants CCR 0204312, CNS 0309825, CNS 0408996, CCF 0541056, and CNS 0615197, and by ARO grant W911NF-06-1-0425. This work was presented in preliminary form at the 26^{th} IEEE Real-Time Systems Symposium [Devi and Anderson, 2005].

1 Introduction

One evident trend in the design of both general-purpose and embedded processing systems is the increasing use of tightly-coupled multiprocessors. This is evidenced by the availability of affordable symmetric multiprocessor platforms (SMPs) and the emergence of multicore architectures, which have multiple processing units on a single die. Another trend of relevance is the proliferation of real-time applications that have workloads that necessitate a multiprocessor platform and for which soft real-time guarantees are sufficient. Examples include systems that track people and machines, virtual-reality systems, multimedia systems, systems that host web-sites, and some signal-processing systems. Due to the above trends, multiprocessor-based software designs, including soft real-time system designs, are of growing importance.

Motivation. In this paper, we consider the scheduling of soft real-time task systems on multiprocessors. The soft real-time tasks that we consider are modeled as *periodic* or *sporadic* tasks that can tolerate bounded *tardiness*, *i.e.*, deadline misses by bounded amounts. As an example, consider a video decoding task that decodes a video stream at the rate of 30 frames per second in a multimedia system. While it is desirable that the task decode every frame within 33.3 ms, a tardiness of a few milliseconds will not compromise the quality of the output if the rate of decoding is still 30 frames per second over reasonably long intervals of time. Tardiness may add to jitter in frame completion times, but it is unlikely that jitter of the order of a few tens of milliseconds will be perceptible to the human eye. Similarly, tardiness will add to the buffering needs of a task, but should be reasonable, if maximum tardiness is reasonably bounded and a system designer is able to choose a tardiness value that balances the processing and memory needs of the system. A similar reasoning can be applied to see that the other example soft real-time applications mentioned in the prior paragraph can also tolerate a small, constant tardiness that is independent of elapsed time. Hence, scheduling algorithms that ensure bounded tardiness and that can be applied when other methods cannot are of considerable value and interest.

Multiprocessor scheduling. Two approaches traditionally considered for scheduling on multiprocessors are *partitioning* and *global scheduling*. Under partitioning, tasks are statically partitioned among the available processors. Each processor schedules the tasks assigned to it using a separate instance of a uniprocessor scheduling algorithm. Thus, each task is bound to exactly one processor, upon which all its jobs execute. In contrast, under global scheduling, a task may execute on any processor and may migrate across processors. A single ready queue stores all ready jobs, and the job with the highest priority is chosen for execution from this queue at each scheduling decision. The two approaches can be differentiated further based on the scheduling algorithm that is used. For instance, the *earliest-deadline-first* (EDF) or the *rate-monotonic* (RM) scheduling algorithm [Liu and Layland, 1973] could be used as the per-processor scheduler under partitioning or as the system-wide scheduler under global scheduling. Global Pfair scheduling algorithms [Baruah et al., 1996] are the only known means for optimally* scheduling recurrent real-time task systems, including sporadic task systems [Srinivasan and Anderson, 2002], on multiprocessors. However, in practice, Pfair algorithms can incur significant preemption, migration, and scheduling overheads. Hence, other scheduling algorithms that are not optimal have continued to receive considerable attention. Such attention, however, has primarily been confined to hard real-time systems and, in particular, to ensuring that each job of a sporadic task completes execution before its next job is released. We review results of prior work in this direction after comparing the two multiprocessor scheduling approaches described above for non-Pfair algorithms.

Partitioning vs. global scheduling. It is well known that preemptive EDF is optimal for scheduling periodic or sporadic tasks on uniprocessors and that its schedulable utilization is 100% when relative deadlines are *implicit*, *i.e.*, equal to periods [Liu and Layland, 1973]. (If $\mathcal{U}_{\mathcal{A}}(M)$ is a schedulable utilization for scheduling algorithm $\mathcal{A}$ on M processors, then $\mathcal{U}_{\mathcal{A}}(M)$ is the highest known or possible value for which $\mathcal{A}$ can correctly schedule every recurrent task system with total utilization not exceeding $\mathcal{U}_{\mathcal{A}}(M)$ on M processors.) However, EDF is not optimal on multiprocessors either under partitioning or global scheduling. Yet, traditionally, partitioning has been preferred to global scheduling perhaps for the following reasons: **(i)** partitioning eliminates overheads due to task migrations; **(ii)** partitioning is conceptually simpler and analytically more tractable; **(iii)** schedulable utilization achieved in practice is higher under partitioning.

Despite the above advantages of partitioning, some recent developments in computer architecture and scheduling research provide reasons to believe that global scheduling may be viable in many settings. First, task migrations may be less of an issue in modern systems with high-speed processor-to-memory interconnects, in architectures with shared caches, such as some multicore designs, or in embedded systems with no cache. Second, as discussed in detail later in this section, significant progress has been made in the development of techniques for reasoning about global scheduling algorithms and in understanding their behavior. Third, partitioning may be a cumbersome strategy for dynamic task systems, in which tasks may leave and join, due to the need to repartition tasks when task-set changes occur. Finally, global scheduling may be superior to partitioning for soft real-time systems in that partitioning schemes offer no scope for improving system utilization even if bounded tardiness can be tolerated. This is because if a task set cannot be partitioned without over-utilizing some processor, then deadline misses and tardiness for tasks on that processor will increase with time. Such an example under partitioned EDF is provided in Figure 1(a). (Refer to Section 2 for a description of the task model and notation used.) In this example, τ_1 is assigned to processor 1 and τ_2 and τ_3 are assigned to processor 2. Assume that each job of every task is released as early as permissible and that deadline ties between τ_2 and τ_3 are resolved in favor of τ_3. Here, the second job of τ_3 does not complete executing until time 14, for a tardiness of 2 time units, and the third job incurs a tardiness of 4 time units. It is easy to see that the $(i+1)^{st}$ job of τ_3 does not complete until time $8i+6$, for all $i \geq 1$,

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

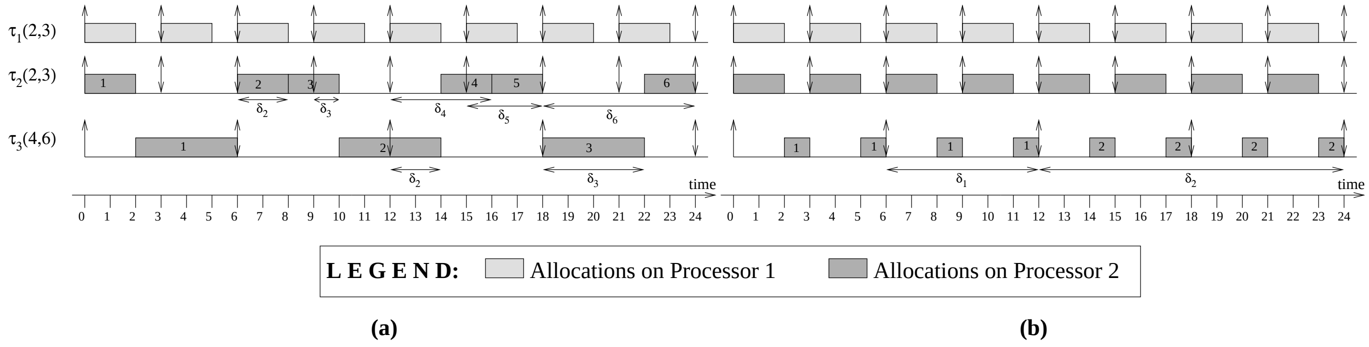


Figure 1: A schedule for the task system indicated under **(a)** partitioned EDF (with task τ_1 assigned to processor 1 and tasks τ_2 and τ_3 assigned to processor 2), and **(b)** global RM. Numbers within shaded rectangles indicate job numbers. δ_i indicates the tardiness of the i^{th} job of the corresponding task.

for a tardiness of $2i$ time units. This tardiness increases with time and thus is unbounded. Similarly, tardiness is unbounded for τ_2 as well. On the other hand, as we show in this paper, the outlook is more promising if tasks are not pinned to processors.

Prior work on global scheduling. One factor that has led many researchers to view global scheduling negatively is the so-called "Dhall effect," which was reported by Dhall and Liu [Dhall and Liu, 1978] as early as 1978. They showed that for every $M \geq 2$, there exist task systems with total utilization arbitrarily close to 1.0 that cannot be correctly scheduled on M processors under global EDF or RM scheduling [Dhall and Liu, 1978]. However, recently, several researchers have noted that the Dhall effect is due to the presence of tasks with high and low utilizations and have shown that it can be overcome by restricting per-task utilizations or according higher priorities to high-utilization tasks [Srinivasan and Baruah, 2002, Goossens et al., 2003, Baruah, 2004, Baker, 2003]. In [Goossens et al., 2003], Goossens, Funk, and Baruah showed that on M processors, EDF can correctly schedule any independent periodic/sporadic task system (with implicit deadlines) if the total utilization of all tasks, U_{sum}, is at most $M - (M-1)u_{\max}$, where $u_{\max}$ is the maximum utilization of any task. Later, Baruah proposed a variant of EDF that prioritizes tasks with utilization exceeding $1/2$ over the rest, and showed that the modified algorithm can correctly schedule any task system with total utilization not exceeding $(M+1)/2$. Schedulability tests that can be applied to task systems with relative deadlines at most periods have been developed by Baker [Baker, 2003] and by Bertogna *et al.* [Bertogna et al., 2005a].

Schedulability tests that have been proposed for implicit-deadline systems depend on $u_{\max}$ and have the property that the total utilization of a task system that is schedulable increases as $u_{\max}$ is decreased. Nevertheless, even with $u_{\max} = 0.5$, half the total available processing capacity will have to be wasted, if every deadline must be met. This may be overkill for soft real-time systems that can tolerate bounded deadline misses.

The research discussed above is for preemptive global EDF (henceforth referred to as g-EDF), under which a job may be preempted by another arriving higher-priority job. To our knowledge, non-preemptive global EDF (referred to as g-NP-EDF) has been considered only in [Baruah, 2006], where a sufficient schedulability condition is derived for task systems in which the maximum execution cost of any task is less than the minimum period of any task.

Non-trivial schedulability tests have been developed for global RM scheduling also [Andersson et al., 2001, Baker, 2003, Bertogna et al., 2005b]. However, the schedulable utilizations that these tests allow are less than that allowed by g-EDF. Furthermore, like partitioning algorithms, global RM (or any global static-priority algorithm) may not be suitable for soft real-time systems. This is because, task systems exist in which tardiness for low-priority tasks increases with time when scheduled under RM. Refer to Figure 1(b) for an example. In this example, the i^{th} job of τ_3 does not complete until time $12i$, for all i, for a tardiness of $6i$ time units.

Contributions of this paper. In this paper, we determine the amount by which any job of a sporadic task may miss its deadline under g-EDF or g-NP-EDF on a multiprocessor, if the total system utilization is not restricted and may be any value up to M, the total number of processors. The tardiness bounds derived apply for sporadic tasks even if jobs may be "early released," *i.e.*, become eligible for execution before their designated release times. Early releasing in multiprocessor systems was first considered by Anderson and Srinivasan [Anderson and Srinivasan, 2004], but similar functionality was considered earlier on uniprocessors by Jeffay and Goddard in their work on rate-based execution [Jeffay and Goddard, 1999]. The tardiness bounds derived depend on the total system utilization, U_{sum}, and on per-task utilizations and execution costs. When $U_{sum} = M$, the tardiness bound derived for g-EDF is $((M-1)\cdot e_{\max} - e_{\min})/(M-(M-2)\cdot u_{\max}) + e_{\max}$, where $e_{\max}$ and $e_{\min}$ are the maximum and minimum execution costs of any task. For g-NP-EDF, we derive a bound of $(M \cdot e_{\max} - e_{\min})/(M - (M-1)\cdot u_{\max}) + e_{\max}$, which is worse than that of g-EDF by over $e_{\max}/(M-(M-1)u_{\max})$. Better bounds, which may be possible by considering the $M-1$ tasks with the highest utilizations and execution costs (as opposed to considering the maximum values only) or by slightly lowering U_{sum}, are also provided in the paper. The difference between the g-EDF and g-NP-EDF bounds is approximately $e_{\max}/M$ when $u_{\max} \to 0.0$, and increases to around $((M+1)\cdot e_{\max})/2$ as $u_{\max} \to 1.0$. However, simulation experiments show that for task systems in which much fewer than M tasks have an execution cost of $e_{\max}$ or a utilization of $u_{\max}$, the difference between the two bounds narrows considerably if computed using the more accurate methods described in the paper. It is also interesting to note that as $M \to \infty$, both bounds converge to $e_{\max}/(1-u_{\max}) + e_{\max}$. Because a job may be blocked for up to $e_{\max}$ time units by a lower-priority job under g-NP-EDF, the small difference between the two bounds at low utilizations suggests that the analysis of g-EDF may not be tight. In a similar vein, the significantly higher difference at high utilizations, when close to M tasks have a utilization of $u_{\max}$ and an execution cost of $e_{\max}$, suggests that a better bound may be possible for g-NP-EDF. In fact, our simulation experiments do confirm that the bounds are not tight. Nevertheless, they seem to be reasonable for practical purposes and these results should enable the class of soft real-time applications described at the beginning of this section to be EDF-scheduled on multiprocessors. We consider the g-NP-EDF result to be particularly significant. Apart from being considerably simpler to implement than g-EDF, g-NP-EDF has the potential to lower worst-case execution-cost estimates by eliminating preemptions and, hence, intra-job migrations. Therefore, a large improvement in system utilization may be possible under g-NP-EDF, if bounded deadline misses are acceptable.

Instead of deriving tardiness bounds independently for g-EDF and g-NP-EDF, we consider a more general model in which tasks may consist of both preemptive and non-preemptive segments. Such a scheme may be useful for modeling and implementing some non-independent tasks, such as tasks that share short critical sections [Devi et al., 2006]. Bounds for g-EDF and g-NP-EDF are deduced as special cases of that derived for the general case.

Organization. The rest of this paper is organized as follows. The task model that we consider is formally presented in Section 2. This is followed by a derivation of tardiness bounds for g-EDF and g-NP-EDF in Section 3. Then, in Section 4, an extension is proposed to the basic sporadic task model, and a derivation of tardiness bounds for the extended model is discussed. Section 5 presents a simulation-based evaluation of the tightness of the bounds (for the basic model). Finally, after related work is reviewed in Section 6, Section 7 concludes.

2 System Model

Sporadic task model. A *sporadic task system* [Mok, 1983] τ composed of $N > M$ independent, sporadic tasks is to be scheduled upon a multiprocessor platform with $M \geq 2$ identical processors. Each task $\tau_i(e_i, p_i)$, where $1 \leq i \leq N$, is characterized by a minimum inter-arrival time, also referred to as its *period* $p_i > 0$, a *worst-case execution cost* $e_i \leq p_i$, and a *relative deadline* $D_i = p_i$. Every task τ_i may be invoked zero or more times with two consecutive invocations separated by at least p_i time units. Each invocation of τ_i is referred to as a *job* of τ_i and the k^{th} job of τ_i, where $k \geq 1$, is denoted $\tau_{i,k}$. The first job may be invoked or *released* at any time at or after time zero. The release time of job $\tau_{i,k}$ is denoted $r_{i,k}$. A *periodic task system*, in which every two consecutive jobs of every task τ_i are separated by exactly p_i time units, is a special case of a sporadic task system. Every job of τ_i executes for at most e_i time units. The *absolute deadline* (or simply, *deadline*) of $\tau_{i,k}$, denoted $d_{i,k}$ and given by $r_{i,k} + D_i$, is the time at or before which $\tau_{i,k}$ should complete execution in a hard real-time system. Each task is sequential, and at any time may execute on at most one processor. The *utilization* of τ_i is given by $u_i = e_i/p_i$. The *total utilization* of τ is defined as $U_{sum}(\tau) = \sum_{i=1}^{n} u_i$. On M processors, it is required that $U_{sum}(\tau) \leq M$ hold. The maximum utilization and the maximum and minimum execution cost of any task are denoted $u_{\max}(\tau)$, $e_{\max}(\tau)$, and $e_{\min}(\tau)$, respectively. The task system τ may be omitted from this notation when unambiguous.

Non-preemptive sections. To provide a common analytic framework for g-EDF and g-NP-EDF, we consider a more general model than the sporadic task model described above, wherein each job of each task τ_i may consist of one or more preemptive and non-preemptive code segments. The maximum execution cost of any non-preemptive segment of τ_i is denoted b_i and the maximum value taken over all tasks in τ is denoted $b_{\max}$. In this extended model, task τ_i is denoted using the triple (e_i, p_i, b_i).

Mixed preemptive/non-preemptive scheduling. We will refer to the EDF algorithm that is cognizant of the non-preemptive segments of a task and executes them non-preemptively as EDF-P-NP. (In a real implementation, special system calls would be used to inform the scheduler when a non-preemptive segment is entered and exited.) At any time, higher priority is accorded to jobs with earlier deadlines, subject to not preempting a job that is executing in a non-preemptive segment. Ties are resolved arbitrarily but consistently in that ties between two jobs are resolved identically at all times. A job executing in a preemptive segment may be preempted by an arriving higher-priority job and may later resume execution on the same or a different processor. EDF-P-NP reduces to g-EDF when $b_{\max} = 0$ and to g-NP-EDF when $b_i = e_i$, for all i. Overheads due to job preemptions and migrations are assumed to be accounted for by inflating worst-case execution costs using standard techniques.

Early releasing. We also consider executing jobs of sporadic tasks "early," *i.e.*, before their release times (as stipulated by the sporadic task model), provided that prior jobs of the same task have completed execution. A job that may be executed before its release time is said to be *early released*. Each job $\tau_{i,j}$ is associated with an *eligibility time*, denoted $\ell_{i,j}$, where $\ell_{i,j} \leq r_{i,j}$ and $\ell_{i,j+1} \geq \ell_{i,j}$ for all i and j. $\ell_{i,j}$ is the earliest time that $\tau_{i,j}$ can commence execution. Early releasing does not alter job deadlines, and hence, how job priorities are determined, but only the set of jobs that can contend at a given time.

Early releasing has been considered by other researchers in prior work and has proved to be useful in scheduling rate-based tasks [Srinivasan and Anderson, 2003, Jeffay and Goddard, 1999, Anderson and Srinivasan, 2004]. On uniprocessors, a swapping argument (that can be used to establish the optimality of EDF) can be used to show that allowing early releases cannot lead to deadline misses for a task system (or a set of jobs) that is otherwise schedulable by EDF. However, the same does not hold for EDF on multiprocessors. An example of a task system in which deadlines are missed due to early releasing is shown in Figure 2.[†] Nevertheless, we show that the tardiness bounds that we derive in this paper hold even if jobs are early released.

Concrete and non-concrete task systems. A task system is said to be *concrete* if the release time, eligibility time, and actual execution time (which is at most the worst-case execution cost) of every job of each of its tasks is specified, and *non-concrete*, otherwise. Note that infinite concrete task systems can be specified for every non-concrete task system. The type of the task system is indicated only when necessary. Unless specified, actual job execution times are to be taken to be equal to worst-case costs. The results in this paper are for non-concrete task systems, and hence hold for every concrete task system.

[†]The task system in this figure is not guaranteed to be schedulable under EDF by any of the algorithm's known sufficient schedulability tests [Goossens et al., 2003, Baker, 2003, Bertogna et al., 2005a]. We believe that deadlines will not be missed even with early releasing if a task system satisfies at least one of these tests.

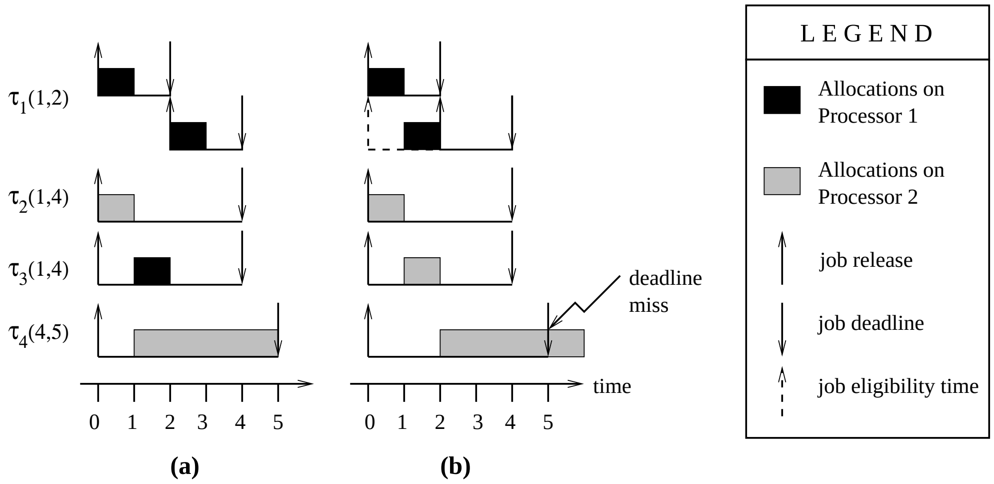


Figure 2: Illustration of deadline misses due to early releasing. EDF schedules are shown for some initial jobs of a task system with four tasks (as indicated) and $U_{sum} = 1.8$ on two processors. **(a)** No job is released early. **(b)** Job $\tau_{1,2}$ is early released at time 0. Scheduling $\tau_{1,2}$ at time 1 before its release time leads to a deadline miss.

Soft real-time model. In *soft* real-time systems, tasks may miss their deadlines. As discussed in the introduction, this paper is concerned with deriving a lateness or *tardiness* [Sha et al., 2004] bound for a sporadic task system scheduled under g-EDF or g-NP-EDF. Towards this end, we derive a tardiness bound under EDF-P-NP. Formally, the tardiness of a job $\tau_{i,j}$ in schedule $\mathcal{S}$ is defined as *tardiness*$(\tau_{i,j}, \mathcal{S}) = \max(0, t - d_{i,j})$, where t is the time at which $\tau_{i,j}$ completes executing in $\mathcal{S}$. The tardiness of a task system τ under scheduling algorithm $\mathcal{A}$ is defined as the maximum tardiness of any job of any task in τ in any schedule for any concrete instantiation of τ under $\mathcal{A}$. If $\kappa(M)$ is the maximum tardiness of any task system with $U_{sum} \leq M$ under $\mathcal{A}$ on M processors, then $\mathcal{A}$ is said to *ensure a tardiness bound of* $\kappa(M)$ on M processors. Though tasks in a soft real-time system are allowed to have nonzero tardiness, we assume that *missed deadlines do not delay future job releases*. For example, in the schedule in Figure 1(a), though jobs of τ_2 and τ_3 miss their deadlines, releases of future jobs are not impacted. Hence, guaranteeing a reasonable bound on tardiness that is independent of time is sufficient to ensure that in the long run each task is allocated a processor share that is in accordance with its utilization. Because each task is sequential and jobs have an implicit precedence relationship, a later job cannot commence execution until all prior jobs of the same task have completed execution. Thus, a missed deadline effectively reduces the interval over which the next job should be scheduled in order to meet its deadline.

Discussion. It may be argued that a real-time task model in which no job misses its deadline but in which jobs can occasionally be skipped may be better suited for some soft real-time systems, such as multimedia systems. Taking a video decoding task of a multimedia system (considered in Section 1) as an example, such a model may have the disadvantage that optimally determining whether to skip a frame can incur additional scheduling overhead at run-time. This is because not too many frames can be skipped consecutively from the same stream and every frame that is not skipped should meet its deadline. Further, to our knowledge, no experimental evidence is available as to which is a better approach. Also, if the video stream is MPEG-coded, it may not be possible to skip frames arbitrarily. For instance, since the I and P frames are needed in decoding B frames and other P frames, there is less freedom in choosing frames to skip.

Additional notation. The tardiness bound we derive is expressed in terms of the highest task execution costs, utilizations, and non-preemptive segment costs, and the total system utilization. To facilitate expressing the bound, we define the following: ϵ_i (resp., μ_i) denotes the i^{th} execution cost (resp., task utilization) in a non-increasing order of the execution costs (resp., utilizations) of all the tasks. (Note that ϵ_i and μ_i need not be parameters of the same task for any i.) Similarly, β_i denotes the i^{th} largest non-preemptive segment cost. For simplicity, we assume the following.

$$(\forall i, j \leq N :: e_i \leq e_j \Leftrightarrow b_i \leq b_j) \tag{1}$$

Note that the above assumption holds for both g-EDF and g-NP-EDF. This assumption can be eliminated for mixed-mode tasks at the expense of either a slightly higher tardiness bound or a more complicated algorithm (as opposed to a closed-form expression) for choosing tasks whose execution costs are to be used in computing a less-pessimistic bound. One such algorithm is described in [Devi, 2006].

We also define the following subsets of tasks in order to lower the pessimism in the bounds specified. In these definitions, $k \leq N$ and $k_2 \leq k_1 \leq N$ hold. (These definitions may be skipped on first reading without loss of continuity and referred back to when needed.) The purpose of defining these sets is as follows. The tardiness bounds derived are dependent upon the sum of the execution costs of tasks in some subset of all the tasks and that of the non-preemptive segment costs of tasks in another disjoint subset. While it is simpler to use the maximum execution costs and maximum non-preemptive segment costs as upper bounds, that approach overlooks the fact that the subsets are disjoint and, hence, that at most one of the execution cost and non-preemptive segment cost of a task may be used. The challenge is to identify tasks for the two subsets that can be used to compute an upper bound for the needed quantity. If (1) holds, then $\Gamma^{(k_1,k_2)}$ and $\Pi^{(k_1,k_2)}$, serve the purpose.[‡]

$$\Gamma^{(k)} \stackrel{\text{def}}{=} \text{Subset of } k \text{ tasks of } \tau \text{ with the highest execution costs} \tag{2}$$

$$\Gamma^{(k_1,k_2)} \stackrel{\text{def}}{=} \text{Subset of } k_2 \text{ tasks } \tau_i \text{ of } \Gamma^{(k_1)} \text{ with the highest values for } e_i - b_i \tag{3}$$

$$\Pi^{(k_1,k_2)} \stackrel{\text{def}}{=} \Gamma^{(k_1)} \setminus \Gamma^{(k_1,k_2)} \tag{4}$$

Ties in selecting tasks for $\Gamma^{(k)}$ are resolved by task indices. When choosing tasks for $\Gamma^{(k_1,k_2)}$ ties are first resolved in favor of tasks with higher execution costs; any remaining ties are resolved by task indices. The following are

[‡]If (1) does not hold, then these definitions are not applicable. In such a case, $\Gamma^{(k_1,k_2)}$ and $\Pi^{(k_1,k_2)}$ will have to be determined iteratively. In either case, defining $\Gamma^{(k_1,k_2)}$ (resp., $\Pi^{(k_1,k_2)}$) as a subset of k_1 (resp., $k_2 - k_1$) tasks of τ with the highest execution costs (resp., non-preemptive segment costs) will serve as a looser upper bound.

i	1	2	3	4	5	6	7	8	9
e_i	20	10	16	2	15	4	12	4	20
p_i	60	15	20	10	20	16	20	10	40
u_i	0.33	0.67	0.8	0.2	0.75	0.25	0.6	0.4	0.5
b_i	7	2	6	0	3	0.5	1	0	7
ϵ_i	20	20	16	15	12	10	4	4	2
μ_i	0.8	0.75	0.67	0.6	0.5	0.4	0.33	0.25	0.2
β_i	7	7	6	3	2	1	0.5	0	0

Table 1: Illustration of notation.

immediate from the above definitions.

$$\Gamma^{(k_1,k_2)} \cup \Pi^{(k_1,k_2)} = \Gamma^{(k_1)} \tag{5}$$
$$\Gamma^{(k_1,k_2)} \cap \Pi^{(k_1,k_2)} = \emptyset$$

Finally, Λ is defined as follows.

$$\Lambda \;=\; \begin{cases} U_{sum}(\tau) - 1, & U_{sum}(\tau) \text{ is integral} \\ \lfloor U_{sum}(\tau) \rfloor, & \text{otherwise} \end{cases} \tag{6}$$

(Λ is simply equal to $\lceil U_{sum}(\tau) \rceil - 1$, but the above expression is more explicit.)

Example. Let τ be a task system with nine tasks τ_1 through τ_9 as follows: $\tau = \{\tau_1(20, 60, 7),\ \tau_2(10, 15, 2),\ \tau_3(16, 20, 6),\ \tau_4(2, 10, 0),\ \tau_5(15, 20, 3),\ \tau_6(4, 16, 0.5),\ \tau_7(12, 20, 1),\ \tau_8(4, 10, 0),\ \tau_9(20, 40, 7)\}$. In this example, $U_{sum} = 4.5$, $u_{\max} = u_3 = 0.8$, $e_{\max} = e_1 = e_9 = 20$, $e_{\min} = e_4 = 2.0$, and $b_{\max} = b_1 = b_9 = 7$. ϵ and μ values for this task set are enumerated in Table 1. For this task set, $\Gamma^{(5)} = \{\tau_1, \tau_9, \tau_3, \tau_5, \tau_7\}$. Since $e_1 - b_1 = e_9 - b_9 > e_5 - b_5 > e_7 - b_7 > e_3 - b_3$, we have $\Gamma^{(5,3)} = \{\tau_1, \tau_9, \tau_5\}$ and $\Pi^{(5,3)} = \{\tau_7, \tau_3\}$.

3 A Tardiness Bound under EDF-P-NP

In this section, we derive a tardiness bound under EDF-P-NP. Our approach involves comparing the allocations to a concrete task system τ in a processor sharing (PS) schedule and an actual EDF-P-NP schedule of interest for τ, both on M processors, and quantifying the difference between the two. A PS schedule is an ideal fluid schedule in which each task executes at a precisely uniform rate given by its utilization. Similar proof techniques that use a fluid schedule for reference have previously been considered in research on fair bandwidth allocation in computer networks [Parekh and Gallager, 1993] and fair uniprocessor [Stoica et al., 1996] and multiprocessor [Baruah et al., 1996, Srinivasan and Anderson, 2002] scheduling algorithms. In particular, our derivation borrows techniques developed by Srinivasan and Anderson for reasoning about Pfair algorithms on multiprocessors [Srinivasan and

Anderson, 2002, 2006, 2005, 2003]. The applicability to EDF of techniques used with fair scheduling algorithms is not surprising since EDF also approximates an ideal fluid scheduler, although more coarsely than fair schedulers. Valente and Lipari also use a similar approach in deriving a tardiness bound under g-EDF [Valente and Lipari, 2005].[§]

In a PS schedule, each job of τ_i can be thought of as being allocated a fraction u_i of a processor at each instant in the interval between its release time and its deadline. (Non-preemptivity constraints are ignored.) In other words, a processor of capacity u_i is made available for the exclusive use of task τ_i whenever it is active (defined formally later). Equivalently, a job can be thought of as being allocated a fraction u_i of each instant between its release and deadline. Such a schedule can be realized if task contexts can be switched at arbitrarily fine time scales. Because $D_i = p_i$, for all i, and $U_{sum} \leq M$ holds, the total demand at any instant will not exceed M in a PS schedule, and hence no deadlines will be missed; in fact, every job will complete executing exactly at its deadline. We begin by setting the required machinery in place.

3.1 Definitions and Notation

All times referred to in this paper are real. The system start time is assumed to be zero. The time interval $[t_1, t_2)$, where $t_2 \geq t_1$, consists of all times t, where $t_1 \leq t < t_2$, and is of length $t_2 - t_1$. The interval (t_1, t_2) excludes t_1. For any time $t > 0$, the notation t^- is used to denote the time $t - \epsilon$ in the limit $\epsilon \to 0+$.

Definition 1 (active tasks, active jobs, and windows): A task τ_i is said to be *active* at time t if there exists a job $\tau_{i,j}$ (called τ_i's *active job* at t) such that $r_{i,j} \leq t < d_{i,j}$ (*i.e.*, $\tau_{i,j}$'s release time has elapsed, but its deadline has not). The interval $[r_{i,j}, d_{i,j})$ is referred to as the *window* of $\tau_{i,j}$. By our task model, $D_i = p_i$ holds, hence every task can have at most one active job at any time. $\tau_{i,j}$ is not considered to be active before its release time even if its eligibility time, $\ell_{i,j}$, is earlier.

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

Definition 3 (ready jobs): $\tau_{i,j}$ is said to be *ready* at time t in a schedule $\mathcal{S}$ if $t \geq \ell_{i,j}$, and all prior jobs of τ_i have completed execution by t in $\mathcal{S}$, but $\tau_{i,j}$ has not.

We next quantify the total allocation to τ in an interval $[t_1, t_2)$ in a PS schedule for τ, denoted PS_τ. For this, let $\mathsf{A}(\mathcal{S}, \tau_i, t_1, t_2)$ denote the total time allocated to τ_i in an arbitrary schedule $\mathcal{S}$ for τ in $[t_1, t_2)$. In PS_τ, τ_i is

[§] As publicly reported by the first author of [Valente and Lipari, 2005] at RTSS '05, the proof published in [Valente and Lipari, 2005] is in error. To our knowledge, an updated document has not yet been published at a refereed venue.

allocated a fraction u_i of each instant at which it is active in $[t_1, t_2)$, regardless of whether its active job is executing in a preemptive or a non-preemptive section. (Recall that non-preemptivity constraints are ignored within PS schedules.)

$$\mathsf{A}(\mathsf{PS}_\tau, \tau_i, t_1, t_2) \leq (t_2 - t_1) \cdot u_i.$$

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

$$\mathsf{A}(\mathsf{PS}_\tau, \tau, t_1, t_2) \leq \sum_{\tau_i \in \tau} (t_2 - t_1) u_i = U_{sum} \cdot (t_2 - t_1) \leq M \cdot (t_2 - t_1). \tag{7}$$

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

$$\mathsf{lag}(\tau_i, t, \mathcal{S}) = \mathsf{A}(\mathsf{PS}_\tau, \tau_i, 0, t) - \mathsf{A}(\mathcal{S}, \tau_i, 0, t). \tag{8}$$

Schedule $\mathcal{S}$ has performed less work on the jobs of τ_i until t than PS_τ (or τ_i is under-allocated in $\mathcal{S}$ at t) if $\mathsf{lag}(\tau_i, t, \mathcal{S})$ is positive and more work (or τ_i is over-allocated in $\mathcal{S}$ at t) if $\mathsf{lag}(\tau_i, t, \mathcal{S})$ is negative. The total lag of a task system τ at t, denoted $\mathsf{LAG}(\tau, t, \mathcal{S})$, is given by the following.

$$\begin{aligned}
\mathsf{LAG}(\tau, t, \mathcal{S}) &= \mathsf{A}(\mathsf{PS}_\tau, \tau, 0, t) - \mathsf{A}(\mathcal{S}, \tau, 0, t) && (9)\\
&= \sum_{\tau_i \in \tau} \mathsf{A}(\mathsf{PS}_\tau, \tau_i, 0, t) - \sum_{\tau_i \in \tau} \mathsf{A}(\mathcal{S}, \tau_i, 0, t) \\
&= \sum_{\tau_i \in \tau} \mathsf{lag}(\tau_i, t, \mathcal{S}) && (10)
\end{aligned}$$

Note that $\mathsf{LAG}(\tau, 0, \mathcal{S})$ and $\mathsf{lag}(\tau_i, 0, \mathcal{S})$ are both zero, and that by (8) and (9), we have the following for $t_2 > t_1$.

$$\begin{aligned}
\mathsf{lag}(\tau_i, t_2, \mathcal{S}) &= \mathsf{lag}(\tau_i, t_1, \mathcal{S}) + \mathsf{A}(\mathsf{PS}_\tau, \tau_i, t_1, t_2) - \mathsf{A}(\mathcal{S}, \tau_i, t_1, t_2) \\
\mathsf{LAG}(\tau, t_2, \mathcal{S}) &= \mathsf{LAG}(\tau, t_1, \mathcal{S}) + \mathsf{A}(\mathsf{PS}_\tau, \tau, t_1, t_2) - \mathsf{A}(\mathcal{S}, \tau, t_1, t_2) && (11)
\end{aligned}$$

Lag for jobs. The notion of lag defined above for tasks and task sets can be applied to jobs and job sets in an obvious manner. Let τ denote a concrete task system and Ψ a subset of all jobs in τ with deadlines not later than a specified time. Let $\mathsf{A}(\mathsf{PS}_\tau, \tau_{i,j}, t_1, t_2)$ and $\mathsf{A}(\mathcal{S}, \tau_{i,j}, t_1, t_2)$ denote the allocations to $\tau_{i,j}$ in $[t_1, t_2)$ in PS_τ and $\mathcal{S}$, respectively. Then, $\mathsf{lag}(\tau_{i,j}, t, \mathcal{S}) = \mathsf{A}(\mathsf{PS}_\tau, \tau_{i,j}, r_{i,j}, t) - \mathsf{A}(\mathcal{S}, \tau_{i,j}, \ell_{i,j}, t)$. (This is because a job cannot receive any allocation outside its window in a PS schedule and before its eligibility time in an actual schedule.) The total allocation in $\mathcal{S}$ in $[0, t)$, where $t > 0$, to a job that is not pending at t^- in $\mathcal{S}$ is at least its allocation in the same interval in PS_τ. Hence, its lag at t is at most zero. (Note that if a task is not pending at t^- but at t, it implies that the task has a new job released at t but none of its prior jobs is pending, and, hence, its lag at t is zero.) Also, the total lag of Ψ is given by the sum of the lags of its jobs. Therefore, for $t > 0$, we have

$$\begin{aligned}
\mathsf{LAG}(\Psi, t, \mathcal{S}) &= \sum_{\{\tau_{i,j} \text{ is in } \Psi\}} \mathsf{lag}(\tau_{i,j}, t, \mathcal{S}) \\
&= \sum_{\substack{\{\tau_{i,j} \text{ is in } \Psi \text{ and is} \\ \text{pending at } t^-\}}} \mathsf{lag}(\tau_{i,j}, t, \mathcal{S}) + \sum_{\substack{\{\tau_{i,j} \text{ is in } \Psi \text{ and is} \\ \text{not pending at } t^-\}}} \mathsf{lag}(\tau_{i,j}, t, \mathcal{S}) \\
&\leq \sum_{\substack{\{\tau_i \in \tau \mid \text{some} \\ \tau_{i,j} \text{ is in } \Psi \text{ and is} \\ \text{pending at } t^-\}}} \Bigg(\sum_{\substack{\{\tau_{i,j} \text{ is in } \Psi \\ \text{and is pending} \\ \text{at } t^-\}}} \mathsf{lag}(\tau_{i,j}, t, \mathcal{S}) + \sum_{\substack{\{\tau_{i,j} \text{ is in } \Psi \text{ and} \\ \text{is not pending} \\ \text{at } t^-\}}} \mathsf{lag}(\tau_{i,j}, t, \mathcal{S}) \Bigg) \\
&\leq \sum_{\substack{\{\tau_i \in \tau \mid \text{some} \\ \tau_{i,j} \text{ is in } \Psi \text{ and is} \\ \text{pending at } t^-\}}} \Bigg(\sum_{\substack{\{\tau_{i,j} \text{ is in } \Psi \\ \text{and is pending} \\ \text{at } t^-\}}} \mathsf{lag}(\tau_{i,j}, t, \mathcal{S}) + \sum_{\substack{\{\tau_{i,j} \text{ is in } \Psi \text{ and} \\ \text{is not pending} \\ \text{at } t^-\}}} \mathsf{lag}(\tau_{i,j}, t, \mathcal{S}) + \sum_{\{\tau_{i,j} \text{ is not in } \Psi\}} \mathsf{lag}(\tau_{i,j}, t, \mathcal{S}) \Bigg) \qquad (12)
\end{aligned}$$

The last inequality holds because by the definition of Ψ, every job of τ_i that is not in Ψ is released after every job of the same task that is in Ψ. Hence, if some job of τ_i that is in Ψ is pending at t^-, then no later job can receive any allocation by t, and hence, the lag for every such job is at least zero. Because the lag of every task τ_i is given by the sum of the lags of its jobs, (12) implies the following.

$$\mathsf{LAG}(\Psi, t, \mathcal{S}) \leq \sum_{\substack{\{\tau_i \in \tau \mid \text{ some } \tau_{i,j} \text{ is in} \\ \Psi \text{ and is pending at } t^-\}}} \mathsf{lag}(\tau_i, t, \mathcal{S}) \qquad (13)$$

Let the total *instantaneous utilization* of Ψ at time t be defined as the sum of the utilizations of all the tasks with an active job at t in Ψ:

$$\begin{aligned}
U_{sum}(\Psi, t) &= \sum_{\substack{\{\tau_i \in \tau \mid \tau_{i,j} \text{ is in } \Psi \\ \text{and is active at } t\}}} u_i \qquad (14) \\
\Rightarrow \quad U_{sum}(\Psi, t) &\leq U_{sum}(\tau). \qquad (15)
\end{aligned}$$

The counterparts of (7) and (11) for job sets can now be expressed as follows. (As with (7) and (11), $t_1 < t_2$.)

$$\begin{aligned}
\mathsf{A}(\mathsf{PS}_\tau, \Psi, t_1, t_2) &= \int_{t_1}^{t_2} U_{sum}(\Psi, t)\mathrm{d}t \qquad (16) \\
&\leq (t_2 - t_1) \cdot U_{sum}(\tau) \\
\mathsf{LAG}(\Psi, t_2, \mathcal{S}) &= \mathsf{LAG}(\Psi, t_1, \mathcal{S}) + \mathsf{A}(\mathsf{PS}_\tau, \Psi, t_1, t_2) - \mathsf{A}(\mathcal{S}, \Psi, t_1, t_2) \qquad (17)
\end{aligned}$$

Definition 4 (busy and non-busy intervals): A time interval $[t, t+\delta)$, where $\delta > 0$, is said to be *busy* for Ψ if all M processors are executing some job in Ψ at each instant in the interval, *i.e.*, no processor is ever idle or executes a job not in Ψ in the interval. An interval $[t, t+\delta)$ that is not busy for Ψ is said to be *non-busy* for Ψ. $[t, t+\delta)$ is *continually non-busy* if every time instant in $[t, t+\delta)$ is non-busy, and is *maximally non-busy* if $[t, t+\delta)$ is continually non-busy and either $t = 0$ or t^- is busy.

If at least $U_{sum}(\Psi, t)$ jobs from Ψ are executing at every instant t in $[t_1, t_2)$ in a schedule $\mathcal{S}$, then, by (16), the total allocation in $\mathcal{S}$ to jobs in Ψ is at least the allocation that Ψ receives in a PS schedule. Therefore, by (17), the LAG of Ψ at t_2 cannot exceed that at t_1, and we have the following lemma.

Lemma 1 *If* $\mathsf{LAG}(\Psi, t+\delta, \mathcal{S}) > \mathsf{LAG}(\Psi, t, \mathcal{S})$*, where* $\delta > 0$ *and* $\mathcal{S}$ *is a schedule for* τ*, then* $[t, t+\delta)$ *is a non-busy interval for* Ψ *in* $\mathcal{S}$*. Furthermore, there exists at least one instant* t' *in* $[t, t+\delta)$ *at which fewer than* $U_{sum}(\Psi, t')$ *processors are executing jobs from* Ψ*.*

Definition 5 (continually-increasing LAG): If $\mathsf{LAG}(\Psi, t', \mathcal{S}) > \mathsf{LAG}(\Psi, t'^-, \mathcal{S})$ for all t' in $(t, t+\delta]$, then $[t, t+\delta)$ is said to be an interval with *continually-increasing* LAG for Ψ in $\mathcal{S}$.

If at least $U_{sum}(\Psi, t^-)$ jobs from Ψ are executing at t^- in $\mathcal{S}$, then by (16) and (17) again, $\mathsf{LAG}(\Psi, t, \mathcal{S}) \leq \mathsf{LAG}(\Psi, t^-, \mathcal{S})$, and the lemma below follows.

Lemma 2 *If* $[t, t+\delta)$*, where* $\delta > 0$*, is an interval across which* LAG *for* Ψ *is continually increasing, then fewer than* $U_{sum}(\Psi, t')$ *processors execute jobs from* Ψ *at every instant* t' *in* $[t, t+\delta)$*.*

3.2 Deriving a Tardiness Bound

In this section, we show that on M processors, $\mathsf{EDF\text{-}P\text{-}NP}$ ensures a tardiness not exceeding $x + e_k$ for every task τ_k of every task system with total utilization at most M, where

$$x = \frac{\sum_{\tau_i \in \Gamma^{(\Lambda+1,\Lambda)}} e_i + \sum_{\tau_i \in \Pi^{(\Lambda+1,\Lambda)}} b_i + \sum_{i=1}^{M-\Lambda-1} \beta_i - e_{\min}}{M - \sum_{i=1}^{\Lambda-\rho} \mu_i}, \tag{18}$$

and ρ is as defined in (19) (in page 15). Our proof is by contradiction. So, assume to the contrary that there exists some concrete task system with a job whose tardiness under $\mathsf{EDF\text{-}P\text{-}NP}$ exceeds x plus its worst-case execution cost. Let τ be one such concrete task system defined as follows.

Definition 6 (τ): τ is a concrete instantiation of a non-concrete task system τ^N, $\tau_{\ell,j}$ is a job of a task τ_ℓ in τ, $t_d = d_{\ell,j}$ ($\tau_{\ell,j}$'s deadline), and $\mathcal{S}$ is an $\mathsf{EDF\text{-}P\text{-}NP}$ schedule for τ such that (P1)–(P4) defined below hold.

(P1) The tardiness of $\tau_{\ell,j}$ in $\mathcal{S}$ is greater than $x + e_\ell$.

(P2) No concrete instantiation of τ^N satisfying (P1) releases fewer jobs than τ.

(P3) No concrete instantiation of τ^N satisfying (P1) and (P2) has a smaller cumulative actual execution time for its jobs than τ.

(P4) The tardiness of every job of every task τ_k in τ with deadline less than t_d is at most $x + e_k$ in $\mathcal{S}$.

(P2) and (P3) can be thought of as identifying a minimal task system in the sense of releasing the minimum number of jobs and having the smallest value for the sum of the actual execution times for its jobs for which the claimed tardiness bound does not hold. It is easy to see that if the claimed bound does not hold for all task systems, then some task system satisfying (P2) and (P3) necessarily exists. (P4) identifies a job with the earliest deadline in τ for which the bound does not hold. In what follows, we show that if τ as defined above exists, then (18) is contradicted. In particular, we show that unless x, on which the tardiness of $\tau_{\ell,j}$ depends, is less than the quantity specified on the right-hand side of (18), (P1) cannot hold, thereby establishing the desired tardiness bound.

Proof overview. The completion time of $\tau_{\ell,j}$, and hence its tardiness, depends on the amount of work pending for $\tau_{\ell,j}$ at t_d and the amount of work that can compete with $\tau_{\ell,j}$ after t_d. Hence, to meet our objective, we follow the steps below.

(S1) Determine an upper bound on the sum of the work pending for $\tau_{\ell,j}$ at t_d and the work that can compete with $\tau_{\ell,j}$ after t_d (UB). This is dealt with in Lemmas 9 and 10, with Lemmas 4 and 5 and Claims 2 and 3 providing some needed properties. UB is determined by computing task lags, and work due to non-preemptive sections that have commenced execution and are pending at t_d.

(S2) Determine a lower bound (LB) on the amount of such work (as described in (S1) above) required for the tardiness of $\tau_{\ell,j}$ to exceed $x + e_\ell$. Lemma 3 provides the needed lower bound.

(S3) Show that unless (18) is contradicted, $\mathsf{UB} \leq \mathsf{LB}$, in turn showing that *tardiness*$(\tau_{\ell,j})$ is at most $x + e_\ell$, which is a contradiction to (P1).

To facilitate computing LB and UB, we define Ψ and $\overline{\Psi}$ as follows.

$$\Psi \stackrel{\text{def}}{=} \text{set of all jobs with deadlines at most } t_d \text{ of tasks in } \tau$$

$$\overline{\Psi} \stackrel{\text{def}}{=} \text{set of all jobs of } \tau \text{ that are not in } \Psi \text{ (\textit{i.e.}, jobs with deadlines later than } t_d)$$

Under $\mathsf{EDF\text{-}P\text{-}NP}$, the sum of the work pending for $\tau_{\ell,j}$ and competing work for $\tau_{\ell,j}$ at t_d is given by **(i)** the amount of work pending at t_d for jobs in Ψ plus **(ii)** the amount of work demanded after t_d by the non-preemptive sections of jobs that are not in Ψ but whose execution commenced before t_d. Because the deadline of every job in Ψ is at most t_d, all jobs in Ψ complete execution by t_d in $\mathsf{PS}\tau$. Hence, component (i) is given by $\mathsf{LAG}(\Psi, t_d, \mathcal{S})$. To facilitate computing component (ii), which will be denoted $\mathsf{B}(\tau, \Psi, t_d, \mathcal{S})$, we define the following.

Definition 7 (priority inversions, blocking jobs, and blocked jobs): Under EDF-P-NP, a *priority inversion* occurs when a ready, higher-priority job (*i.e.*, a job with an earlier deadline) waits while one or more lower-priority jobs (*i.e.*, jobs with later deadlines) execute in non-preemptive sections. Under such scenarios, the waiting higher-priority job is said to be a *blocked* job, while the executing lower-priority jobs, *blocking* jobs. Note that a pending, higher-priority job is not considered blocked unless it is ready (*i.e.*, no prior job of the same task is pending).

Definition 8 (blocking and non-blocking, non-busy intervals): Recall that in a non-busy interval for Ψ, there should be at least one time instant in which fewer than M jobs from Ψ execute. In an EDF-P-NP schedule, such a non-busy interval for Ψ can be classified into two types depending on whether a job in $\overline{\Psi}$ is executing while a ready job from Ψ is waiting. We will refer to the two types as *blocking* and *non-blocking* non-busy intervals. A *blocking, non-busy interval* is one in which at one or more instants a job in $\overline{\Psi}$ is executing while a ready job from Ψ is waiting, whereas a *non-blocking, non-busy interval* is one in which at some instants fewer than M jobs from Ψ are executing, but there does not exist a ready job in Ψ that is waiting. Definitions of maximal versions of these intervals are analogous to that of a maximally non-busy interval given in Definition 4.

Definition 9 (pending blocking jobs ($\mathcal{B}$) and work (B)): The set of all jobs in $\overline{\Psi}$ that commence executing a non-preemptive section before t, where t is a blocking, non-busy instant, and may continue to execute the same non-preemptive section at t in $\mathcal{S}$ is referred to as the set of *blocking jobs pending at t* and is denoted $\mathcal{B}(\tau, \Psi, t, \mathcal{S})$; the total amount of work pending at t for such non-preemptive sections is referred to as *pending blocking work at t* and is denoted $\mathsf{B}(\tau, \Psi, t, \mathcal{S})$.

Lastly, we define ρ as follows. Recall that apart from task system parameters, x in (18) also depends on ρ. The higher the value for ρ, the lower the tardiness. ρ is at least zero and its current best-known value is given by Lemma 4.

Definition 10 (ρ):

$$\rho = \begin{cases} \text{Minimum number of jobs with deadlines at or after } t' \text{ guaranteed to execute at } t'^{-}, \text{ where } t < t' \leq t_d, \text{ and } [t, t') \text{ is a non-blocking, non-busy interval with continually-increasing } \mathsf{LAG} & , \text{if } b_{\max} = 0 \\ 0 & , \text{if } b_{\max} > 0 \end{cases} \quad (19)$$

With needed definitions in place, we now turn to deriving a tardiness bound following steps (S1)–(S3). We first consider (S2), that of determining a lower bound on the sum of pending work and competing work required at t_d for tardiness of $\tau_{\ell,j}$ to exceed $x + e_\ell$. (We omit specifying the schedule $\mathcal{S}$ in functions lag, LAG, and B when unambiguous.)

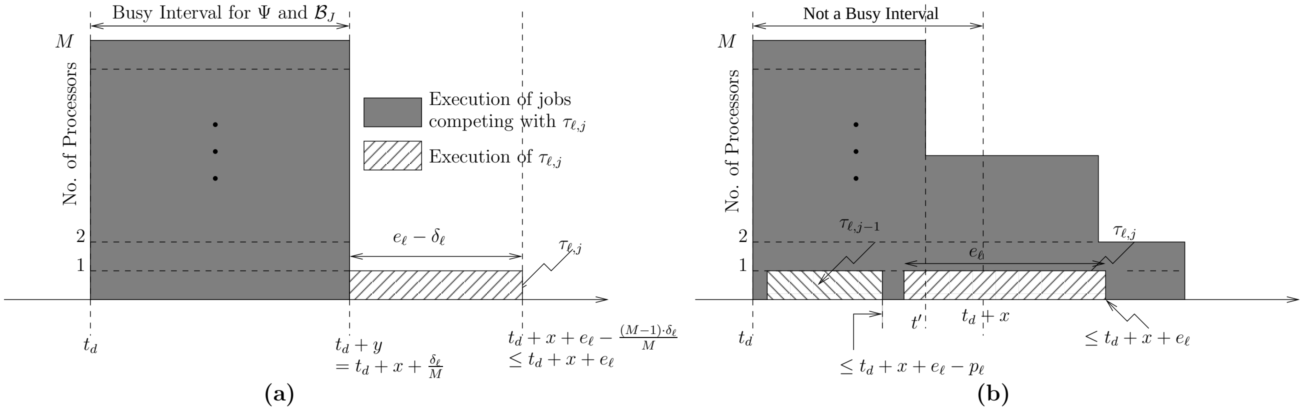


Figure 3: Illustration for some of the cases in the proof of Lemma 3. Shaded blocks denote the execution after t_d of $\tau_{\ell,j}$ and jobs that can compete with $\tau_{\ell,j}$. **(a)** Case 1. Worst-case for $\tau_{\ell,j}$: after t_d, $\tau_{\ell,j}$ executes after all of the competing jobs have completed execution. **(b)** Part of Case 2.

3.2.1 Lower Bound on $\mathsf{LAG}(\Psi, t_d, \mathcal{S}) + \mathsf{B}(\tau, \Psi, t_d, \mathcal{S})$ (Step (S2))

In the trivial case of $\tau_{\ell,j}$ completing execution by t_d, its tardiness is zero. In the other case, since $\tau_{\ell,j}$ cannot be preempted once it begins executing at or after t_d (as shown in Claim 1 below), its tardiness depends on when it begins executing after t_d, and for how much time it executed before t_d, say δ_ℓ (or equivalently, how much of its execution is pending at t_d, which is $e_\ell - \delta_\ell$). The amount of time by which $\tau_{\ell,j}$ can afford to have its execution delayed after t_d for its tardiness to be within a certain amount (which is $x + e_\ell$ in our case) decreases as δ_ℓ decreases. If $\delta_\ell = 0$, then it is easy to see that as long as the competing work due to the other jobs is not greater than $M \cdot x$ (that is, the sum of the pending work and competing work is at most $M \cdot x + e_\ell$) and all processors are busy in $[t_d, t_d + x)$, $\tau_{\ell,j}$ is guaranteed to commence execution by $t_d + x$ and complete by $t_d + x + e_\ell$. Therefore, to establish $M \cdot x + e_\ell$ as the desired lower bound, it suffices to establish that $\tau_{\ell,j}$'s tardiness is at most $x + e_\ell$ in the following two cases also: **(i)** $\delta_\ell < e_\ell$ and competing work due to other jobs exceeds $M \cdot x$ by $e_\ell - \delta_\ell$ (which is depicted in Figure 3(a)); **(ii)** not every processor is busy in $[t_d, t_d + x)$ (part of which is as shown in Figure 3(b)). The lemma below addresses these various cases.

Lemma 3 *If* $\mathsf{LAG}(\Psi, t_d, \mathcal{S}) + \mathsf{B}(\tau, \Psi, t_d, \mathcal{S}) \leq M \cdot x + e_\ell$*, then tardiness*$(\tau_{\ell,j}, \mathcal{S})$ *is at most* $x + e_\ell$.

Proof: Before proving the lemma, we establish the following claim.

> **Claim 1** *If $\tau_{\ell,j}$ executes at or after t_d, then it cannot be subsequently preempted.*
>
> **Proof:** Suppose the claim does not hold and let $t' > t_d$ denote the earliest time at which $\tau_{\ell,j}$ is preempted after executing at or after t_d. Note that only jobs with deadlines at or before t_d can preempt $\tau_{\ell,j}$. Since no job with eligibility time at t' can have its deadline at or before t_d, $\tau_{\ell,j}$ cannot be preempted at t' by a newly-arriving job. Our assumption that ties are resolved consistently implies that if $\tau_{\ell,j}$ is scheduled in preference to another job with an equal priority before t', then that job cannot preempt

$\tau_{\ell,j}$ afterward. Hence, $\tau_{\ell,j}$ is preempted by a job with deadline at or before t_d that is not ready until t' because its predecessor executed until t'. (Otherwise, such a job can preempt $\tau_{\ell,j}$ before t'.) However, for every such job, the processor on which its predecessor executed is available for it at t'. Hence, no such job needs to preempt $\tau_{\ell,j}$. □

With the above claim in place, to prove the lemma, we show that $\tau_{\ell,j}$ completes executing by $t_d + x + e_\ell$. If $j > 1$, then $d_{\ell,j-1} \leq t_d - p_\ell$ holds, and hence by (P4), we have the following.

(R) $\tau_{\ell,j-1}$ completes executing by $t_d - p_\ell + x + e_\ell$, for $j > 1$.

Let $\delta_\ell < e_\ell$ denote the amount of time that $\tau_{\ell,j}$ has executed for before time t_d. Then, the amount of work pending for $\tau_{\ell,j}$ at t_d is $e_\ell - \delta_\ell$. Recall that the total amount of work pending at t_d for jobs in Ψ and the non-preemptive sections of jobs in $\overline{\Psi}$ that commenced execution before t_d (*i.e.*, of jobs in $\mathcal{B}(\tau, \Psi, t_d, \mathcal{S})$) is given by $\mathsf{LAG}(\Psi, t_d, \mathcal{S}) + \mathsf{B}(\tau, \Psi, t_d, \mathcal{S})$, which, by the statement of the lemma, is at most $M \cdot x + e_\ell$. Let $y = x + \delta_\ell/M$. Let $\mathcal{B}_J$ denote the set of all non-preemptive sections of jobs in $\mathcal{B}(\tau, \Psi, t_d, \mathcal{S})$ that commenced execution before t_d in $\mathcal{S}$ and are still pending at t_d. At the risk of abusing terms, let a time interval after t_d in which each processor is busy executing a job of Ψ or a non-preemptive section in $\mathcal{B}_J$ be referred to as a busy interval for Ψ and $\mathcal{B}_J$. We consider the following two cases.

Case 1: $[t_d, t_d + y)$ is a busy interval for Ψ and $\mathcal{B}_J$. An illustration for this case is provided in Figure 3(a). Here, the amount of work completed in $[t_d, t_d + y)$ for jobs in Ψ and $\mathcal{B}(\tau, \Psi, t_d, \mathcal{S})$ is exactly $M \cdot y = M \cdot x + \delta_\ell$, and so, the amount of work pending at $t_d + y$ for jobs in Ψ and $\mathcal{B}(\tau, \Psi, t_d, \mathcal{S})$ is at most $M \cdot x + e_\ell - (M \cdot x + \delta_\ell) = e_\ell - \delta_\ell$. Hence, if $\tau_{\ell,j}$ does not execute in $[t_d, t_d + y)$, then this pending work corresponds to that of $\tau_{\ell,j}$. That is, no job that can compete with $\tau_{\ell,j}$ is pending at $t_d + y$. Thus, the latest time that $\tau_{\ell,j}$ resumes (or begins, if $\delta_\ell = 0$) execution after t_d is at time $t_d + y$. By the claim above, $\tau_{\ell,j}$ cannot be preempted once it commences execution after t_d. Hence, $\tau_{\ell,j}$ completes execution at or before $t_d + y + e_\ell - \delta_\ell \leq t_d + x + e_\ell$.

Case 2: $[t_d, t_d + y)$ is not a busy interval for Ψ and $\mathcal{B}_J$. Refer to Figure 3(b) in following this case. Let t' denote the first non-busy instant in $[t_d, t_d + y)$. This implies the following.

(J) No job in $\overline{\Psi}$ begins executing a non-preemptive section or executes in a preemptive section in $[t_d, t')$; at most $M - 1$ tasks have jobs in Ψ or non-preemptive sections (of jobs in $\overline{\Psi}$) that commenced before t_d pending at or after t'.

Hence, no job of Ψ can be blocked by a job in $\overline{\Psi}$ at or after t'. Therefore, if $\tau_{\ell,j}$ has not completed executing before $t_d + y$, then either $\tau_{\ell,j}$ or a prior job of τ_ℓ should be executing at t'. If $\tau_{\ell,j}$ is executing at t', then because $t' < t_d + y$ holds, $\tau_{\ell,j}$ will complete executing before $t_d + y + e_\ell - \delta_\ell \leq t_d + x + e_\ell$. This is depicted in Figure 3(b). The remaining possibility is that $j > 1$ holds, and that a job of τ_ℓ that is prior to $\tau_{\ell,j}$ is executing at t'. In this case,

$\tau_{\ell,j}$ could not have executed before t_d, and hence $\delta_\ell = 0$ and $y = x$ holds. Thus, $t' < t_d + y = t_d + x$ holds. Let t_c denote the time at which $\tau_{\ell,j-1}$ completes executing. Then, by (R), $t_c \le t_d - p_\ell + x + e_\ell \le t_d + x$ holds. Because some prior job of τ_ℓ is executing at t', $t_c \ge t'$ holds. Hence, by (J), $\tau_{\ell,j}$ can commence execution at $t_c \le t_d + x$ (on the same processor as that on which $\tau_{\ell,j-1}$ executed), and hence, can complete executing by $t_d + x + e_\ell$. ■

We next determine an upper bound on the sum of the pending work and the competing work described above, *i.e.*, $\mathsf{LAG}(\Psi, t_d, \mathcal{S}) + \mathsf{B}(\tau, \Psi, t_d, \mathcal{S})$ (step (S1)).

3.2.2 Upper Bound on $\mathsf{LAG}(\Psi, t_d, \mathcal{S}) + \mathsf{B}(\tau, \Psi, t_d, \mathcal{S})$ (Step (S1))

By Lemma 1, the LAG of Ψ can increase only across a non-busy interval for Ψ. Similarly, if $\mathsf{B}(\tau, \Psi, t_d, \mathcal{S})$ is non-zero, then one or more jobs in $\overline{\Psi}$ should be executing in non-preemptive sections at t_d^-, that is, t_d^- should be a non-busy instant for Ψ. Hence, to determine an upper bound on the value that we are seeking, it suffices to consider only non-busy intervals for Ψ in $[0, t_d)$. As will be seen, an upper bound can be arrived at by reasoning about the number of tasks that can execute in and just before a non-busy interval and their lags. Towards this end, we prove a few lemmas below.

The lemma that follows identifies the existence of a class of jobs executing in a non-busy interval with continually increasing LAG, when no task has a non-preemptive segment. This lemma is somewhat technical in nature and is proved in an appendix. Informally, the lemma can be seen to hold by noting the following. For system LAG for Ψ to continually increase across a non-blocking, non-busy interval $[t, t')$, at each instant in $[t, t')$, there must exist at least one active job that is non-executing at that instant. For this to hold, there should exist at least one job J that has completed execution by the beginning of the interval and whose deadline is at or after the end of the interval. (This is proved in Lemma 12 in Appendix A.1.) Now, if no job executing in the interval has its deadline at or after the end of the interval, then removing J will not impact the schedule for any job executing in or after the interval, including that for $\tau_{\ell,j}$. Hence, the tardiness of $\tau_{\ell,j}$ will still be greater than $x + e_\ell$, contradicting (P2). A formal proof, which accounts for some cases not explained here, is presented in an appendix.

As noted earlier, this lemma provides a value to use for ρ, defined in (19), when $b_{\max} = 0$. In its absence, a weaker value of zero may be used.

Lemma 4 *Let $[t, t')$, where $t < t' \le t_d$, be a non-blocking, non-busy interval across which* LAG *for Ψ is continually increasing. If $b_{\max} = 0$, then there exists at least one job J such that J's deadline is at or after t' and J executes throughout $[\hat{t}, t')$, where $t \le \hat{t} < t'$.*

Because the total system lag, LAG, for τ is given by the sum of the lags of its constituent tasks (refer (10)), towards determining an upper bound on LAG, we determine upper bounds on the lags of individual tasks.

The next lemma bounds the lag of an arbitrary task at any arbitrary time at or before t_d.

Lemma 5 *Let t be an arbitrary time instant at or before t_d. Let τ_k be a task in τ and $\tau_{k,q}$ its earliest pending job at t in $\mathcal{S}$. If $d_{k,q} < t$ holds, then $\mathsf{lag}(\tau_k, t, \mathcal{S}) \le x \cdot u_k + e_k$, else $\mathsf{lag}(\tau_k, t, \mathcal{S}) \le e_k$.*

Proof: Let $\delta_{k,q} < e_k$ denote the amount of time that $\tau_{k,q}$ executed for before t. We determine the lag of τ_k at t by considering two cases depending on $d_{k,q}$.

Case 1: $d_{k,q} < t$. In PS_τ, $\tau_{k,q}$ completes execution at $d_{k,q}$, and the jobs of τ_k succeeding $\tau_{k,q}$ are allocated a share of u_k in every instant in their windows. Because different windows of the same task do not overlap, τ_k is allocated a share of u_k in every instant in $[d_{k,q}, t)$ in which it is active. Thus, the under-allocation to τ_k in $\mathcal{S}$ in $[0, t)$ is equal to the sum of the under-allocation to $\tau_{k,q}$ in $\mathcal{S}$, which is $e_k - \delta_{k,q}$, and the allocation, if any, to later jobs of τ_k in $[d_{k,q}, t)$ in PS_τ. Hence, we have

$$\mathsf{lag}(\tau_k, t, \mathcal{S}) \le e_k - \delta_{k,q} + (t - d_{k,q}) \cdot u_k. \tag{20}$$

If $\tau_{k,q}$ executes for a full e_k time units, then the earliest time that it can complete execution is $t + e_k - \delta_{k,q}$. Because $t \le t_d$, the deadline of $\tau_{k,q}$ is before t_d. Hence, by (P4), the tardiness of $\tau_{k,q}$ is at most $x + e_k$. Therefore, $t + e_k - \delta_{k,q} - d_{k,q} \le x + e_k$, *i.e.*, $t - d_{k,q} \le x + \delta_{k,q}$ holds. Substituting this value in (20), we have $\mathsf{lag}(\tau_k, t, \mathcal{S}) \le x \cdot u_k + e_k + \delta_{k,q}(u_k - 1)$. The lemma follows because $u_k \le 1$.

Case 2: $d_{k,q} \ge t$. In this case, the amount of work done by PS_τ on $\tau_{k,q}$ up to time t is given by $e_k - (d_{k,q} - t) \cdot u_k$. (Because $\tau_{k,q}$ is pending at t, $t \ge r_{k,q}$ holds.) Because all prior jobs of τ_k complete execution by t in both $\mathcal{S}$ and PS_τ, and $\tau_{k,q}$ has executed for $\delta_{k,q}$ time units before t, $\mathsf{lag}(\tau_k, t, \mathcal{S}) = e_k - (d_{k,q} - t) \cdot u_k - \delta_{k,q} \le e_k - \delta_{k,q} \le e_k$. ■

We next prove two claims, which will be used in proving later lemmas.

Claim 2 *Let $[t, t')$ be a maximally blocking, non-busy interval in $[0, t_d)$ in $\mathcal{S}$. Then, the following hold.* **(i)** *$t > 0$;* **(ii)** *any job that is in $\overline{\Psi}$ and is executing at $\hat{t}$ in $[t, t')$ executes a single non-preemptive section continuously in $[t^-, \hat{t}]$.*

Proof: Since every job of Ψ has an earlier deadline than any job in $\overline{\Psi}$, a job in Ψ cannot be blocked at time 0 by a job in $\overline{\Psi}$. Therefore $t > 0$ holds. By the nature of $[t, t')$, no job of Ψ (including jobs that are blocked at t) is blocked by a job in $\overline{\Psi}$ at t^-. Hence, it cannot be the case that a job in Ψ is blocked at t due to a job in $\overline{\Psi}$ commencing execution of a non-preemptive section at t (in preference to the blocked job). Rather, the blocking non-preemptive section should have commenced execution before t and some blocked job becomes ready at t (because, at t, the blocked job is either released or is preempted by a higher-priority job that is in Ψ). Similarly, since every instant in $[t, t')$ is a blocking instant at which one or more ready jobs of Ψ are waiting, no job in $\overline{\Psi}$ can be executing in a preemptive section and no non-preemptive section of a job in $\overline{\Psi}$ can commence execution in (t, t'). The claim follows from these facts. ■

Claim 3 *Let $[t, t')$ be a maximally blocking, non-busy interval in $[0, t_d)$ in $\mathcal{S}$ such that $\mathsf{LAG}(\Psi, t') > \mathsf{LAG}(\Psi, t)$. Then, at t and t^-, at most Λ tasks have jobs in Ψ that are executing. (Note that the tasks executing at t and t^-*

need not be the same.)

Proof: Because $\mathsf{LAG}(\Psi, t') > \mathsf{LAG}(\Psi, t)$ holds, by Lemma 1, (15), and (6), there exists at least one time instant $\hat{t}$ in $[t, t')$ such that at most Λ tasks have jobs in Ψ executing at $\hat{t}$. Let $k \leq \Lambda$ denote the number of such tasks. Then, since $[t, t')$ is a blocking, non-busy interval, no processor is idle in the interval. Hence, exactly $M - k$ jobs from $\overline{\Psi}$ are executing at $\hat{t}$. By Claim 2, $t > 0$ and each of the $M - k$ jobs is executing continuously in $[t^-, \hat{t}]$. Hence, at t^- and t, at most $k \leq \Lambda$ tasks may have executing jobs in Ψ. ■

In completing the derivation, we make use of the following two lemmas, which are proved in an appendix. These lemmas are purely algebraic and involve manipulations of task execution and non-preemptive segment costs and are independent of scheduling rules.

Lemma 6 *The following properties hold for sets $\Gamma(k+c, \ell)$ and $\Pi(k+c, \ell)$ with $0 \leq \ell \leq k \leq N$ and $0 \leq c \leq N-k$, where Γ and Π are as defined in (3) and (4), respectively.*

(i) $\sum_{\tau_i \in \Gamma(k+c,\ell)} e_i + \sum_{\tau_i \in \Pi(k+c,\ell)} b_i \leq \sum_{\tau_i \in \Gamma(k,\ell)} e_i + \sum_{\tau_i \in \pi(k,\ell)} b_i + \sum_{i=1}^{c} \beta_i$.

(ii) $\sum_{\tau_i \in \Gamma(k+c,\ell)} e_i + \sum_{\tau_i \in \Pi(k+c,\ell)} b_i \geq \sum_{\tau_i \in \Gamma(k,\ell)} e_i + \sum_{\tau_i \in \Pi(k,\ell)} b_i$.

Lemma 7 *Let α and β be any two **disjoint** subsets of tasks in τ such that $|\alpha| \leq \ell$ and $|\alpha| + |\beta| \leq k$, where $0 \leq \ell \leq k \leq N$. Then, $\sum_{\tau_i \in \alpha} e_i + \sum_{\tau_i \in \beta} b_i \leq \sum_{\tau_i \in \Gamma(k,\ell)} e_i + \sum_{\tau_i \in \Pi(k,\ell)} b_i$.*

The next two lemmas show how to bound LAG at the end of a non-blocking, non-busy interval, and $\mathsf{LAG} + \mathsf{B}$ at the end of a blocking, non-busy interval. In the first lemma, we bound LAG by simply determining a bound on the number of tasks that are pending at the end of the interval and summing their lag bounds. The second lemma is more involved and is described afterward.

Lemma 8 *Let $[t, t')$ be a non-blocking, non-busy interval in $[0, t_d)$ across which LAG for Ψ is continually increasing. Let k denote the number of processors (or tasks) that are executing jobs in Ψ at t'^-. Then, we have the following:* **(i)** *$k \leq \Lambda$; and* **(ii)** $\mathsf{LAG}(\Psi, t', \mathcal{S}) \leq \sum_{i=1}^{\Lambda-\rho} x \cdot \mu_i + \sum_{\tau_i \in \Gamma(\Lambda)} e_i \leq \sum_{i=1}^{\Lambda-\rho} x \cdot \mu_i + \sum_{\tau_i \in \Gamma(\Lambda+1,\Lambda)} e_i + \sum_{\tau_i \in \Pi(\Lambda+1,\Lambda)} b_i$.

Proof: Let α denote the subset of all tasks in τ that are executing at t'^-. Then, by the statement of the lemma, $|\alpha| = k$ holds. Because LAG is continually increasing across the interval $[t, t')$, by Lemma 2 and (15), we have

$$|\alpha| = k < U_{sum}(\Psi, t'^-) \leq U_{sum}(\tau). \tag{21}$$

Because k is an integer, by (21) and (6), we have $k \leq \Lambda$, which establishes Part (i).

Let Υ denote the set of all tasks that have a job with deadline at or after t' that is executing at t'^-. Because $[t, t')$ is a non-blocking, non-busy interval across which LAG is continually increasing, by Definition 10,

$$|\Upsilon| \geq \rho. \tag{22}$$

By (13), the LAG of Ψ at t' is at most the sum of the lags at t' of all tasks in τ with at least one job in Ψ that is pending at t'^{-}. Because $[t, t')$ is a non-blocking, non-busy interval, a task that is not executing at t'^{-} is not pending at that time. Hence, to determine an upper bound on LAG of Ψ at t', it suffices to determine an upper bound on the sum of the lags of tasks in α. Thus,

$$\begin{aligned}
\mathsf{LAG}(\Psi, t') &\leq \sum_{\tau_i \in \alpha} \mathsf{lag}(\tau_i, t', \mathcal{S}) \\
&= \sum_{\tau_i \in \Upsilon} \mathsf{lag}(\tau_i, t', \mathcal{S}) + \sum_{\tau_i \in \alpha \setminus \Upsilon} \mathsf{lag}(\tau_i, t', \mathcal{S}) \quad \text{(by the defns of } \alpha \text{ and } \Upsilon\text{)}.
\end{aligned}$$

The definition of Υ implies that the deadline of the earliest pending job at t' of every task in that set is at or after t'. Therefore, by Lemma 5, the lag of every task in Υ is at most its execution cost. Hence,

$$\begin{aligned}
\mathsf{LAG}(\Psi, t') &\leq \sum_{\tau_i \in \Upsilon} e_i + \sum_{\tau_i \in \alpha \setminus \Upsilon} (x \cdot u_i + e_i) && \text{(by the definition of } \Upsilon \text{ and Lemma 5)} \\
&= \sum_{\tau_i \in \alpha \setminus \Upsilon} x \cdot u_i + \sum_{\tau_i \in \alpha} e_i \\
&\leq \sum_{i=1}^{k-\rho} x \cdot \mu_i + \sum_{\tau_i \in \alpha} e_i && \text{(by (21), (22), and the definition of } \mu_i\text{)} \\
&\leq \sum_{i=1}^{\Lambda-\rho} x \cdot \mu_i + \sum_{\tau_i \in \alpha} e_i && (k \leq \Lambda, \text{ by Part (i)}) \\
&\leq \sum_{i=1}^{\Lambda-\rho} x \cdot \mu_i + \sum_{\tau_i \in \Gamma^{(k)}} e_i && \text{(by the definition of } \Gamma \text{ in (2))} \\
&\leq \sum_{i=1}^{\Lambda-\rho} x \cdot \mu_i + \sum_{\tau_i \in \Gamma^{(\Lambda)}} e_i && (k \leq \Lambda, \text{ by Part (i)}) \qquad (23) \\
&= \sum_{i=1}^{\Lambda-\rho} x \cdot \mu_i + \sum_{\tau_i \in \Gamma^{(\Lambda,\Lambda)}} e_i && \text{(by (2) and (3))} \\
&\leq \sum_{i=1}^{\Lambda-\rho} x \cdot \mu_i + \sum_{\tau_i \in \Gamma^{(\Lambda+1,\Lambda)}} e_i + \sum_{\tau_i \in \Gamma^{(\Lambda+1,\Lambda)}} b_i && \begin{array}{l}\text{(by Lemma 6(ii) with } \ell = k = \Lambda \leq N, \\ c = 1 \leq M + 1 - \Lambda \leq N - k).\end{array} \qquad (24)
\end{aligned}$$

Part (ii) follows from (23) and (24). ■

We now turn to determining bounds on LAG and $\mathsf{LAG} + \mathsf{B}$ at the end of a blocking, non-busy interval. To determine a bound on LAG, it may be simpler to apply the same approach, as used in the previous lemma, of summing the lag bounds of tasks with jobs in Ψ pending at the end of the interval. However, this approach suffers from the drawback that the number of tasks with positive lags at the end of a blocking, non-busy interval can be as high as $N - 1$. Consequently, the bound on LAG so obtained will be significantly loose. We can do much better by noting that Claims 2 and 3 provide a better bound of Λ on the number of tasks that can have jobs in Ψ pending immediately prior to the beginning of a maximally blocking, non-busy interval across which LAG increases. Let

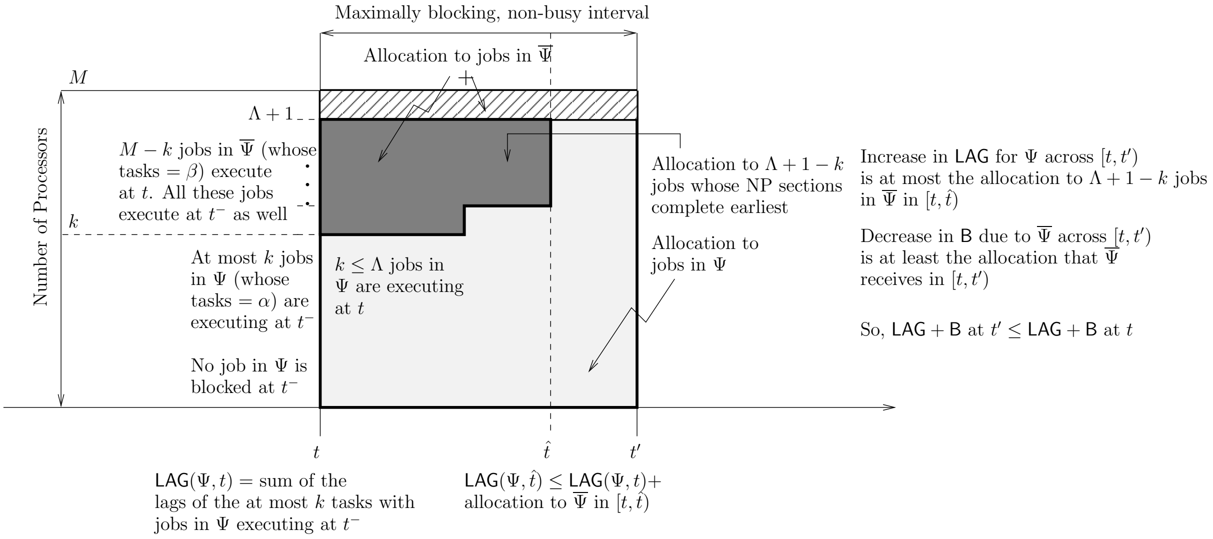


Figure 4: Scenario under consideration in the proof of Lemma 9.

the set of all such tasks be α. We therefore take an alternative approach that consists of the following steps. In following the ensuing description, the reader may refer to Figure 4. **(i)** Determine a bound on LAG for Ψ at the beginning, t, of a maximally blocking, non-busy interval, $[t, t')$, by summing the lag bounds of tasks in α. **(ii)** Determine a bound on B at t using maximum non-preemptive segment costs of tasks. Use the fact that there are at most M tasks in α and the set of all blocking tasks, say β, in the interval, to obtain a bound on $|\beta|$. (β is the set of all tasks whose jobs are executing in non-preemptive segments in the interval.) **(iii)** Determine a bound on LAG at the end of the interval by determining a bound on the deficit in allocation in the interval to jobs in Ψ in $\mathcal{S}$ in comparison to the allocation they receive in PS_τ. **(iv)** Finally, use the fact that the increase in the LAG for Ψ across the interval is at most the allocation in the interval to jobs of tasks in β (that is, the decrease in B) to show that $\mathsf{LAG}+\mathsf{B}$ at the end of the interval is at most that at the beginning of the interval. These arguments are laid out formally in the next lemma.

Lemma 9 *Let $[t, t')$ be a maximally blocking, non-busy interval in $[0, t_d)$ in $\mathcal{S}$ such that $k \leq \Lambda$ tasks have jobs in Ψ executing at t. Then, we have the following*: **(i)** $\mathsf{LAG}(\Psi, t') \leq \sum_{i=1}^{\Lambda-\rho} x \cdot \mu_i + \sum_{\tau_i \in \Gamma(\Lambda+1,\Lambda)} e_i + \sum_{\tau_i \in \Pi(\Lambda+1,\Lambda)} b_i$, *and* **(ii)** $\mathsf{LAG}(\Psi, t') + \mathsf{B}(\tau, \Psi, t') < \sum_{i=1}^{\Lambda-\rho} x \cdot \mu_i + \sum_{\tau_i \in \Gamma(M,\Lambda)} e_i + \sum_{\tau_i \in \Pi(M,\Lambda)} b_i$, *where ρ is as defined in* (19).

Proof: The reader may refer to Figure 4 in following the proof. We begin by determining a bound on the number of blocking tasks in $[t, t')$, which will be used later in the proof in determining a bound on B.

Bound on the number of blocking tasks in $[t, t')$. Let $\mathcal{J}$ denote the set of all jobs of $\overline{\Psi}$ that are executing in some non-preemptive section at t, and hence are blocking one or more jobs of Ψ. Let β denote the subset of all tasks in τ with jobs in $\mathcal{J}$. Since k jobs from Ψ are executing at t (by the statement of the lemma) and $[t, t')$ is a blocking, non-busy interval (hence, no processor is idle), we have

$$|\mathcal{J}| = |\beta| \;=\; M - k \tag{25}$$

$$\geq \;\Lambda + 1 - k \qquad \text{(by } U_{sum} \leq M \text{ and (6)).} \tag{26}$$

Let α denote the set of all tasks with jobs in Ψ that are executing at t^-. We next determine a bound on the number of tasks in α and β put together.

Bound on $|\boldsymbol{\alpha}| + |\boldsymbol{\beta}|$. By the definition of $[t, t')$ in the statement of the lemma and Claim 2, it follows that t^- is a non-blocking, non-busy instant. By (25) and Claim 2 again, it follows that at least $M - k$ jobs of $\overline{\Psi}$, including all the jobs in $\mathcal{J}$, are executing at t^-. Therefore, at most k jobs from Ψ can be executing at t^-, that is, at most k tasks can be in α. Hence,

$$|\alpha| \leq k. \tag{27}$$

Note that by their definitions, sets α and β are disjoint and the following hold.

$$\alpha \cap \beta \;=\; \emptyset \tag{28}$$

$$|\alpha| + |\beta| \;\leq\; M \tag{29}$$

Bound on LAG at t. Since no job of Ψ that is not executing at a non-blocking, non-busy instant can be pending (and hence no job that is not executing at t^- or a task with no job executing at t^- may have a positive lag at t), by (13), we have the following.

$$\begin{aligned}
\mathsf{LAG}(\Psi, t) &\leq \sum_{\substack{\{\tau_i \in \tau \mid \text{ some } \tau_{i,j} \text{ is in} \\ \Psi \text{ and is executing at} \\ t^-\}}} \mathsf{lag}(\tau_i, t, \mathcal{S}) \\
&\leq \sum_{\substack{\{\tau_i \in \tau \mid \text{ some } \tau_{i,j} \text{ is in} \\ \Psi \text{ and is executing at} \\ t^-\}}} (x \cdot u_i + e_i) && \text{(by Lemma 5)} \\
&\leq \sum_{\tau_i \in \alpha} (x \cdot u_i + e_i) && \text{(by the definition of } \alpha\text{)}
\end{aligned} \tag{30}$$

Bound on B at t. By Claim 2, every job of $\overline{\Psi}$ that is executing anywhere in $[t, t')$ is in $\mathcal{J}$. Furthermore, every such job executes a single non-preemptive section in $[t^-, t')$. Hence, for any time u in $[t, t')$, $\mathsf{B}(\tau, \Psi, u)$ is given by the amount of work pending at u for the non-preemptive sections executing at t of jobs in $\mathcal{J}$. Let τ_i be a task with a job J in $\mathcal{J}$. Then, the amount of work that can be pending for the non-preemptive section of J executing at t is less than b_i. Therefore, by the definition of set β, we have the following.

$$\mathsf{B}(\tau, \Psi, t) < \sum_{\tau_i \in \beta} b_i \tag{31}$$

Bound on $\mathsf{LAG} + \mathsf{B}$ at t. By (30) and (31), we have

$$\begin{aligned}
&\mathsf{LAG}(\Psi, t) + \mathsf{B}(\tau, \Psi, t) \\
&< \sum_{\tau_i \in \alpha} (x \cdot u_i + e_i) + \sum_{\tau_i \in \beta} b_i \\
&= \sum_{\tau_i \in \alpha} x \cdot u_i + \sum_{\tau_i \in \alpha} e_i + \sum_{\tau_i \in \beta} b_i \\
&\le \sum_{i=1}^{\Lambda} x \cdot \mu_i + \sum_{\tau_i \in \Gamma^{(M,\Lambda)}} e_i + \sum_{\tau_i \in \Pi^{(M,\Lambda)}} b_i && \text{(by Lemma 7, because } \alpha \text{ and } \beta \text{ are disjoint, and by (27) and the stmt. of the lemma, } |\alpha| \le k \le \Lambda\text{, and by (29), } |\alpha| + |\beta| \le M) \\
&= \sum_{i=1}^{\Lambda-\rho} x \cdot \mu_i + \sum_{\tau_i \in \Gamma^{(M,\Lambda)}} e_i + \sum_{\tau_i \in \Pi^{(M,\Lambda)}} b_i && \text{(} [t, t') \text{ is a blocking interval, so } b_{\max} > 0\text{, and by (19), } \rho = 0). \qquad (32)
\end{aligned}$$

Finally, we are left with determining an upper bound on LAG and the sum of LAG and B at t'. We will determine an upper bound on LAG first.

Bound on LAG at t'. Let $\hat{t}$ denote the earliest time, if any, in (t, t') at which at least $\Lambda + 1$ jobs from Ψ are executing. If no such time exists, then let $\hat{t} = t'$. By the nature of $[t, t')$ and Claim 2, no job in $\overline{\Psi}$ can begin executing (either a preemptive or non-preemptive section) anywhere in $[t, t')$, and hence, at least $\Lambda + 1$ jobs from Ψ are executing at each instant in $[\hat{t}, t')$. By (6) and (15), $\Lambda + 1 \ge U_{sum}(\tau) \ge U_{sum}(\Psi, u)$, for all u. Hence, by Lemma 1, LAG for Ψ does not increase across $[\hat{t}, t')$. That is,

$$\mathsf{LAG}(\Psi, t', \mathcal{S}) \le \mathsf{LAG}(\Psi, \hat{t}, \mathcal{S}). \tag{33}$$

Thus, to determine an upper bound on LAG at t', it suffices to determine an upper bound on LAG at $\hat{t}$.

Let X denote the total execution within $[t, t')$ of the first $\Lambda + 1 - k$ jobs in $\mathcal{J}$ to complete executing their non-preemptive sections (the actual completion times of these non-preemptive sections is immaterial and can be after t'). Note that X is well defined because by (26), $|\mathcal{J}| \ge \Lambda + 1 - k$ holds. Because $[t, t')$ is maximally blocking, no processor is idle in $[t, \hat{t})$, and because k jobs in Ψ are executing at t, at least k jobs in Ψ execute at every instant in $[t, t')$. Also, when a job in $\mathcal{J}$ completes executing its non-preemptive section in $[t, t')$, the processor it executed on is allocated to a waiting job in Ψ. Hence, the total time allocated to jobs in Ψ in $[t, \hat{t})$, $\mathsf{A}(\mathcal{S}, \Psi, t, \hat{t})$, is equal to $k \cdot (\hat{t} - t) + (\Lambda + 1 - k) \cdot (\hat{t} - t) - X = (\Lambda + 1) \cdot (\hat{t} - t) - X$. In PS_τ, jobs in Ψ could execute for at most $U_{sum}(\tau) \cdot (\hat{t} - t)$

time, *i.e.*, $\mathsf{A}(\mathsf{PS}_\tau, \Psi, t, \hat{t}) \leq U_{sum}(\tau)\cdot(\hat{t}-t) \leq (\Lambda+1)\cdot(\hat{t}-t)$. (The second inequality is by (6).) Therefore, by (11),

$$\begin{aligned}
\mathsf{LAG}(\Psi, \hat{t}) &= \mathsf{LAG}(\Psi, t) + \mathsf{A}(\mathsf{PS}_\tau, \Psi, t, \hat{t}) - \mathsf{A}(\mathcal{S}, \Psi, t, \hat{t}) \\
&\leq \mathsf{LAG}(\Psi, t) + (\Lambda+1)\cdot(\hat{t}-t) - (\Lambda+1)\cdot(\hat{t}-t) + X \\
&= \mathsf{LAG}(\Psi, t) + X. \qquad (34)
\end{aligned}$$

Let $\beta^{(\Lambda+1-k)}$ denote the subset of $\Lambda+1-k$ tasks whose jobs in $\mathcal{J}$ complete executing their non-preemptive sections the earliest. (Again, by (26), $|\mathcal{J}| \geq \Lambda+1-k$.) Then, by the discussion thus far,

$$\begin{aligned}
\mathsf{LAG}(\Psi, t', \mathcal{S}) &\leq \mathsf{LAG}(\Psi, \hat{t}, \mathcal{S}) && \text{(by (33))} \\
&\leq \mathsf{LAG}(\Psi, t) + X && \text{(by (34))} \\
&\leq \sum_{\tau_i\in\alpha}(x\cdot u_i + e_i) + \sum_{\tau_i\in\beta^{(\Lambda+1-k)}} b_i && \text{(by (30) and the definition of } X\text{)} \\
&\leq \sum_{i=1}^{k} x\cdot\mu_i + \sum_{\tau_i\in\alpha} e_i + \sum_{\tau_i\in\beta^{(\Lambda+1-k)}} b_i && \text{(by (27) and the definition of } \mu_i\text{)} \\
&\leq \sum_{i=1}^{\Lambda} x\cdot\mu_i + \sum_{\tau_i\in\Gamma^{(\Lambda+1,\Lambda)}} e_i + \sum_{\tau_i\in\Pi^{(\Lambda+1,\Lambda)}} b_i && \begin{array}{l}\text{(by Lemma 7; note that } \alpha \text{ and } \beta^{(\Lambda+1-k)} \\ \text{are disjoint, } |\alpha| \leq k \leq \Lambda \text{ holds by (27),} \\ \text{and because } |\beta^{(\Lambda+1-k)}| = \Lambda+1-k, \\ |\alpha| + |\beta^{(\Lambda+1-k)}| \leq \Lambda+1 \text{ holds as well)}\end{array} \\
&\leq \sum_{i=1}^{\Lambda-\rho} x\cdot\mu_i + \sum_{\tau_i\in\Gamma^{(\Lambda+1,\Lambda)}} e_i + \sum_{\tau_i\in\Pi^{(\Lambda+1,\Lambda)}} b_i && \text{(by (19), } \rho = 0 \text{ since } b_{\max} > 0\text{).}
\end{aligned}$$

The above establishes Part (i) of the lemma.

Bound on $\mathsf{LAG}+\mathsf{B}$ at t'. To determine an upper bound on LAG plus B at t', let $X' \leq \mathsf{B}(\tau, \Psi, t)$ denote the total amount of time that jobs in $\mathcal{J}$ execute on all M processors in $[t, t')$. Then, the total time allocated to jobs in Ψ in $[t, t')$, $\mathsf{A}(\mathcal{S}, \Psi, t, t')$, is equal to $M\cdot(t'-t) - X'$. In PS_τ, jobs in Ψ could execute for at most $U_{sum}(\tau)\cdot(t'-t)$ time, *i.e.*, $\mathsf{A}(\mathsf{PS}_\tau, \Psi, t, t') \leq U_{sum}(\tau)\cdot(t'-t)$. Therefore, $\mathsf{LAG}(\Psi, t') = \mathsf{LAG}(\Psi, t) + \mathsf{A}(\mathsf{PS}_\tau, \Psi, t, t') - \mathsf{A}(\mathcal{S}, \Psi, t, t') \leq \mathsf{LAG}(\Psi, t) + (U_{sum}(\tau) - M)\cdot(t'-t) + X' \leq \mathsf{LAG}(\Psi, t) + X'$. Since jobs in $\mathcal{J}$ execute for a total time of X' in $[t, t')$, the pending work for non-preemptive sections of jobs in $\mathcal{J}$, and hence, those in $\mathcal{B}(\tau, \Psi, t')$ at t', *i.e.*, $\mathsf{B}(\tau, \Psi, t')$, is at most $\mathsf{B}(\tau, \Psi, t) - X'$. Thus, $\mathsf{LAG}(\Psi, t') + \mathsf{B}(\tau, \Psi, t') \leq \mathsf{LAG}(\Psi, t) + \mathsf{B}(\tau, \Psi, t)$, which by (32), establishes Part (ii) of the lemma. ■

Finally, Lemmas 8 and 9 can be used to establish the desired upper bound on $\mathsf{LAG}+\mathsf{B}$ at t_d.

Lemma 10 $\mathsf{LAG}(\Psi, t_d) + \mathsf{B}(\tau, \Psi, t_d) \leq \sum_{i=1}^{\Lambda-\rho} x\cdot\mu_i + \sum_{\tau_i\in\Gamma^{(\Lambda+1,\Lambda)}} e_i + \sum_{\tau_i\in\Pi^{(\Lambda+1,\Lambda)}} b_i + \sum_{i=1}^{M-\Lambda-1}\beta_i$, *where* ρ *is as defined in* (19).

Proof: We consider the following cases based on the nature of t_d^-.

Case 1: t_d^- is a busy instant or a non-blocking, non-busy instant. In this case, neither t_d^- nor t_d is a blocking instant. Hence, by definition, $\mathsf{B}(\tau, \Psi, t_d) = 0$. By Lemma 1, the LAG of Ψ can increase only across a non-busy interval. Therefore, LAG at t_d is at most that at the end of the latest non-busy instant before t_d. If no non-blocking, non-busy interval or maximally blocking, non-busy interval across which LAG increases exists in $[0, t_d)$, then $\mathsf{LAG}(\Psi, t_d) \leq \mathsf{LAG}(\Psi, 0) = 0$. (Since every non-maximal, blocking interval is contained in some maximal, blocking interval, it suffices to consider only maximally blocking intervals.) Otherwise, if the latest such non-busy interval $[t, t')$ that is in $[0, t_d)$ is non-blocking, then LAG at t_d is at most LAG at t', which, by Part (ii) of Lemma 8, is at most $\sum_{i=1}^{\Lambda-\rho} x \cdot \mu_i + \sum_{\tau_i \in \Gamma^{(\Lambda+1,\Lambda)}} e_i$, establishing the lemma in this case. (If t_d^- is a non-blocking instant, then t' could be t_d.) The remaining case is that the latest such non-busy interval before t_d is maximally blocking across which LAG increases, in which case by Part (i) of Lemma 9, LAG is at most $\sum_{i=1}^{\Lambda-\rho}(x \cdot \mu_i) + \sum_{\tau_i \in \Gamma^{(\Lambda+1,\Lambda)}} e_i + \sum_{\tau_i \in \Pi^{(\Lambda+1,\Lambda)}} b_i$. The lemma thus holds in this case too. (In invoking Lemma 9, $k \leq \Lambda$ holds, where k is the number of tasks with jobs in Ψ executing at the beginning of the maximally blocking, non-busy interval. This is by Claim 3, because LAG is increasing across the interval.)

Case 2: t_d^- is a blocking, non-busy instant. Let $t < t_d$ be the earliest instant before t_d such that $[t, t_d)$ is a maximally blocking, non-busy interval. Let k denote the number of tasks whose jobs in Ψ are executing at t. We consider the following two subcases.

Subcase 2(a): $\mathsf{LAG}(\Psi, t_d) > \mathsf{LAG}(\Psi, t)$ or $k \leq \Lambda$. If $\mathsf{LAG}(\Psi, t_d) > \mathsf{LAG}(\Psi, t)$ holds, then by Claim 3, $k \leq \Lambda$ holds. Hence, in either case, by Part (ii) of Lemma 9, we have

$$\begin{aligned}
\mathsf{LAG}(\Psi, t_d) + \mathsf{B}(\tau, \Psi, t_d) &< \sum_{i=1}^{\Lambda-\rho} x \cdot \mu_i + \sum_{\tau_i \in \Gamma^{(M,\Lambda)}} e_i + \sum_{\tau_i \in \Pi^{(M,\Lambda)}} b_i \\
&\leq \sum_{i=1}^{\Lambda-\rho} x \cdot \mu_i + \sum_{\tau_i \in \Gamma^{(\Lambda+1,\Lambda)}} e_i + \sum_{\tau_i \in \Pi^{(\Lambda+1,\Lambda)}} b_i + \sum_{i=1}^{M-\Lambda-1} \beta_i,
\end{aligned}$$

establishing the lemma for this subcase. The second inequality follows from Lemma 6(i) (because, by (6) and $M \leq U_{sum}$, $M \geq \Lambda + 1$).

Subcase 2(b): $k > \Lambda$ and $\mathsf{LAG}(\Psi, t_d) \leq \mathsf{LAG}(\Psi, t)$. By the conditions of this subcase, it follows that LAG at t_d is bounded from above by the LAG at the end of the latest non-blocking or maximally blocking, non-busy interval before t across which LAG increases. If no such interval exists, then by Lemma 1, $\mathsf{LAG}(\tau, t_d) \leq \mathsf{LAG}(\tau, 0) = 0$.

Otherwise, (as with Case 1) by Part (ii) of Lemma 8 and Part (i) of Lemma 9, we have

$$\mathsf{LAG}(\Psi, t_d) \leq \sum_{i=1}^{\Lambda-\rho} x \cdot \mu_i + \sum_{\tau_i \in \Gamma^{(\Lambda+1,\Lambda)}} e_i + \sum_{\tau_i \in \Pi^{(\Lambda+1,\Lambda)}} b_i. \tag{35}$$

Since $k > \Lambda$ holds, at most $M - \Lambda - 1$ jobs from $\overline{\Psi}$ can be executing at t and by Claim 2, at most $M - \Lambda - 1$ such jobs can be executing at t_d^- as well. The amount of time for which such a job of task τ_i can execute past t_d in its non-preemptive section is less than b_i. Thus, $\mathsf{B}(\tau, \Psi, t_d) < \sum_{i=1}^{M-\Lambda-1} \beta_i$ holds. Therefore, by (35), $\mathsf{LAG}(\Psi, t_d) + \mathsf{B}(\tau, \Psi, t_d) < \sum_{i=1}^{\Lambda-\rho} x \cdot \mu_i + \sum_{\tau_i \in \Gamma^{(\Lambda+1,\Lambda)}} e_i + \sum_{\tau_i \in \Pi^{(\Lambda+1,\Lambda)}} b_i + \sum_{i=1}^{M-\Lambda-1} \beta_i$ holds. ■

3.3 Finishing Up (Step (S3))

By (P1) and Lemma 3, $\mathsf{LAG}(\tau, t_d, \mathcal{S}) + \mathsf{B}(\Psi, \tau, t_d, \mathcal{S}) > M \cdot x + e_\ell$. Therefore, by Lemma 10, $M \cdot x + e_\ell < \sum_{i=1}^{\Lambda-\rho} x \cdot \mu_i + \sum_{\tau_i \in \Gamma^{(\Lambda+1,\Lambda)}} e_i + \sum_{\tau_i \in \Pi^{(\Lambda+1,\Lambda)}} b_i + \sum_{i=1}^{M-\Lambda-1} \beta_i$, *i.e.*,

$$x < \frac{\sum_{\tau_i \in \Gamma^{(\Lambda+1,\Lambda)}} e_i + \sum_{\tau_i \in \Pi^{(\Lambda+1,\Lambda)}} b_i + \sum_{i=1}^{M-\Lambda-1} \beta_i - e_\ell}{M - \sum_{i=1}^{\Lambda-\rho} \mu_i} \leq \frac{\sum_{\tau_i \in \Gamma^{(\Lambda+1,\Lambda)}} e_i + \sum_{\tau_i \in \Pi^{(\Lambda+1,\Lambda)}} b_i + \sum_{i=1}^{M-\Lambda-1} \beta_i - e_{\min}}{M - \sum_{i=1}^{\Lambda-\rho} \mu_i},$$

which contradicts (18), implying that unless x is less than the quantity on the right-hand side of (18), (P1) cannot hold. Theorem 1 below therefore follows.

Theorem 1 *Let τ be a sporadic task system with non-preemptable segments and $U_{sum} \leq M$. Then, on M processors,* EDF-P-NP *ensures a tardiness of at most $x + e_k$ to every task τ_k in τ, where x is as defined by* (18).

EDF-P-NP reduces to g-EDF (fully-preemptive EDF) if no task has non-preemptable segments. In such a case, $b_i = \beta_i = 0$, for all i. Hence, $b_{\max} = 0$, and by Lemma 4, $\rho \geq 1$. Futher $e_i - b_i = e_i$, and hence, $\Gamma^{(\Lambda+1,\Lambda)}$ is simply a subset of Λ tasks of τ with the highest execution costs, *i.e.*, $\Gamma^{(\Lambda+1,\Lambda)} = \Gamma^{(\Lambda)}$. Therefore, a tardiness bound under g-EDF is given by the following corollary to Theorem 1.

Corollary 1 g-EDF *ensures a tardiness bound of*

$$\frac{\sum_{i=1}^{\Lambda} \epsilon_i - e_{\min}}{M - \sum_{i=1}^{\Lambda-1} \mu_i} + e_k \tag{36}$$

to every task τ_k of a sporadic task system τ with $U_{sum} \leq M$.

A sufficient condition on the individual task utilizations for a given tardiness bound under g-EDF that places no restriction on the total utilization can be derived from the above corollary, as given below.

Corollary 2 g-EDF *ensures a tardiness of at most $x_p + e_k$ for every task τ_k of a sporadic task system τ on M processors if the sum of the utilizations of the $\Lambda - 1$ tasks with highest utilizations, $\sum_{i=1}^{\Lambda-1} \mu_i$, is at most $M - \frac{\sum_{i=1}^{\Lambda} \epsilon_i + e_{\min}}{x_p}$ and $U_{sum}(\tau) \leq M$.*

Proof: By Corollary 1, the tardiness for task τ_k of τ in a g-EDF schedule is at most $\frac{\sum_{i=1}^{\Lambda} \epsilon_i - e_{\min}}{M - \sum_{i=1}^{\Lambda-1} \mu_i} + e_k$. Therefore, a tardiness not exceeding $x_p + e_k$ can be guaranteed if $\frac{(\sum_{i=1}^{\Lambda} \epsilon_i) - e_{\min}}{M - \sum_{i=1}^{\Lambda-1} \mu_i} + e_k \leq x_p + e_k$ holds. On rearranging the terms, we arrive at the condition in the corollary. ■

Similarly, EDF-P-NP reduces to g-NP-EDF (fully-non-preemptive EDF) if $b_i = e_i$, for all i. Further, $\Gamma^{(\Lambda+1,\Lambda)} \cup \Pi^{(\Lambda+1,\Lambda)} = \Gamma^{(\Lambda+1)}$, and by (19), $\rho = 0$ holds. Thus, in this case, the tardiness bound of Theorem 1 reduces to the following.

Corollary 3 *On M processors,* g-NP-EDF *ensures a tardiness bound of*

$$\frac{\sum_{i=1}^{\Lambda+1} \epsilon_i + \sum_{i=1}^{M-\Lambda-1} \beta_i - e_{\min}}{M - \sum_{i=1}^{\Lambda} \mu_i} + e_k$$

to every task τ_k of a sporadic task system τ with $U_{sum} \leq M$.

3.4 Tardiness Bound under g-EDF for Two-Processor Systems

If $1 < U_{sum} \leq 2$, then the tardiness bound under g-EDF given by Corollary 1 for task τ_k under two processors (*i.e.*, when $M = 2$) is $(e_{\max} - e_{\min})/2 + e_k$. However, an improved bound of $e_{\max}/2 + e_k/2$ can be ensured if we note that $\mathsf{LAG} + \mathsf{B}$ at t_d, as given by Lemma 10, is at most $e_{\max}$, which is independent of x. The improved bound is derived in the following theorem.

Theorem 2 *On two processors,* g-EDF *ensures a tardiness bound of $\frac{e_{\max}+e_k}{2}$ for every task τ_k of every sporadic task system with $U_{sum} \leq 2$.*

Proof: Suppose the theorem does not hold. Then, there exists a concrete instantiation τ of a non-concrete task system with $U_{sum} \leq 2$ such that $\tau_{\ell,j}$ is a job of task τ_ℓ in τ, $\mathcal{S}$ is a g-EDF schedule for τ, $t_d = d_{\ell,j}$, and (P1)–(P4) (defined in the beginning of Section 3.2) hold, where $x = \frac{e_{\max}-e_\ell}{2}$. By Lemma 10, because $\rho = 1$ by Lemma 4, $b_i = 0$, for all i, and $\Lambda = 1$ (by (6) because $U_{sum} \leq M \leq 2$) we have

$$\mathsf{LAG}(\tau, t_d, \mathcal{S}) + \mathsf{B}(\Psi, \tau, t_d, \mathcal{S}) \leq e_{\max}. \tag{37}$$

By (P1), *tardiness*$(\tau_{\ell,j}) > x + e_\ell$. Hence, by the contrapositive of Lemma 3, we have $\mathsf{LAG}(\tau, t_d, \mathcal{S}) + \mathsf{B}(\Psi, \tau, t_d, \mathcal{S}) > 2 \cdot x + e_\ell$, which, by (37), implies that $e_{\max} > 2 \cdot x + e_\ell = e_{\max}$, a contradiction. The theorem, therefore, follows. ■

3.5 Improving Accuracy and Speed

In this subsection, we discuss possible improvements to the accuracy of the tardiness bounds of g-EDF and g-NP-EDF given in Corollaries 1 and 3, respectively, and the time required to compute them. We will refer to the bounds given by Corollaries 1 and 3 as EDF-BASIC and NP-EDF-BASIC, respectively.

Improving accuracy. If the time taken to compute a tardiness bound or sufficient task utilizations for a given tardiness bound is not a concern, then some improvement to the values may be possible by relaxing a pessimistic assumption that we make in Lemmas 8 and 9. We not only assume that the tasks that are executing at the end of the non-blocking, non-busy interval in Lemma 8, and just before the beginning of the blocking, non-busy interval in Lemma 9 have the highest utilizations, but also that they have the highest execution costs. For g-EDF, this can be relaxed by sorting tasks by non-increasing order of $x \cdot u_k + e_k$ (where, as defined in (P4), the tardiness of τ_k is at most $x + e_k$) and by using the utilizations and execution costs of the top $\Lambda - 1$ tasks in the expression in Corollary 1. (The Λ^{th} execution cost should be taken as the maximum of the execution costs of the remaining tasks.) If x is known (*i.e.*, when determining utilization restrictions for a given tardiness), as in applying Corollary 2, then this procedure is straightforward. Nevertheless, even when seeking x (*i.e.*, when determining a tardiness bound), as in Corollary 1, an iterative procedure could be used. In this iterative procedure, the bound given by EDF-BASIC will be used as the initial value for x. This initial value will then be used to determine the set of tasks whose utilizations and execution costs should be used in improving x. The procedure should be continued until the task set converges. (It is easy to show that convergence is guaranteed.) We will refer to the bound computed using such an iterative procedure as EDF-ITER. This procedure is illustrated in the example below.

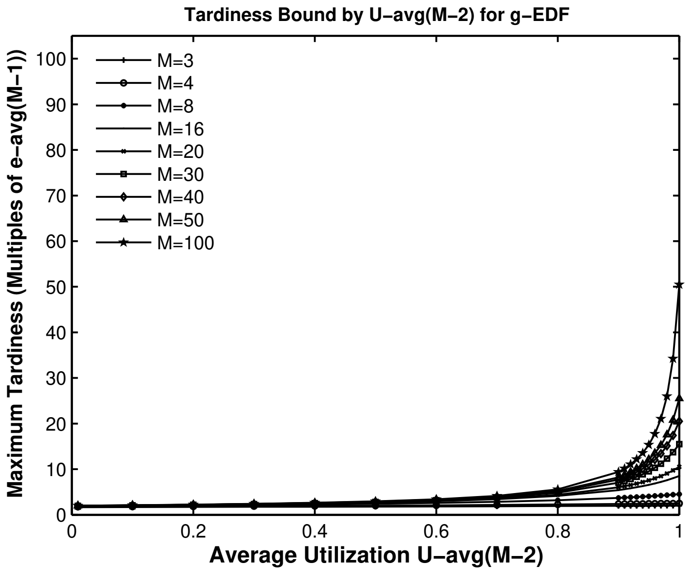
Example. Let τ be a task set comprised of the following eight tasks with $U_{sum} = 4.0$ scheduled under g-EDF on $M = 4$ processors: $\tau_1(15, 150), \ldots, \tau_4(15, 150)$, $\tau_5(9, 10), \ldots, \tau_8(9, 10)$. For this task set, $\mu_1 = \mu_2 = 0.9$, $\epsilon_1 = \epsilon_2 = \epsilon_3 = 15$, and $\Lambda = 3$. Therefore, a tardiness bound for τ_k obtained using EDF-BASIC is $(45 - 9)/(4 - 18/10) + e_k$, which equals $360/22 + e_k$. Thus, $x \approx 16.36$. Computing $x \cdot u_k + e_k$ for each task τ_k, we obtain values of 16.636 for tasks $\tau_1, \ldots, \tau_4$, and 23.727 for $\tau_5, \ldots, \tau_8$. Hence, the two tasks with the highest value for $x \cdot u_k + e_k$ are τ_5 and τ_6. Using the execution costs and utilizations of τ_5 and τ_6 and 15 as the Λ^{th} execution cost, we obtain an improved value of 10.9 for x, which is more than 5 units less than the initial value. The set of tasks with the highest value for $x \cdot u_k + e_k$ is not altered in the next iteration, and so, the procedure terminates.

A similar procedure can be used for computing an improved bound for g-NP-EDF also, except that Λ tasks τ_k with the highest values for $x \cdot u_k + e_k$ have to be considered.

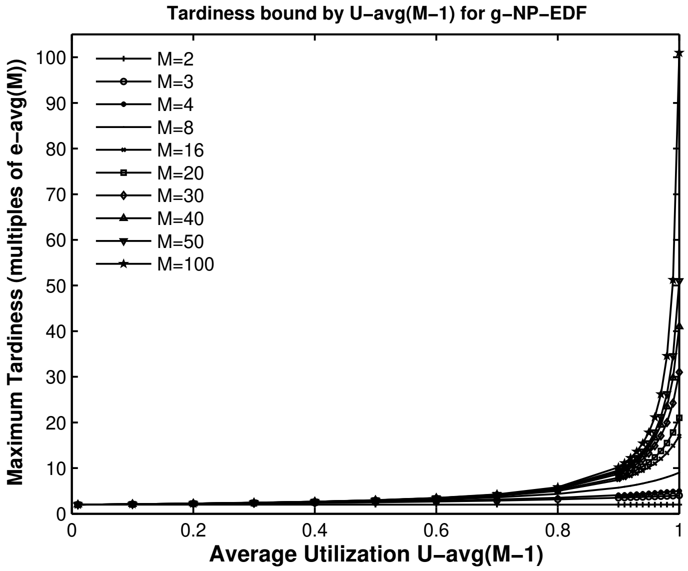
Improving computation time. Computing the tardiness bound or determining utilization restrictions on tasks for a given bound involves selecting $O(M)$ tasks with the highest utilizations, and $O(M)$ tasks with the highest execution costs (or the highest values for $e_i - b_i$), and summing these values. Because worst-case [Blum et al., 1973] and expected [Floyd and Rivest, 1975, Hoare, 1961] linear-time selection algorithms are known, these are $O(N + M)$ computations. If speed is a concern, as in online admission tests, and if $u_{\max}$ and $e_{\max}$ are known, then $O(1)$ computations, which assume a utilization of $u_{\max}$ and an execution cost of $e_{\max}$ for each of the $M - 1$ tasks, can be used, at the expense of more pessimistic values. Under this assumption, tardiness under g-EDF and g-NP-EDF is at most $((M-1)e_{\max} - e_{\min})/(M-(M-2)u_{\max}) + e_i$ and $(M \cdot e_{\max} - e_{\min})/(M-(M-1)u_{\max}) + e_i$, respectively, for each task τ_i. Similarly, for a tardiness of at most $x + e_i$ for every τ_i under g-EDF, it is sufficient that $u_{\max}$ be at most $(M \cdot x - (M-1)e_{\max} + e_{\min})/((M-2)x)$. We will refer to the bounds computed using $u_{\max}$ and $e_{\max}$ as EDF-FAST and NP-EDF-FAST.

Tightness of the bound. As discussed in the introduction, we do not believe that our results are tight for either g-EDF or g-NP-EDF. Let $u_{\text{avg}}(k)$ (resp., $e_{\text{avg}}(k)$) denote the average of the k highest task utilizations (resp., execution costs). For fixed values of $u_{\text{avg}}(M-2)$ and $e_{\text{avg}}(M-1)$ (resp., $u_{\text{avg}}(M-1)$ and $e_{\text{avg}}(M)$), it can be verified that the bound derived for g-EDF (resp., g-NP-EDF) is an increasing function of M when $U_{sum} = M$. In addition, for a given M, the g-EDF (resp., g-NP-EDF) bound is an increasing function of $u_{\text{avg}}(M-2)$ and $e_{\text{avg}}(M-1)$ (resp., $u_{\text{avg}}(M-1)$ and $e_{\text{avg}}(M)$). Approximate plots of the tardiness bounds computed for g-EDF and g-NP-EDF for varying values of M and u_{avg}, assuming $U_{sum} = M$, are shown in Figures 5 and 6. The plotted bounds are in multiples of $e_{\text{avg}}(M-1)$ for g-EDF and $e_{\text{avg}}(M)$ for g-NP-EDF. As can be seen from the plots, the bounds are quite reasonable unless both of M and u_{avg} are high. For instance, for g-EDF, if u_{avg} is at most 0.75 (resp., 0.5), then the bound guaranteed is approximately at most $5 \cdot e_{\text{avg}}$ (resp., $3 \cdot e_{\text{avg}}$) for all M, and if M is at most 8, then the bound is approximately at most $4 \cdot e_{\text{avg}}$ even when u_{avg} is close to 1.0. If $M = 8$, then the tardiness bound for g-EDF is at most $3 \cdot e_{\text{avg}}$ (resp., $2 \cdot e_{\text{avg}}$) when u_{avg} is at most 0.75 (resp., 0.5). Of course, whether that much tardiness is tolerable is application dependent. However, if execution costs are low, say of the order of a few milliseconds, as can be expected with modern processors, then in many cases, the tardiness bounds guaranteed can be expected to be within limits.

Though the bounds may not be tight, in general, we can show that our result for g-EDF is within approximately $e_{\max}$ of tardiness possible in the worst case for small values of $u_{\max}$. For this, consider a task system τ that consists of a primary task $\tau_1(e_1, p_1)$ and $(M - u_1)p_1/\delta$ auxiliary tasks. Let $\delta \ll e_1$ and p_1 be the execution cost and period, respectively, of each auxiliary task. Let δ divide $(M - u_1)$ evenly. The total utilization of this task system is $(\delta/p_1) \times ((M - u_1)p_1/\delta) + u_1 = M$. Suppose that the first job of each task is released at time 0 and suppose that the first job of every auxiliary task is scheduled before $\tau_{1,1}$ and executes for a full δ time units. In such a schedule, the auxiliary jobs will execute continuously until time $(\frac{(M-u_1)p_1}{\delta} \times \delta)/M = p_1 - e_1/M$ on each processor. Hence, τ_1 will not begin executing until time $p_1 - e_1/M$, and hence would not complete until $p_1 + e_1(1 - 1/M)$ for a tardiness of $e_1(1 - 1/M)$. By choosing $e_1 = M$, this tardiness can be made to equal $e_1 - 1$. In this example, $e_{\max} = e_1$, and hence tardiness is at least $e_{\max} - 1$. In the example schedule described



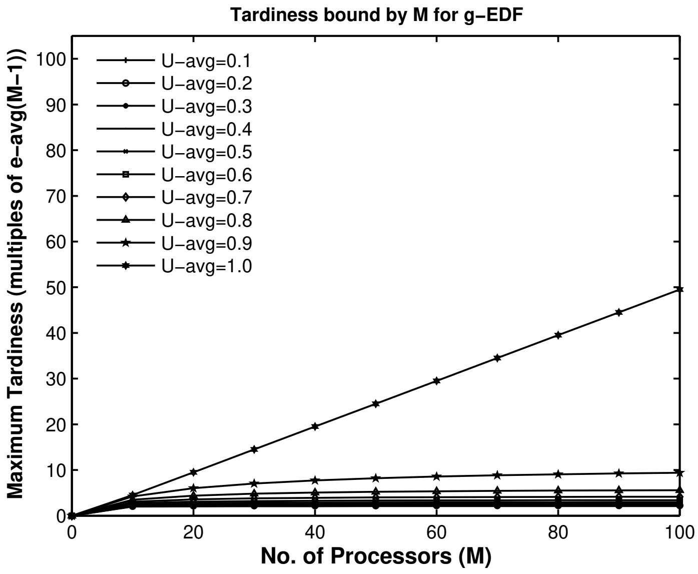
(a)



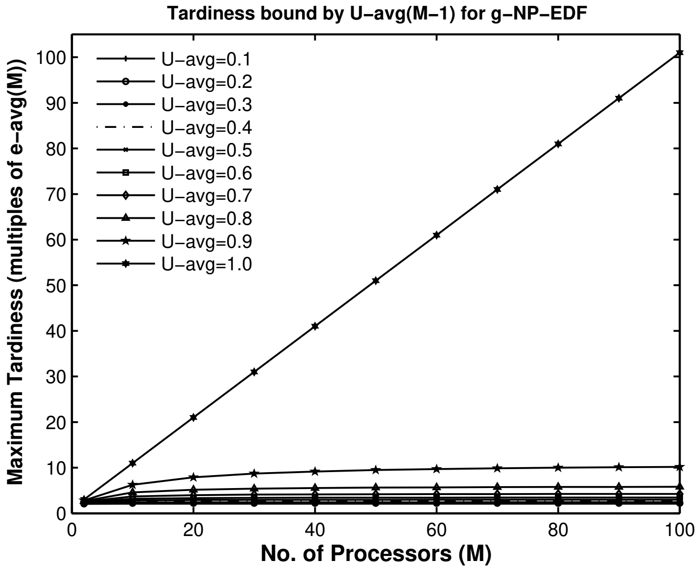
(b)

Figure 5: Tardiness bounds for **(a)** g-EDF and **(b)** g-NP-EDF for varying number of processors (M) as functions of average task utilization (u_{avg}) assuming $U_{sum} = M$. Values of M are higher for curves higher up.

above, tardiness is independent of $u_1 = e_1/p_1$, and hence, holds even when u_1 is arbitrarily small, which is possible by choosing $p_1 \gg e_1$. Note that when $u_{\max}$ is arbitrarily low, a tardiness bound under g-EDF computed using EDF-FAST is around $((M-1) \cdot e_{\max} - e_{\min})/M + e_{\max}$, which is $((M-1) \cdot M - \delta)/M + M$ for this example (because $e_{\max} = M$). This bound differs from the observed tardiness of $M - 1$ by $M - \delta/M$, which is at most $M = e_{\max}$. (The bound computed using EDF-BASIC is $(M + (M-3) \cdot \delta)/(M - u_1 - (M-3) \cdot u_1 \cdot \delta/M) + M$ and differs from the observed tardiness by $(M + (M-3) \cdot \delta)/(M - u_1 - (M-3) \cdot u_1 \cdot \delta/M) + 1$, which can be verified to be at most $(2+\delta)M/(M-1)$ for $u_1 \leq \frac{1}{1+\delta}$.)



(a)



(b)

Figure 6: Tardiness bounds for **(a)** g-EDF and **(b)** g-NP-EDF for varying average task utilizations (u_{avg}) as functions of the number of processors M assuming $U_{sum} = M$. u_{avg} is higher for curves higher up.

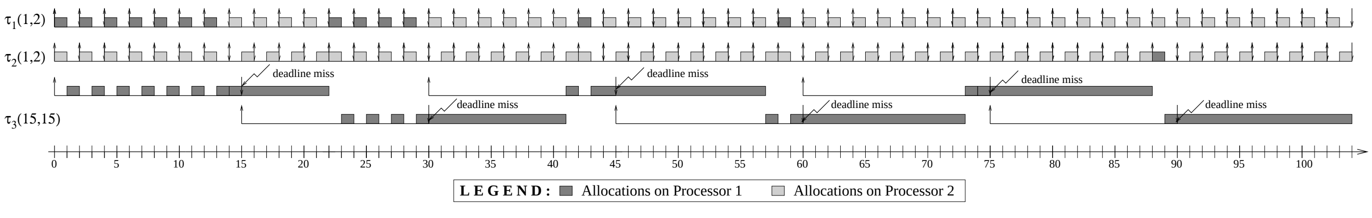


Figure 7: A schedule under g-EDF for the the first few jobs of the task system indicated on two processors. Tardiness for the sixth job of τ_3 is 14 time units, which is $e_{\max} - 1$.

Tightness of the g-EDF bound on two processors. By Theorem 2, tardiness is at most $(e_{\max} + e_\ell)/2$ for task τ_ℓ on two-processor systems. The following is an example that shows that this result is reasonably tight. Let $\tau_1(1,2)$, $\tau_2(1,2)$, and $\tau_3(2k+1, 2k+1)$, where $k \geq 1$, be three periodic tasks all of whose first jobs are released at time zero. If deadline ties are resolved in favor of τ_1 and τ_2 over τ_3, then on two processors, tardiness for jobs of τ_3 can be as high as $2k$ time units. (If τ_3 is favored, then its jobs can miss by $2k-1$.) Such a schedule for a task system with $k = 7$ is shown in Figure 7. In this schedule, the sixth job of τ_3, with deadline at time 90, completes executing at time 104, and hence, has a tardiness of 14 time units, which is $e_{\max} - 1$. It can be verified that tardiness converges to 14 time units after time 104. For this task set, estimated tardiness is $e_{\max} = 2k+1$, and for large k, the difference between estimated and actual tardiness is negligible. The above example can be tweaked to generate other task sets in which the maximum utilization is much less than unity, but which can yet miss their deadlines by $e_{\max} - 1$ time units under g-EDF. A task system with the following seven tasks is one example: $\tau_1(1,4), \ldots, \tau_6(1,4)$, $\tau_7(2k+1, 4k+2)$, where $k \geq 1$. In this example, τ_7 can miss deadlines by up to $2k$ time units, and the remaining tasks by up to k time units. (Note that $2k = e_{\max} - 1$ and $k = \frac{e_{\max}+e_\ell}{2} - 1$, for $1 \leq \ell \leq 6$.)

An empirical evaluation of the accuracy of the bounds is available in Section 5. Interestingly, we have found that tardiness under g-EDF can exceed $e_{\max}$ even for task systems with $u_{\max}$ near 1/2. The following is one such task set: $\tau_1(1,2), \ldots, \tau_4(1,2)$, $\tau_5(1,5), \ldots, \tau_7(1,5)$, $\tau_8(1,11)$, $\tau_9(34,110)$, $\tau_{10}(23,63)$, $\tau_{11}(7,18)$,$\tau_{12}(7,18)$, $\tau_{13}(3,7)$,$\tau_{14}(3,7)$. The total utilization of this task set is five. When scheduled on five processors with deadline ties resolved using task indices, a job of τ_9 misses its deadline at time 7295 by $35 > 34 = e_{\max}$ time units. (The EDF-BASIC and EDF-ITER bounds for this task set are 54 and 51.78, respectively.) Hence, the best bound possible for an arbitrary task set definitely exceeds $e_{\max}$.

4 A Useful Task Model Extension

In this section, we consider a useful extension to the sporadic task model, and discuss how a tardiness bound can be derived under it. We argue that the derivation of such a bound involves only minimal changes to that used for the base model (*i.e.*, the model described in Section 2).

As explained below, the extension that we consider can be used to model certain dynamic task behaviors. In this extended model, the number of tasks associated with the task system is allowed to exceed N and the total utilization of all tasks is allowed to exceed M. (The number of tasks could potentially be infinite.) However, to prevent overload, at any given time, only a subset of tasks whose total utilization is at most M is allowed to be active, *i.e.*, is allowed to release jobs. Additionally, we allow the final job of a task to *halt* at some time t_h before its deadline, provided that the allocation that the job receives in the actual schedule is at most the allocation it receives up to t_h in a PS schedule and the job is not executing in a non-preemptive segment at t_h. When a job halts, its execution cost is altered to equal the amount of time that the job actually executed for in the actual schedule up to t_h. Hence, the halting job receives no allocation in the PS schedule after t_h, even if its deadline is after t_h. At any time t, each task τ_i can be in one of the following states.

- *Active*, if the first job of τ_i is released at or before t, the deadline of its final job is after t, and its final job has not halted at or before t. A task whose final job has its deadline at or before t is not considered active at t even if the final job is pending at t.
- *Inactive*, if the release time of the first job of τ_i is after t.
- *Terminated*, if the deadline of τ_i's final job is before t.
- *Halted*, if the final job has halted but its deadline has not elapsed.

As can be easily seen, a task that is either inactive or terminated at time t cannot have active jobs at t. Also, though the deadline of a halted job may be after its halting time, the job receives no allocation in the PS schedule after it halts.

The set of all tasks associated with the task system is partitioned into N task classes such that the following hold: **(i)** active intervals are disjoint for every two tasks in each class and **(ii)** tasks within a class are governed by precedence constraints, *i.e.*, the first job of a task cannot begin execution until all jobs of all tasks with earlier active intervals in its class have completed execution. The second requirement implies that tasks that are not bound by precedence constraints should belong to different classes even if their active intervals are disjoint.

Examples of task systems that conform to the proposed extension are shown in Figure 8. In each example, tasks that belong to different classes are demarcated using solid lines; those within a class are separated using dashed lines. An initial few jobs are shown for each task. There are eight tasks and six task classes in the task system in inset (a). Each task becomes active at the release time of its first job. Tasks whose jobs are followed by ellipsis are still active at time 18 and may be active beyond that time. Termination times for the remaining tasks are given by the deadlines of their final jobs. For example, τ_2 and τ_3 terminate at time 6, and τ_6 is active in the interval $[12, \infty)$. It can be verified that the total utilization of all the active tasks at any instant is 2. Note that since τ_5 and τ_6 are in the same class, $\tau_{6,1}$ cannot begin execution until both jobs of τ_5 have completed executing. Hence, at time 13, the fourth job of τ_1 is scheduled even though its deadline (which is at time 16) is later than that of $\tau_{6,1}$ (which is at time 15). Inset (b) shows an example wherein a job is halted before its deadline. In this example, at time 14, the fourth job of τ_1, $\tau_{1,4}$, is halted, and a new task, τ_9, which belongs to the same class as τ_1, becomes active. At time 14, $\tau_{1,4}$ receives equal allocations (of one time unit) in both the g-EDF and PS schedules, and hence, halting is permissible.

Applications. The extended task model can readily be used to model tasks whose job execution costs vary but whose utilizations remain unchanged. An example is shown in Figure 9. The task system in this example has two tasks with fixed per-job execution costs and two with variable execution costs. The first variable execution-cost

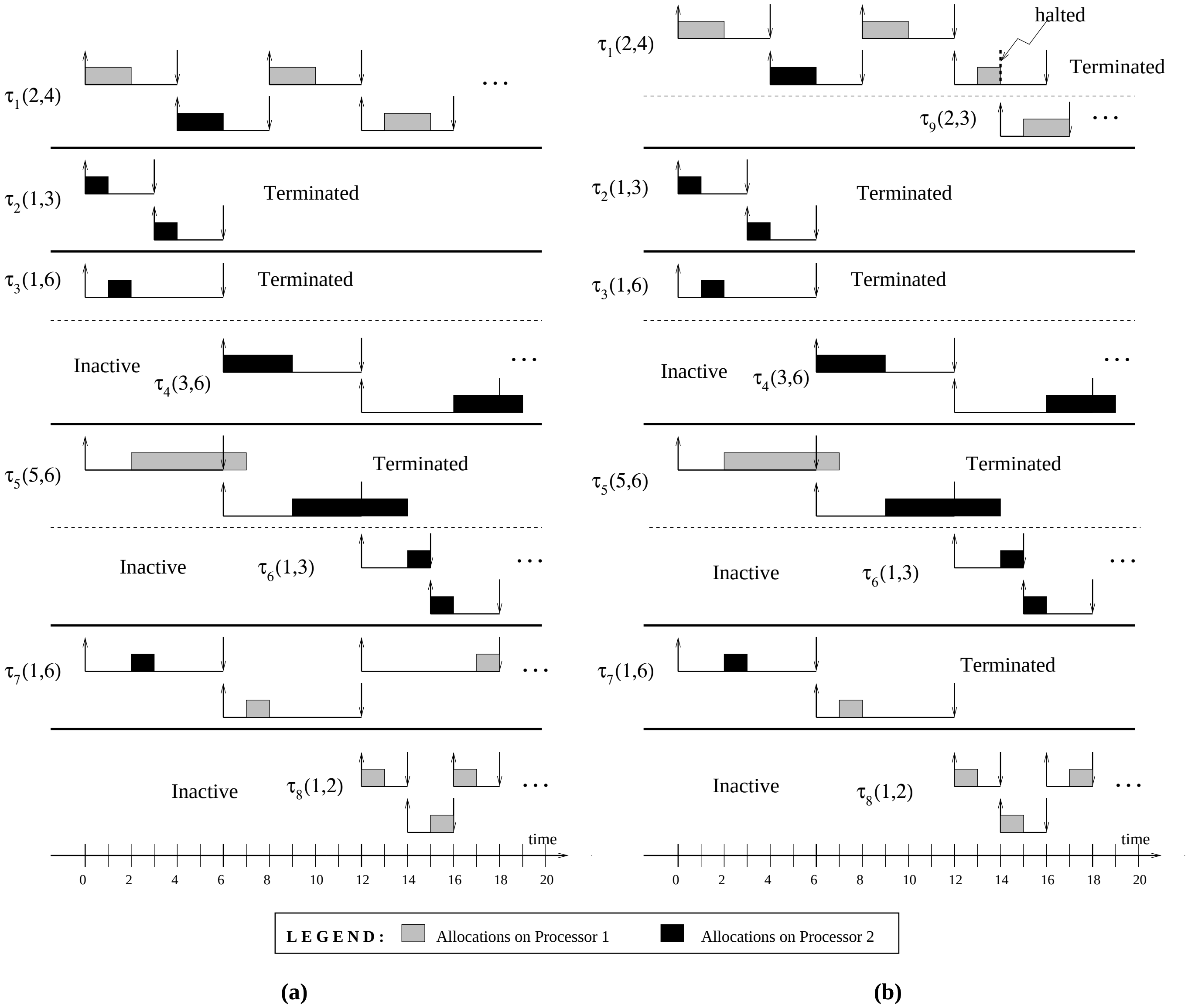


Figure 8: Example g-EDF schedules for extended sporadic task systems. **(a)** No job halts. **(b)** The fourth job of τ_1 halts at time 14.

task has a utilization of 2/3, an execution requirement of six for its first two jobs, and two, for later jobs. The second such task has a utilization of 3/4, an execution requirement of three for its first three jobs, and six, for later jobs. Both the tasks can be represented using task classes of the extended task model. In Figure 9, tasks τ_1 and τ_2 belong to the first task class. τ_1 is active in the interval $[0, 18)$ and τ_2 becomes active at time 18. Similarly, tasks τ_3 and τ_4 together model the second variable execution-cost task.

The extended task model can also be used with dynamic tasks whose utilizations can vary with time. In particular, a single task with a varying utilization can be modeled as a class of potentially infinite tasks, each with a non-changing utilization. Such an approach has been considered for scheduling dynamic task systems under g-EDF and g-NP-EDF in [Block et al., 2006].

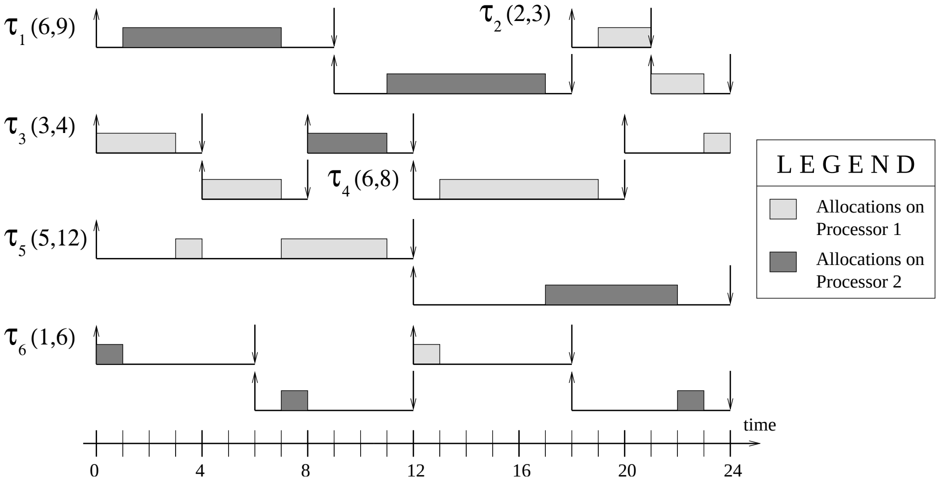


Figure 9: Example application of the extended task model to tasks with variable per-job execution costs.

Tardiness bounds. The approach used for the base model can be used to derive a tardiness bound for the extended model if each task class of the extended model is treated as a distinct task of the base model. For clarity, we will initially assume that no job halts. Letting τ_ℓ^C denote task class ℓ (the superscript C is used to distinguish a task from a task class), in Lemma 5, the lag for τ_ℓ^C can easily be shown to be at most $x \cdot \max_{\tau_i \in \tau_\ell^C} u_i + e_{k,q}$, where $\tau_{k,q}$ is the earliest pending job of any of the tasks in τ_ℓ^C. Note that in a non-blocking, non-busy interval with continually increasing LAG, as considered in Lemma 8, at most Λ task classes can have pending jobs. Letting $u_\ell^C \stackrel{\text{def}}{=} \max_{\tau_i \in \tau_\ell^C} u_i$ and $e_\ell^C \stackrel{\text{def}}{=} \max_{\tau_i \in \tau_\ell^C} e_\ell$, it can be verified that Lemmas 8 and 9 directly apply to the extended model (*i.e.*, hold word for word) if each task class is considered as a task and a C superscript is added to the various terms. (μ_i^C and ϵ_i^C for all $1 \leq \imath \leq N$ are defined for task classes in a straightforward manner). Hence, if at any time the total utilization of all tasks that are active is at most M, then the tardiness bounds derived hold if the maximum utilization and execution cost of each task class are used instead of individual task utilizations and execution costs.

We now discuss why halting is not of issue. Recall that a job can halt only if the allocation to it in the PS schedule is no less than the allocation it receives in the actual schedule. Further, the actual execution cost of the halting job is altered to equal the allocation the job receives in the actual schedule. Hence, the allocation to the job in the PS schedule can also retroactively be altered to equal its actual execution cost. Also, since the job will receive no allocation in the PS schedule after it halts, it is safe to reclaim the utilization of the halting task and activate some other task. That is, there is no risk of overload (*i.e.*, U_{sum} will not exceed M), which can adversely impact other tasks, in the interval spanning the job halting time and job deadline. Finally, by definition, a job can halt only before its deadline and is not pending when it halts. Therefore, because allocations in the PS schedule are altered as described above, the tardiness and lag of a job are zero when the job halts. Hence, lag bounds derived for task classes are not worsened even if its jobs halt. Also, since a job cannot halt in a non-preemptive segment, bounds on blocking times are not altered.

5 Simulation-Based Evaluation

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

The first set of experiments compares the tardiness bounds given by EDF-BASIC and NP-EDF-BASIC and their fast (EDF-FAST and NP-EDF-FAST) and iterative (EDF-ITER and NP-EDF-ITER) variants for 4 and 8 processors (M). For each M, 10^6 task sets were generated. For each task set, new tasks were added as long as U_{sum} was less than M and a final task was added so that U_{sum} equalled M. For each task τ_i, first its utilization u_i was generated as a uniform random number between $(0, y]$, where y was fixed at 0.1 for the first $100,000$ task sets, and was incremented in steps of 0.1 for every $100,000$ task sets. τ_i's execution cost was chosen as a uniform random number in the range $(0, 20]$. For each task set, the maximum tardiness of any task and the average of the maximum tardiness of all tasks as given by the six bounds (three each for g-EDF and g-NP-EDF) were computed. The mean maximum tardiness is plotted for $M = 4$ and $M = 8$ in Figures 10 and 11 as a function of e_{avg} and u_{avg}, respectively, where u_{avg} denotes the average of the $M - 2$ highest utilizations and e_{avg} that of the $M - 1$ highest execution costs. (Mean average tardiness is around $e_{\text{avg}}/2$ time units less.) The descriptions of the plots can be found in the caption of the figure. The rest of this section discusses the results in some detail.

Comparison of BASIC and FAST. Referring to Figures 10(a)and (b) and Figure 11(a), for $M = 4$, the difference between BASIC and FAST is negligible for g-EDF, but is quite considerable for g-NP-EDF at high values of u_{avg} and e_{avg}. This is due to an additional $e_{\max}$ term in the numerator and a negative $u_{\max}$ term in the denominator of NP-EDF-FAST. That is, NP-EDF-FAST has an extra additive term of $e_{\max}$ in the numerator and a subtractive term of $u_{\max}$ in the denominator in comparison to EDF-FAST. On the other hand, the corresponding extra terms for NP-EDF-BASIC in comparison to EDF-BASIC are ϵ_4 and μ_3, which are at most $e_{\max}$ and $u_{\max}$, respectively, and in the task sets generated are likely to be considerably less than the maximum values. Hence, it is likely that NP-EDF-FAST − EDF-FAST > NP-EDF-BASIC − EDF-BASIC implying that NP-EDF-FAST − NP-EDF-BASIC > EDF-FAST − EDF-BASIC. This is further corroborated by the fact that the difference between BASIC and FAST (for both EDF and NP-EDF) is much higher at $M = 8$ for high values of u_{avg}, with the NP-EDF difference almost dwarfing the EDF difference (as ϵ_8 and μ_7 are likely to be considerably lower than ϵ_4 and μ_3). Referring to Figures 10(c) and 11(b). the NP-EDF-FAST bound can be up to twice as much as that of NP-EDF-BASIC. For g-EDF, the difference seems to be tolerable at $M = 8$, but can be expected to be quite significant for higher values of M. Overall, FAST appears to yield good results for small M and small u_{avg} or e_{avg}.

Comparison of BASIC and ITER. The difference between ITER and BASIC is almost the same as that between BASIC and FAST for g-EDF for $M = 4$ (refer to Figures 10(a), 10(b), and 11(a)), but is lower for g-NP-EDF. The difference between the two pairs increases with increasing M for both g-EDF and g-NP-EDF. This is due to the much

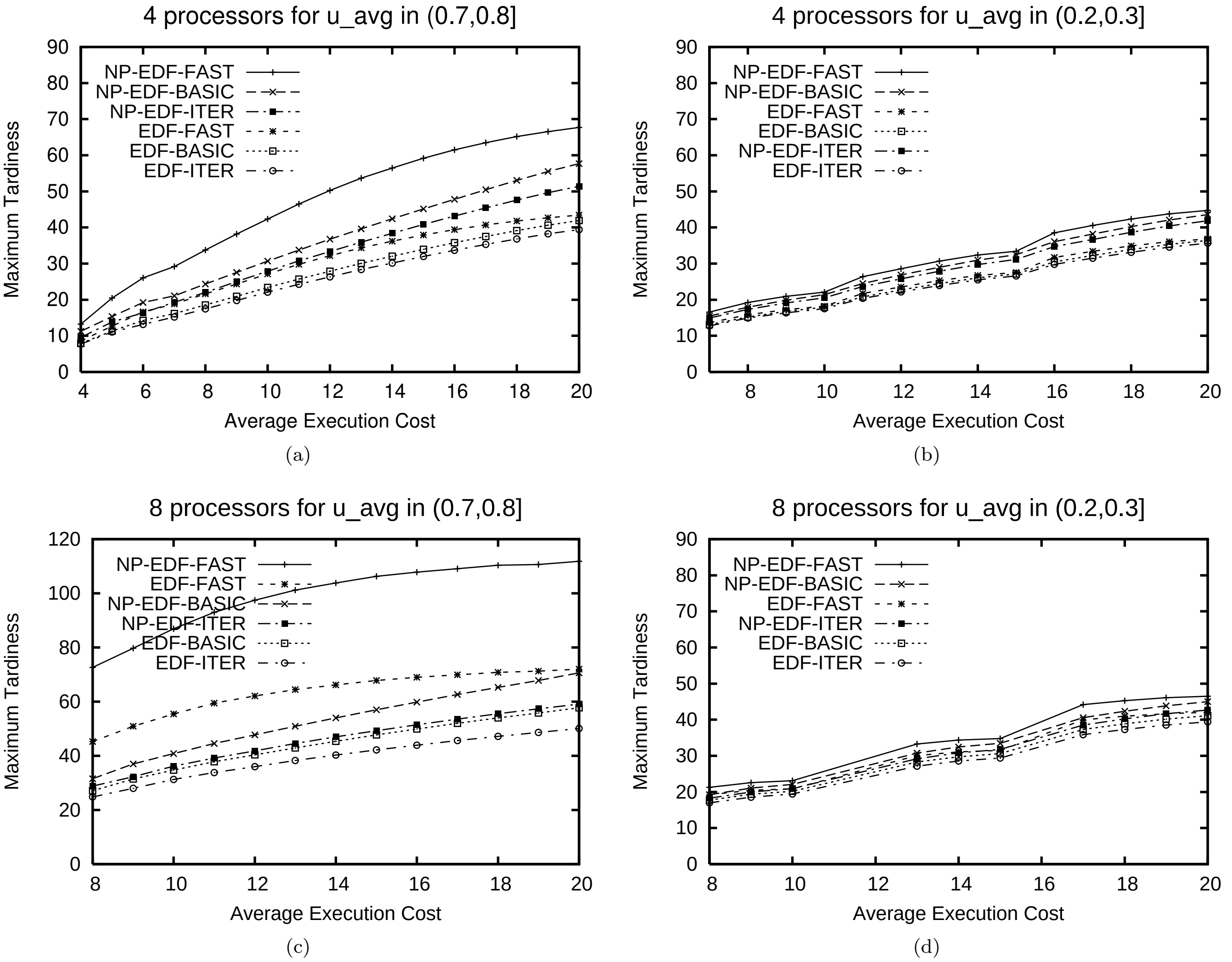


Figure 10: Comparison of the three bounds derived for g-EDF and g-NP-EDF. Tardiness bounds are plotted as functions of e_{avg} for **(a) & (b)** $M = 4$ and **(c) & (d)** $M = 8$ processors. u_{avg} is in $(0.7, 0.8]$ in **(a) & (c)** and is in $(0.2, 0.3]$ in **(b) & (d)**. The order of the legends and curves coincide in all the graphs.

higher rate of increase (with increasing M) of FAST in comparison to BASIC than that of BASIC in comparison to ITER.

Comparison of g-EDF and g-NP-EDF. While there is a large difference between the FAST versions of g-EDF and g-NP-EDF, which widens with increasing M, the difference between EDF-ITER and NP-EDF-ITER is much less and narrows with increasing M. The peak difference between the ITER versions, which is less than 20 time units, occurs for $M = 4$, $0.9 < u_{\text{avg}} \leq 1.0$, and $19 < e_{\text{avg}} \leq 20$ (see Figure 11(a)). At low utilizations, the difference is around five time units for $M = 4$ and three time units for $M = 8$. The difference between the BASIC versions is also not much and decreases with M. However, it should be noted that while these observations hold for the task sets generated here, they cannot be taken to be conclusive. Another point worth mentioning is that, these results assume the same worst-case execution costs for both g-EDF and g-NP-EDF, whereas in practice the estimates for

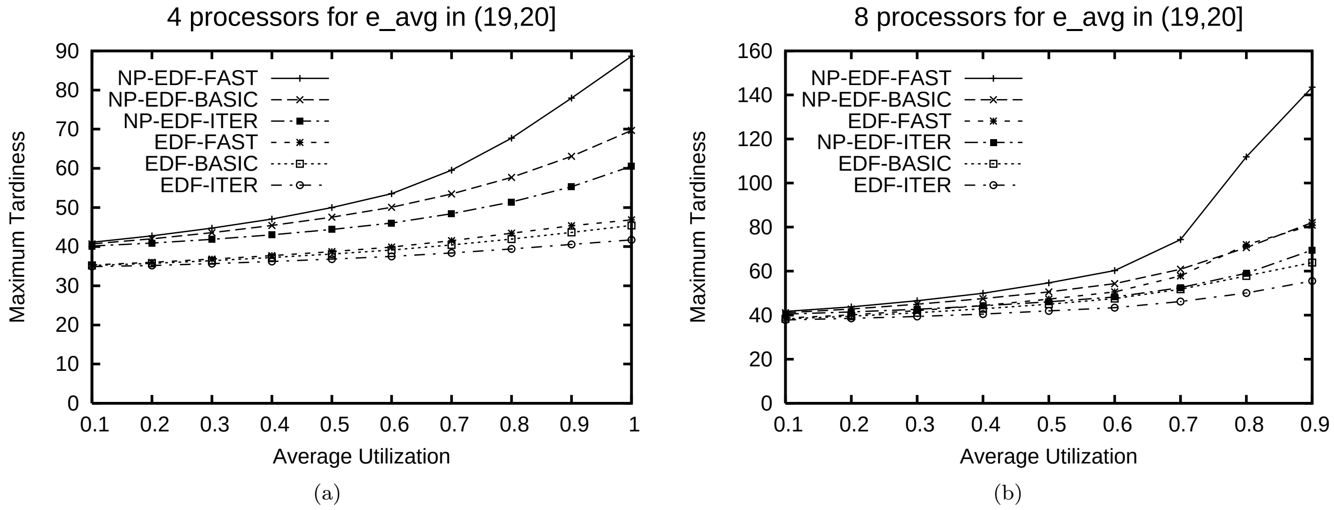


Figure 11: Tardiness bounds as a function of u_{avg} for e_{avg} in $(19, 20]$ on **(a)** $M = 4$ and **(b)** $M = 8$ processors. (The order of the legends and curves coincide in both the graphs.)

g-NP-EDF will be lower due to the absence of job preemptions and migrations. This should further close the gap between the two.

Comparison of ITER to actual tardiness. The experiments in the second set compare the bounds estimated by EDF-ITER and NP-EDF-ITER, the best bounds derived, to actual tardiness observed under g-EDF and g-NP-EDF, respectively. In this case, 100,000 task sets were generated for each M. For each task set, the tardiness bounds given by EDF-ITER and NP-EDF-ITER were computed. Also, a g-EDF and a g-NP-EDF schedule were generated for each task set for 20,000 and 50,000 time units, respectively, and the maximum tardiness observed in each schedule was noted. Plots of the average of the estimated and observed values for task sets grouped by e_{avg} and u_{avg} are shown in Figure 12. For medium values of e_{avg}, the estimates are twice as much as the observed values, with the difference increasing with increasing execution costs. It is somewhat surprising that actual tardiness does not increase much with increasing e_{avg} and that it decreases in some cases. It should also be noted that the difference is higher when M is higher. These plots are a clear indication that there is significant room for improvement. However, tightening the bounds any further does not seem to be easy.

6 Related Work

A primary goal of research on soft real-time systems is providing weaker guarantees on meeting timing constraints, where permissible, in an attempt to improve resource utilization. Most prior work on soft real-time scheduling has focussed on uniprocessor systems only. On uniprocessors, several real-time models that differ in the type of soft real-time guarantees they provide have been proposed in the literature. We will classify the models as either *deterministic* or *probabilistic* based on the predominant nature of the soft real-time guarantees they provide.

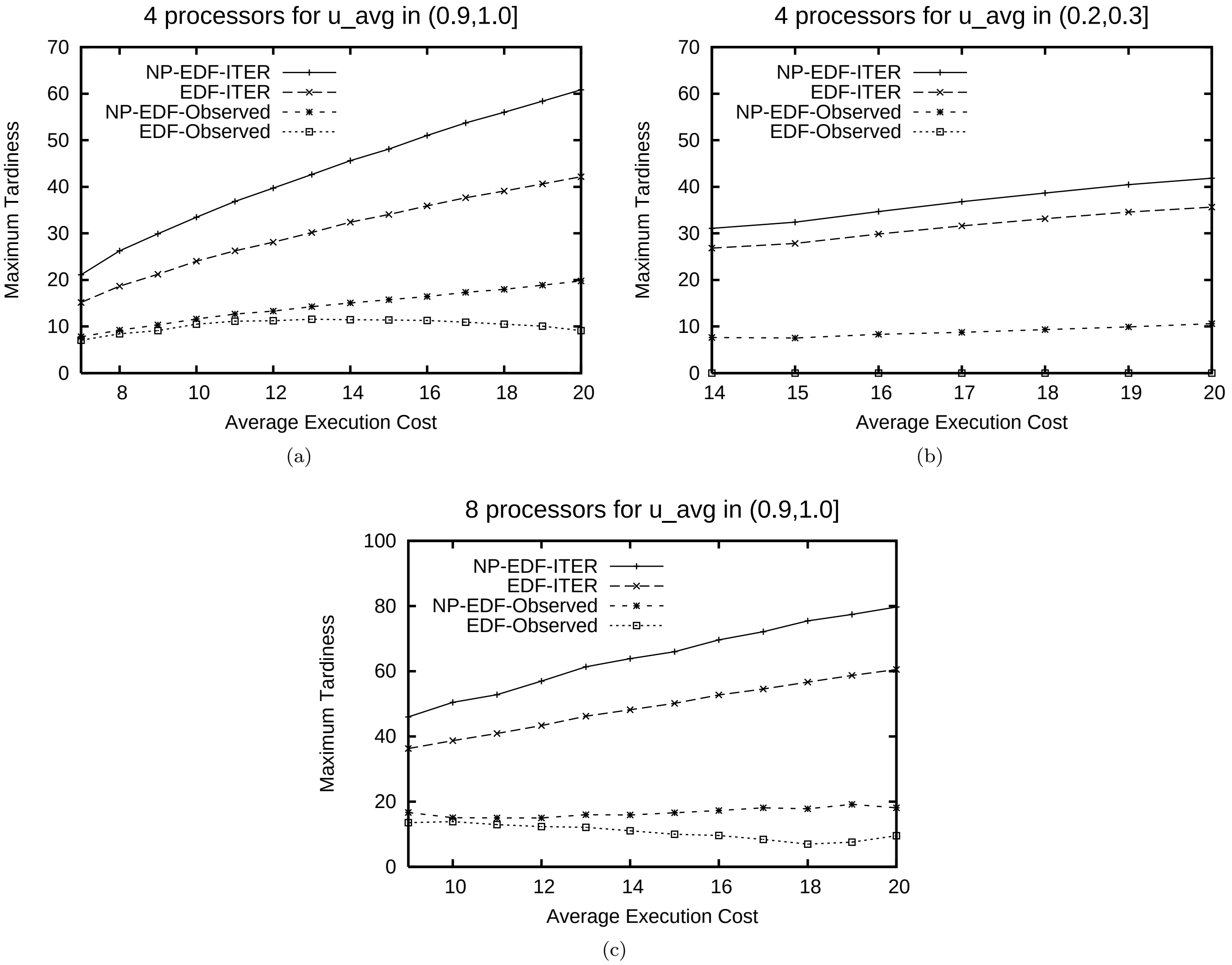


Figure 12: Comparison of EDF-ITER and NP-EDF-ITER to tardiness observed in actual g-EDF and g-NP-EDF schedules for **(a) & (b)** $M = 4$, and for **(c)** $M = 8$.

Deterministic models. Among the early deterministic models is the *skippable periodic task model* of Koren and Shasha [Koren and Shasha, 1995]. In this model, jobs of a task may be *skipped*, *i.e.*, may either miss their deadlines or be aborted at any time, as long as there is a minimum separation between two consecutive skips. Koren and Shasha showed that utilizing skips optimally is NP-hard and proposed algorithms that can exploit skips. Variations on this theme have been proposed by others either independently or as follow-up work. In [Hamdaoui and Ramanathan, 1995], Hamdaoui and Ramanathan introduced the (m, k)-*firm deadline model*, in which at least m jobs in every k consecutive jobs should meet their deadlines, and devised algorithms that can provide probabilistic guarantees on meeting the (m, k) constraint. The *window-constrained model* [West and Poellabauer, 2000] is another similar model, whereas the *weakly-hard real-time model* [Bernat et al., 2001] strives to provide a general framework for the specification of soft real-time constraints. In particular, in the weakly-hard model, a specification for soft real-time constraints (referred to as *weakly-hard constraints* by the authors), which can allow multiple constraints to be associated with each task, and an algebra that relates different constraints, are developed. Schedulability analysis for such task systems under fixed-priority scheduling is also presented. The weakly-hard real-time model also allows jobs that do not meet their deadlines to complete late, but no bound on tardiness is provided.

The models discussed above allow some jobs to be treated as optional and their executions to either complete late or be discarded entirely. In contrast, the imprecise computation model [Liu et al., 1991] is another deterministic model in which some portion of every job can be optional and can be discarded. In this model, each job is composed of a mandatory part and an optional part. The goal is to execute the mandatory part of each job by the job deadline, and to schedule the optional part so that one or more performance metrics such as the number of jobs whose optional parts are discarded or the total amount of time by which all the optional parts are late is minimized. In [Aydin et al., 2001], Aydin *et al.* associate a reward function, which is non-decreasing with the amount of work completed, with the optional parts of the jobs of each periodic task, and show how to schedule imprecise tasks such that the weighted average reward is maximized.

The soft real-time model considered in this paper also provides deterministic guarantees. This model, where sporadic tasks with implicit deadlines have a tardiness threshold, has not been considered much in the context of uniprocessors. This is because, if the system is not overloaded, then it is obvious that scheduling under EDF is sufficient to ensure that each job completes execution by its deadline, and that there is no scope of allowing a higher utilization even if bounded tardiness is tolerable. This model can alternatively be viewed as one in which timing constraints are hard, but the relative deadlines of the sporadic tasks are *larger* than their periods.¶ When viewed in this alternative manner, the model has been the focus of some research in the context of *deadline-monotonic* scheduling on uniprocessors. In [Lehoczky, 1990], Lehoczky provides a demand-based test, which may require exponential time, to determine the schedulability of task systems that use the model, and also derives utilization bounds when the relative deadline of each task is a multiple (greater than one) of its period. However, determining the maximum time after the end of its period that any job may complete executing has not been addressed.

Probabilistic models. For some tasks, such an those in a video-conferencing application or an animation game, the inter-arrival time between two consecutive jobs or the execution requirements of different jobs can vary widely over time. For such systems, reserving resources based on worst-case parameters (shortest inter-arrival time and longest execution time) can be extremely wasteful. One way of improving resource utilization for such systems is to model task parameters probabilistically (using probability distributions for inter-arrival times or execution times) and provide probabilistic guarantees on meeting deadlines. Examples of research in this direction can be found in [Tia et al., 1995, Atlas and Bestavros, 1998, Abeni and Buttazzo, 1998, 1999, Diaz et al., 2002]. Within the probabilistic domain, Lehoczky's *real-time queueing theory* [Lehoczky, 1996] strives to provide a framework

¶It should be noted that if the relative deadlines of all tasks are not increased by equal amounts, then it is possible for a task τ_i with a shorter period than another task τ_j to have a larger relative deadline than τ_j, or vice versa. Hence, when viewed in this alternative manner, task priorities determined based on relative deadlines or job priorities based on absolute deadlines may not match those determined under our model in which relative deadlines equal periods.

for combining the timing constraints of real-time scheduling with the stochastic elements of queueing theory. This theory may be used to determine the fraction of jobs missing their deadlines in either single-queue (*i.e.*, single task), single-server systems [Doytchinov et al., 2001], or acyclic networks of servers with multiple queues [Kurk et al., 2004] under heavy traffic conditions‖.

Almost all of the work on the models described above is concerned with dealing with overload on uniprocessors, *i.e.*, scenarios in which the total system utilization exceeds the available capacity, which is 1.0. (In general, deterministic models are used when the overload is over longer terms, such as system lifetimes. Probabilistic models are more suited for systems that may be overloaded occasionally for short durations but whose average loads do not exceed the available capacity. Combinations of both are also possible.) However, in the context of multiprocessors, there exist scheduling algorithms under which guarantees on timeliness that can be provided are not known when the total utilization exceeds the schedulable utilization of the algorithm. The worst-case schedulable utilization of the concerned algorithms is $(M+1)/2$. Hence, possible guarantees on timeliness are not known even when the system is much below full load. This is true of most of the known algorithms except the optimal Pfair algorithms, necessitating a study of the model of this paper. We conjecture that results based on this model may, in fact, be necessary in dealing with overload.

On multiprocessors, soft Pfair-based real-time scheduling has previously been considered in [Srinivasan and Anderson, 2003] and [Devi and Anderson, 2004], where tardiness bounds are derived for a suboptimal Pfair scheduling algorithm that is less expensive than optimal algorithms. An algorithm with limited inter-processor migrations, based on partitioned-EDF, for scheduling soft real-time systems has been presented in [Anderson et al., 2005].

7 Conclusion

We have derived tardiness bounds under preemptive and non-preemptive global EDF for sporadic real-time task systems on multiprocessors, when the total utilization of a task system is not restricted, but may equal the number of processors, M. Our tardiness bounds depend on the total system utilization, and per-task utilizations and execution costs — the lower these values, the lower the tardiness bounds. Though the bounds are not believed to be tight, they are reasonable and should be acceptable as long as the maximum task utilization is not very high, say above 0.8. These results should help in improving the effective system utilization when soft real-time tasks that can tolerate bounded deadline misses but require guarantees on the long-run fraction of the processing time allocated are multiplexed on a multiprocessor. Improving the accuracy of the bounds remains a challenging open problem.

Our task model can alternatively be viewed as one in which the relative deadline of each task is greater than its period by an amount that is the same for all tasks and is dependent on the parameters of the task system.

‖Under heavy traffic conditions, the traffic intensity or the average utilization of the processor converges to one.

Our conditions that check if a tardiness bound can be guaranteed can then be used as schedulability tests for such task systems. While it is possible to extend the EDF schedulability tests derived in prior research discussed in the introduction to task systems with relative deadlines greater than periods, it does not seem likely that such extensions would allow total utilization to equal M even when per-task utilizations are low and relative deadlines are large.

Apart from not being tight, another limitation of our result is that the tardiness bound that can be guaranteed to each task is fixed and is dependent on task parameters. Guaranteeing arbitrary and different tardiness bounds to different tasks is another challenging problem towards which we have taken some steps in later work [Devi and Anderson, 2006].

References

L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings of the 19th IEEE Real-Time Systems Symposium*, pages 4–13, December 1998.

L. Abeni and G. Buttazzo. QoS guarantees using probabilistic deadlines. In *Proceedings of the 11th Euromicro Conference of Real-Time Systems*, pages 242–249, June 1999.

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

J. Anderson, V. Bud, and U. Devi. An EDF-based scheduling algorithm for multiprocessor soft real-time systems. In *Proceedings of the 17th Euromicro Conference on Real-Time Systems*, pages 199–208, July 2005.

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

A. Atlas and A. Bestavros. Statistical rate monotonic scheduling. In *Proceedings of the 19th IEEE Real-Time Systems Symposium*, pages 123–132, December 1998.

H. Aydin, R. Melhem, D. Mosse, and P. M. Alvarez. Optimal reward-based scheduling for periodic real-time tasks. *IEEE Transactions on Computers*, 50(2):111–130, February 2001.

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

S. Baruah. The non-preemptive scheduling of periodic tasks upon multiprocessors. *Real-Time Systems*, 32(1-2): 9–20, February 2006.

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

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

G. Bernat, A. Burns, and A. Liamosi. Weakly hard real-time systems. *IEEE Transactions on Computers*, 50(4): 308–321, April 2001.

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

M. Bertogna, M. Cirinei, and G. Lipari. New schedulability tests for real-time task sets scheduled by deadline monotonic on multiprocessors. In *Proceedings of the 9th International Conference on Principles of Distributed Systems*, December 2005b.

A. Block, J. Anderson, and U. Devi. Task reweighting under global scheduling on multiprocessors. In *Proceedings of the 18th Euromicro Conference on Real-Time Systems*, pages 128–138, July 2006.

M. Blum, R. Floyd, V. Pratt, R. Rivest, and R. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7(4):448–461, August 1973.

U. Devi. *Soft Real-Time Scheduling on Multiprocessors*. PhD thesis, University of North Carolina at Chapel Hill, December 2006.

U. Devi and J. Anderson. Improved conditions for bounded tardiness under EPDF fair multiprocessor scheduling. In *Proceedings of the 12th International Workshop on Parallel and Distributed Real-Time Systems*, April 2004. 8 pages (On CD-ROM).

U. Devi and J. Anderson. Tardiness bounds under global EDF scheduling on a multiprocessor. In *Proceedings of the 26th IEEE Real-Time Systems Symposium*, pages 330–341, December 2005.

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

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

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

J. L. Diaz, D. F. Garcia, K. Kim, C.-G. Lee, L. Bello, J. M. Lopez, S. L. Min, and O. Mirabella. Stochastic analysis of periodic real-time systems. In *Proceedings of the 23rd IEEE Real-Time Systems Symposium*, pages 289–300, December 2002.

B. Doytchinov, J. Lehoczky, and S. Shreve. Real-time queues in heavy traffic with Earliest-Deadline-First queue discipline. *Annals of Applied Probability*, 11:332–378, 2001.

R. Floyd and R. Rivest. Expected time bounds for selection. *Communications of the ACM*, 18(3):165–172, 1975.

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

M. Hamdaoui and P. Ramanathan. A dynamic priority assignment technique for streams with (m, k)-firm deadlines. *IEEE Transactions on Computers*, 44(12):1443–1451, December 1995.

C.A.R. Hoare. Algorithm 63 (PARTITION) and algorithm 65 (FIND). *Communications of the ACM*, 4(7):321–322, 1961.

K. Jeffay and S. Goddard. A theory of rate-based execution. In *Proceedings of the Real-Time Systems Symposium*, pages 304–314, Phoenix, AZ, December 1999. IEEE Computer Society Press.

G. Koren and D. Shasha. Skip-over: Algorithms and complexity for overloaded systems that allow skips. In *Proceedings of the 16th IEEE Real-Time Systems Symposium*, pages 110–117. IEEE, December 1995.

L. Kurk, J. Lehoczky, S. Shreve, and S.-N. Yeung. Earliest-Deadline-First service in heavy-traffic acyclic networks. *Annals of Applied Probability*, 14(3):1306–1352, 2004.

J. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Proceedings of the 11st IEEE Real-Time Systems Symposium*, pages 201–209. IEEE, December 1990.

J. P. Lehoczky. Real-time queueing theory. In *Proceedings of the 17th IEEE Real-Time Systems Symposium*, pages 186–195, December 1996.

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

J. W. S. Liu, K.-J. Lin, W.-K.Shih, and A. C. Yu. Algorithms for scheduling imprecise computations. *IEEE Computer*, 24(5):58–68, 1991.

A. Mok. *Fundamental Design Problems of Distributed Systems for Hard Real-Time Environments*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Mass., 1983.

A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The single node case. *IEEE/ACM Transactions on Networking*, 1(3):344–357, 1993.

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

A. Srinivasan and J. Anderson. Optimal rate-based scheduling on multiprocessors. In *Proceedings of the 34th ACM Symposium on Theory of Computing*, pages 189–198, May 2002.

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

A. Srinivasan and J. Anderson. Fair scheduling of dynamic task systems on multiprocessors. *Journal of Systems and Software*, 77(1):67–80, April 2005.

A. Srinivasan and J. Anderson. Optimal rate-based scheduling on multiprocessors. *Journal of Computer and System Sciences*, 72(6):1094–1117, September 2006.

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

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

T.-S. Tia, D.-Z. Deng, M. Shankar, M. Storch, J. Sun, L.-C. Wu, and J.-S. Liu. Probabilistic performance guarantee for real-time tasks with varying computation times. In *Proceedings of the 2nd IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 164–173, May 1995.

P. Valente and G. Lipari. An upper bound to the lateness of soft real-time tasks scheduled by EDF on multiprocessors. In *Proceedings of the 26th IEEE Real-Time Systems Symposium*, pages 311–320, December 2005.

R. West and C. Poellabauer. Analysis of a window-constrained scheduler for real-time and best-effort packet streams. In *Proceedings of the 21st IEEE Real-Time Systems Symposium*, pages 239–248. IEEE, December 2000.

A Remaining Proofs

In this appendix, we provide proofs omitted in the main sections.

A.1 Proof of Lemma 4

We first prove Lemma 4. (Lemma 4 is necessary to establish the bound as given in (36) for g-EDF. Otherwise, only a bound of $\frac{\sum_{i=1}^{\Lambda} \epsilon_i - e_{\min}}{M - \sum_{i=1}^{\Lambda} \mu_i} + e_k$ can be established. The difference is in the second term of the denominator. Also, without Lemma 4, the bound derived for two processors would not be close to being tight.)

Displacements. In proving Lemma 4, we consider removing one or more jobs from τ. Note that the resulting task system (comprised of the remaining jobs) is still a valid concrete instantiation of τ^N. (We can simply assume either that jobs that are of the same task as any removed job and are released after that job are appropriately renumbered or that the actual execution costs of removed jobs are reduced to zero.) However, such job removals can cause one or more of the remaining jobs or job fragments to shift earlier in $\mathcal{S}$. A *job fragment* is defined with respect to a schedule, and is a portion of a job, which could potentially be a complete job, that executes continuously without preemption in some schedule under consideration. As described below, such shifts due to job removals can be reduced to a set of zero or more *disjoint* displacement chains comprised of job fragments of equal lengths. (If a job removal does not result in any change in the schedule for the remaining jobs, then the number of displacement chains is zero.) Each *displacement chain* Δ_i is denoted $(\Delta_{i,1}, \Delta_{i,2}, \ldots, \Delta_{i,n_i})$ and is a sequence of n_i equal-length and disjoint displacements. L denotes the length of every job fragment in every displacement. Each *displacement* $\Delta_{i,j}$ is a 4-tuple denoted $\langle J^{(i,j)}, t_{i,j}, J^{(i,j+1)}, t_{i,j+1} \rangle$, with the meaning that job fragment $J^{(i,j+1)}$ scheduled at $t_{i,j+1}$ displaces fragment $J^{(i,j)}$ scheduled at $t_{i,j}$. Here $J^{(i,j)}$ is the *displaced* job fragment and $J^{(i,j+1)}$, the *displacing* fragment. The displacements are disjoint in the sense that fragments $J^{(i,j)}$ and $J^{(i,j+1)}$ are disjoint. However, the time intervals in which the fragments are scheduled may overlap. Similarly, the displacement chains are disjoint in that no two chains can share a common job fragment or fragments that are overlapping.

Informally, displacement chain Δ_i denotes, in sequence, job fragments whose schedules are altered due to the removal of the fragment at its head, namely, $J^{(i,1)}$. Hence, it is required that the displacing and displaced job fragments (*i.e.*, the second and first components) of two consecutive displacements in any chain be the same. Though it is not immediately obvious, it can be seen that by partitioning each removed job into as many fragments as necessary, all the shifts resulting from job removals can be reduced to a series of zero or more disjoint displacement chains of equal-length displacements. We will also make a reasonable assumption that the number of chains per removal is finite. Two examples are provided in Figure 13. In the second example, the removed job J is partitioned into five fragments. Each fragment except the last is the head of a displacement chain of job fragments, which are linked.

Finally, a note on notation: a J with a two-tuple superscript as in $J^{(i,j)}$ denotes a fragment of an arbitrary job, and not a fragment of job J.

A few auxiliary lemmas are needed in establishing Lemma 4, but first we state the following simple claim. The claim follows because if jobs all fully preemptable, then jobs in Ψ are not impacted by those in $\overline{\Psi}$. Hence, a

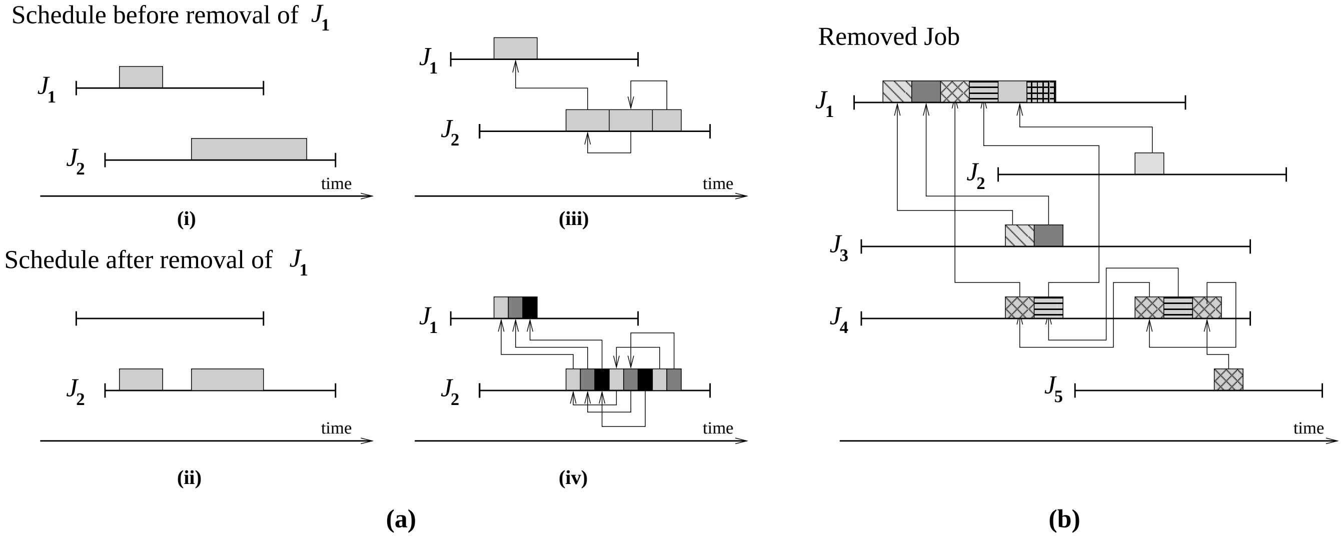


Figure 13: Displacement chains resulting due to the removal of job J_1. **(a)** An example with simple shifts. **(a)-(ii)** Schedule for J_2 after the removal of J_1. **(a)-(iii)** A straightforward displacement chain, whose displacements are not of equal length. **(a)-(iv)** Decomposition of the displacement chain in **(a)-(iii)** into three chains of disjoint and equal-length displacements. **(b)** An example with complex shifts. In **(a)-(iv)** and **(b)**, job fragments that are parts of the same displacement chain are shaded identically and are linked using arrows. Arrows point from the displacing to the displaced fragment of each displacement.

non-empty $\overline{\Psi}$ would contradict the assumption in (P2).

Claim 4 *If* $b_{\max} = 0$ *then* $\overline{\Psi} = \emptyset$.

In the lemmas that follow, $b_{\max} = 0$ holds. Hence, EDF-P-NP reduces to g-EDF. To simplify description, in the proofs, we will assume that the scheduler is g-EDF.

The lemma below concerns displacements and is quite intuitive. It follows from the work-conserving nature of g-EDF and our assumption that deadline ties are resolved consistently.

Lemma 11 *If* $b_{\max} = 0$*, then for every job removed from* τ*, the resulting sequence of shifts in* $\mathcal{S}$ *can be reduced to a set of zero or more displacement chains such that for each displacement* $\Delta_{i,j}$ *in each chain* Δ_i*,* $t_{i,j+1} > t_{i,j}$*, i.e., every displacement is to the left.*

We need one more lemma before proving Lemma 4.

Lemma 12 *Let* $[t, t')$*, where* $t < t' \leq t_d$*, be a non-blocking, non-busy interval across which* LAG *for* Ψ *is continually increasing. If* $b_{\max} = 0$*, then there exists at least one job* J *such that* J*'s deadline is at or after* t' *and* J *completes executing at or before* t.

Proof: Contrary to the statement of the lemma, assume that there does not exist a job J as defined. First, we establish a part of the lemma in the following claim.

Claim 5 *There exists at least one job J with deadline at or after t' that completes executing before t'.*

Proof: Suppose J does not exist. Let $\hat{t}$ denote the latest of time t and the latest deadline of any job that completes executing before t'. By our assumption that a job like J does not exist, $\hat{t} < t'$ holds. Because $[\hat{t}, t')$ is a non-blocking, non-busy interval, there does not exist any instant in $[\hat{t}, t')$ at which a ready job from Ψ is waiting. Furthermore, because J does not exist, and by our choice of $\hat{t}$, there is no instant in $[\hat{t}, t')$ at which there is an active job in Ψ that completed earlier, and hence, is not executing. Thus, there does not exist any instant in $[\hat{t}, t')$ where a task with an active or a pending job in Ψ is not executing. Hence, the total allocation at any instant u in the interval $[\hat{t}, t')$ to jobs in Ψ in $\mathcal{S}$ is $|\{\tau_i | \tau_{i,j} \text{is in } \Psi \text{ and is active or pending at } u\}|$, which, by (16) and (14), is at least that those jobs receive in PS_τ. Therefore, by (17), LAG of Ψ at t' cannot exceed that at $\hat{t}$. This contradicts the fact that LAG for Ψ is continually increasing in $[t, t')$. ■

Note that J, as specified in the above claim, is not $\tau_{\ell,j}$. Otherwise, since J completes executing before its deadline, its tardiness is zero, which contradicts (P1). Next, we show that J completes executing at or before t. (J necessarily executes at least partially before t. In other words, the actual execution cost is greater than zero. Otherwise, J can be removed without impacting the schedule for the remaining jobs and, hence, the tardiness for $\tau_{\ell,j}$, contradicting (P2).) Assume to the contrary, and consider a concrete instantiation τ' of τ^N obtained from τ by lowering the actual execution time of J such that J does not have to execute after t. Let $\mathcal{S}'$ be a schedule for τ' such that all ties among jobs are resolved the same way as in $\mathcal{S}$. Then in $\mathcal{S}'$, every job in τ' except J is scheduled exactly in the same intervals as in $\mathcal{S}$. This is because no job of Ψ is ready but waiting at any instant in $[t, t')$ in $\mathcal{S}$, and hence, the fact that another processor becomes available in $[t, t')$ due to J not executing in that interval would not cause additional jobs from Ψ to execute in $[t, t')$. Since $b_{\max} = 0$, by Claim 4, $\overline{\Psi} = \emptyset$. Hence, no change to the schedule for jobs in Ψ due to the displacement of jobs in $\overline{\Psi}$ is possible. Thus, the completion time of every job in Ψ except J will be the same in both $\mathcal{S}$ and $\mathcal{S}'$. This contradicts (P3), since as explained above, J is not $\tau_{\ell,j}$. ■

We are finally ready to prove Lemma 4.

Lemma 4 *Let $[t, t')$, where $t < t' \leq t_d$, be a non-blocking, non-busy interval across which* LAG *for Ψ is continually increasing. If $b_{\max} = 0$, then there exists at least one job J and some time $\hat{t}$ such that $d(J) \geq t'$, $t \leq \hat{t} < t'$, and J executes continuously in $[\hat{t}, t')$.*

Proof: By Lemma 12, there exists a job J' such that $d(J') \geq t'$ and J' completes executing by t. Because J' completes executing before its deadline, J' is not $\tau_{\ell,j}$. Otherwise, (P1) is contradicted. Our proof is by contradiction. Hence, assume (A) below holds.

(A) J as described in the statement of the lemma does not exist.

We show that if (A) holds, then (P2) does not hold, and thereby derive a contradiction.

Consider a concrete instantiation τ' of τ^N obtained from τ by removing J'. (As explained above, J' is not $\tau_{\ell,j}$. Also, the actual execution cost of J' is greater than zero. Otherwise, the removal of J' will impact neither the schedule for the remaining jobs nor the tardiness of $\tau_{\ell,j}$, contradicting (P2).) Let $\mathcal{S}'$ be a g-EDF schedule for τ' such that ties among jobs are resolved identically in $\mathcal{S}$ and $\mathcal{S}'$. To show that (P2) is contradicted, we show that if (A) holds, then the removal of J' does not result in any job fragment scheduled at or after t' in $\mathcal{S}$ (either partially or fully, *i.e.*, regardless of whether the fragment commenced execution before t' or later) to shift left to before t' in $\mathcal{S}'$. This would imply that no job fragments executing at or after t', and in particular those of $\tau_{\ell,j}$, shift earlier, and hence all jobs scheduled at or after t' have the same tardiness in both $\mathcal{S}$ and $\mathcal{S}'$.

Let the removal of J' result in a sequence of left shifts for the remaining jobs and let these shifts be reduced to a set of equal-length, disjoint displacement chains. Our objective is to show that if some job fragment executing at or after t' gets displaced to the left, then (A) is contradicted. Hence, assume that some job fragment scheduled at or after t' undergoes a left shift. Because there is at least one displacement, the number of displacement chains is at least one, and by Lemma 11, we have the following.

(D1) For every displacement $\Delta_{i,j}$ in every chain Δ_i, $t_{i,j+1} > t_{i,j}$.

Also, the fragment at the head of each displacement chain is that of job J', whose deadline is at or after t'. Therefore, by (D1), g-EDF prioritizes J' over the job of a fragment later in the chain. Hence, we have the following.

(D2) The deadline of the job of every fragment involved in every displacement is at least t'.

Let $\Delta_{i,k} = \langle J^{(i,k)}, t_{i,k}, J^{(i,k+1)}, t_{i,k+1}\rangle$ be a displacement such that $t_{i,k+1} + L > t'$, where L is the length of every fragment in the displacement chains under consideration, and $t_{i,k} < t'$. In other words, the displacement shifts the job fragment $J^{(i,k+1)}$ either partially or fully to the left of t'. (By (D1), $t_{i,k} < t_{i,k+1}$ holds.) If $t_{i,k+1} < t'$, then because $t_{i,k+1} + L > t'$, in $\mathcal{S}$, $J^{(i,k+1)}$ executes continuously in $[t_{i,k+1}, t')$, contradicting (A). On the other hand, if $t_{i,k+1} \geq t'$, then we argue as follows. Because $J^{(i,k+1)}$ is not executing before t' in $\mathcal{S}$ (implied by $t_{i,k+1} \geq t'$), the non-blocking, non-busy nature of $[t, t')$ and the fact that $J^{(i,k+1)}$ is executing before t' in $\mathcal{S}'$ (implied by the displacement $\Delta_{i,k}$ and $t_{i,k} < t'$) imply that $J^{(i,k+1)}$ is not ready at t'^{-} in $\mathcal{S}$ because a prior job fragment, say, J^p, of the same task is executing there in $\mathcal{S}$. It also follows that $J^{(i,k+1)}$ is ready in $\mathcal{S}'$ before t' because J^p is displaced. By (D2), this implies that the deadline of the job of J^p is at or after t'. But then, because J^p is executing at t'^{-}, (A) is contradicted. Hence, if (A) holds, displacement $\Delta_{i,k}$ is not possible. That is, even if J' is removed, no job fragment can shift to the left of t', and so, the tardiness for every job completing execution after t' in $\mathcal{S}$, and in particular, for $\tau_{j,\ell}$, is the same in $\mathcal{S}'$, as well, which contradicts (P2). ■

A.2 Proofs of Lemmas 6 and 7

Lemma 6 *The following properties hold for sets* $\Gamma(k+c,\ell)$ *and* $\Pi(k+c,\ell)$ *with* $0 \le \ell \le k \le N$ *and* $0 \le c \le N-k$*, where* Γ *and* Π *are as defined in* (3) *and* (4)*, respectively.*

- **(i)** $\sum_{\tau_i \in \Gamma^{(k+c,\ell)}} e_i + \sum_{\tau_i \in \Pi^{(k+c,\ell)}} b_i \le \sum_{\tau_i \in \Gamma^{(k,\ell)}} e_i + \sum_{\tau_i \in \pi^{(k,\ell)}} b_i + \sum_{i=1}^{c} \beta_i$.
- **(ii)** $\sum_{\tau_i \in \Gamma^{(k+c,\ell)}} e_i + \sum_{\tau_i \in \Pi^{(k+c,\ell)}} b_i \ge \sum_{\tau_i \in \Gamma^{(k,\ell)}} e_i + \sum_{\tau_i \in \Pi^{(k,\ell)}} b_i$.

Proof: Let τ' denote the set of all tasks with ranks from $k+1$ to $k+c$ in a non-increasing ordering of the tasks by execution costs. Then,

$$|\tau'| = c \tag{38}$$

and $\Gamma^{(k+c,\ell)} \cup \Pi^{(k+c,\ell)} = \Gamma^{(k,\ell)} \cup \Pi^{(k,\ell)} \cup \tau'$ hold. Furthermore, $|\Gamma^{(k+c,\ell)}| = |\Gamma^{(k,\ell)}|$, $|\Pi^{(k+c,\ell)}| = |\Pi^{(k,\ell)}| + c$, and $\Pi^{(k,\ell)} \subseteq \Pi^{(k+c,\ell)}$ hold. (The final subset containment can be seen to hold as follows: if $\Gamma^{(k+1,\ell)} = \Gamma^{(k,\ell)}$, then $\Pi^{(k+1,\ell)}$ includes the task with rank $k+1$ in addition to every task in $\Pi^{(k,\ell)}$; on the other hand, if the task with rank $k+1$ is included in $\Gamma^{(k+1,\ell)}$, then some task from $\Gamma^{(k,\ell)}$ is added to $\Pi^{(k,\ell)}$ to form $\Pi^{(k+1,\ell)}$.) Define

$$A \subseteq \tau' \quad \wedge \quad B \subseteq \Gamma^{(k,\ell)} \tag{39}$$

such that

$$\Gamma^{(k+c,\ell)} = (\Gamma^{(k,\ell)} \setminus B) \cup A \tag{40}$$

holds. Note that, by (39), $A \cap B = \emptyset$, and by (39), (40), and (4), the following hold.

$$|A| = |B| \tag{41}$$

$$\Pi^{(k+c,\ell)} = \Pi^{(k,\ell)} \cup (\tau' \setminus A) \cup B \tag{42}$$

Let $C = \tau' \setminus A$. Then,

$$\begin{aligned}
&\sum_{\tau_i \in \Gamma^{(k+c,\ell)}} e_i + \sum_{\tau_i \in \Pi^{(k+c,\ell)}} b_i \\
&= \sum_{\tau_i \in \Gamma^{(k,\ell)}} e_i + \sum_{\tau_i \in A} e_i - \sum_{\tau_i \in B} e_i + \sum_{\tau_i \in \Pi^{(k,\ell)}} b_i + \sum_{\tau_i \in C} b_i + \sum_{\tau_i \in B} b_i && \text{(by (39), (40), and (42))} \\
&\le \sum_{\tau_i \in \Gamma^{(k,\ell)}} e_i + \sum_{\tau_i \in \Pi^{(k,\ell)}} b_i + \sum_{\tau_i \in C} b_i + \sum_{\tau_i \in B} b_i && \text{(by (39) and the definitions of } \tau' \text{ and } \Gamma^{(k,\ell)}\text{, the execution costs of tasks in } A \text{ are at most those of tasks in } B\text{, and by (41), } |A| = |B|) \\
&\le \sum_{\tau_i \in \Gamma^{(k,\ell)}} e_i + \sum_{\tau_i \in \Pi^{(k,\ell)}} b_i + \sum_{i=1}^{c} \beta_i && \text{(by (41) and } C = \tau' \setminus A\text{, } |B| + |C| = |\tau'|\text{; by (38), } |\tau'| = c\text{; by their definitions, } B \text{ and } C \text{ are disjoint).}
\end{aligned}$$

The above establishes Part (i). Part (ii) can be shown to hold as follows. Because tasks in A are in $\Gamma^{(k+c,\ell)}$, whereas those in B are in $\Pi^{(k+c,\ell)}$, by (3) and (4), we have

$$(\forall \tau_i, \tau_j : \tau_i \in A, \tau_j \in B :: e_i - b_i \geq e_j - b_j). \tag{43}$$

Finally,

$$\begin{aligned}
&\sum_{\tau_i \in \Gamma^{(k+c,\ell)}} e_i + \sum_{\tau_i \in \Pi^{(k+c,\ell)}} b_i \\
&= \sum_{\tau_i \in \Gamma^{(k,\ell)}} e_i + \sum_{\tau_i \in A} e_i - \sum_{\tau_i \in B} e_i + \sum_{\tau_i \in \Pi^{(k,\ell)}} b_i + \sum_{\tau_i \in C} b_i + \sum_{\tau_i \in B} b_i \quad \text{(by (39), (40), and (42)} \\
&\geq \sum_{\tau_i \in \Gamma^{(k,\ell)}} e_i + \sum_{\tau_i \in \Pi^{(k,\ell)}} b_i + \sum_{\tau_i \in A} b_i + \sum_{\tau_i \in C} b_i \quad \text{(by (43), since } |A| = |B| \text{ by (41))} \\
&\geq \sum_{\tau_i \in \Gamma^{(k,\ell)}} e_i + \sum_{\tau_i \in \Pi^{(k,\ell)}} b_i,
\end{aligned}$$

establishing Part (ii). ■

We prove two auxiliary lemmas before proving Lemma 7.

Lemma 13 $\sum_{\tau_i \in \Gamma^{(k,\ell+c)}} e_i + \sum_{\tau_i \in \Pi^{(k,\ell+c)}} b_i \geq \sum_{\tau_i \in \Gamma^{(k,\ell)}} e_i + \sum_{\tau_i \in \Pi^{(k,\ell)}} b_i$, *where* $0 \leq \ell \leq k \leq N$ *and* $0 \leq c \leq k - \ell$.

Proof: The proof is by induction on c. It is easy to see that the lemma holds trivially when $c = 0$, which forms the base case. For the induction hypothesis, assume that the lemma holds for all c, where $0 \leq c \leq c'$, where $c' < k - \ell$. For the induction step, we show that $\sum_{\tau_i \in \Gamma^{(k,\ell+c'+1)}} e_i + \sum_{\tau_i \in \Pi^{(k,\ell+c'+1)}} b_i \geq \sum_{\tau_i \in \Gamma^{(k,\ell+c')}} e_i + \sum_{\tau_i \in \Pi^{(k,\ell+c')}} b_i$, which, by the induction hypothesis establishes the lemma. By (2) and (3), $\Gamma^{(k,\ell+c'+1)}$ includes one more task that is in $\Gamma^{(k)}$ in addition to every task in $\Gamma^{(k,\ell+c')}$. By (4), this additional task, say τ_j, is from $\Pi^{(k,\ell+c')}$. Also, $\Pi^{(k,\ell+c'+1)}$ contains every task in $\Pi^{(k,\ell+c')}$ except τ_j. Therefore,

$$\begin{aligned}
\sum_{\tau_i \in \Gamma^{(k,\ell+c'+1)}} e_i + \sum_{\tau_i \in \Pi^{(k,\ell+c'+1)}} b_i &= \sum_{\tau_i \in \Gamma^{(k,\ell+c')}} e_i + e_j + \sum_{\tau_i \in \Pi^{(k,\ell+c')}} b_i - b_j \\
&\geq \sum_{\tau_i \in \Gamma^{(k,\ell+c')}} e_i + \sum_{\tau_i \in \Pi^{(k,\ell+c')}} b_i \qquad \text{(because } e_j \geq b_j\text{).}
\end{aligned}$$

■

Lemma 14 *Let* α *and* β *be any two* ***disjoint*** *subsets of tasks in* τ *such that* $|\alpha| = k_2$ *and* $|\alpha| + |\beta| = k_1$*. Then,* $\sum_{\tau_i \in \alpha} e_i + \sum_{\tau_i \in \beta} b_i \leq \sum_{\tau_i \in \Gamma^{(k_1,k_2)}} e_i + \sum_{\tau_i \in \Pi^{(k_1,k_2)}} b_i$.

Proof: To prove this lemma, we need to show that there do not exist disjoint subsets α and β of τ with cardinalities k_2 and $k_1 - k_2$, respectively, such that the sum of the execution costs of tasks in α and the non-preemptive section costs of tasks in β is greater than the sum of the execution costs of tasks in $\Gamma^{(k_1,k_2)}$ and the non-preemptive costs of tasks in $\Pi^{(k_1,k_2)}$.

Let $\tau' = \tau \setminus (\Gamma^{(k_1,k_2)} \cup \Pi^{(k_1,k_2)})$. Then, by (5), $\tau' = \tau \setminus \Gamma^{(k_1)}$. By (3), $\Gamma^{(k_1,k_2)} \subseteq \Gamma^{(k_1)}$ holds. Therefore, by the definition in (2), no task in τ' has a higher execution cost than a task in $\Gamma^{(k_1,k_2)}$. Similarly, by (4), $\Pi^{(k_1,k_2)} \subseteq \Gamma^{(k_1)}$ holds, and, hence, by (2) and (1), no task in τ' has a higher non-preemptive section cost than a task in $\Pi^{(k_1,k_2)}$. Therefore, replacing a task in $\Gamma^{(k_1,k_2)}$ or $\Pi^{(k_1,k_2)}$ by a task in τ' will not lead to a pair of subsets α and β such that the lemma is contradicted. Hence, we are left with showing that there do not exist subsets α and β, and tasks τ_i and τ_j, such that $\alpha \cup \beta = \Gamma^{(k_1,k_2)} \cup \Pi^{(k_1,k_2)}$, $\tau_i \in \Gamma^{(k_1,k_2)}$, and $\tau_j \in \Pi^{(k_1,k_2)}$, and $\alpha = (\Gamma^{(k_1,k_2)} \setminus \tau_i) \cup \{\tau_j\}$ and $\beta = (\Pi^{(k_1,k_2)} \setminus \tau_j) \cup \{\tau_i\}$, for which $\sum_{\tau_i \in \alpha} e_i + \sum_{\tau_i \in \beta} b_i > \sum_{\tau_i \in \Gamma^{(k_1,k_2)}} e_i + \sum_{\tau_i \in \Pi^{(k_1,k_2)}} b_i$. (That is, we are left with showing that α and β derived from $\Gamma^{(k_1,k_2)}$ and $\Pi^{(k_1,k_2)}$ by interchanging the set memberships of tasks τ_i and τ_j cannot contradict the lemma.) Since τ_i and τ_j are as defined, it suffices to show that $e_i + b_j \geq e_j + b_i$. For this, note that by the definitions in (3) and (4), $e_i - b_i \geq e_j - b_j$ holds, which implies $e_i + b_j \geq e_j + b_i$. ■

Lemma 7 *Let α and β be any two* ***disjoint*** *subsets of tasks in τ such that $|\alpha| \leq \ell$ and $|\alpha| + |\beta| \leq k$, where $0 \leq \ell \leq k \leq N$. Then, $\sum_{\tau_i \in \alpha} e_i + \sum_{\tau_i \in \beta} b_i \leq \sum_{\tau_i \in \Gamma^{(k,\ell)}} e_i + \sum_{\tau_i \in \Pi^{(k,\ell)}} b_i$.*

Proof: By Lemma 14, $\sum_{\tau_i \in \alpha} e_i + \sum_{\tau_i \in \beta} b_i \leq \sum_{\tau_i \in \Gamma^{(|\alpha|+|\beta|,|\alpha|)}} e_i + \sum_{\tau_i \in \Pi^{(|\alpha|+|\beta|,|\alpha|)}} b_i$ holds. By part (ii) of Lemma 6, $\sum_{\tau_i \in \Gamma^{(|\alpha|+|\beta|,|\alpha|)}} e_i + \sum_{\tau_i \in \Pi^{(|\alpha|+|\beta|,|\alpha|)}} b_i \leq \sum_{\tau_i \in \Gamma^{(k,|\alpha|)}} e_i + \sum_{\tau_i \in \Pi^{(k,|\alpha|)}} b_i$ holds. Lastly, by Lemma 13, $\sum_{\tau_i \in \Gamma^{(k,|\alpha|)}} e_i + \sum_{\tau_i \in \Pi^{(k,|\alpha|)}} b_i \leq \sum_{\tau_i \in \Gamma^{(k,\ell)}} e_i + \sum_{\tau_i \in \Pi^{(k,\ell)}} b_i$ holds, establishing Lemma 7. ■

B Tables of Symbols and Frequently-Used Expressions

In this appendix, we list some frequently-used symbols and expressions for ease of reference.

B.1 Table of Symbols

Symbol	Description
M	number of processors
τ	set of all tasks
N	number of tasks in τ
τ_i	the i^{th} task of τ
e_i	worst-case execution cost of τ_i
p_i	period of τ_i
D_i	relative deadline of τ_i
b_i	maximum execution cost of any non-preemptive segment of τ_i
u_i	utilization of τ_i

$e_{\max}$	maximum worst-case execution cost of any task in τ
$u_{\max}$	maximum utilization of any task in τ
ϵ_i	the i^{th} execution cost in a non-increasing order of the execution costs of all the tasks in τ
μ_i	the i^{th} utilization in a non-increasing order of the utilizations of all the tasks in τ
$U_{sum}(\tau)$	total system utilization of τ
$U_{sum}(\tau, t)$	instantaneous total utilization of τ at t
$\tau_{i,j}$	the j^{th} job of τ_i
$r_{i,j}$	release time of $\tau_{i,j}$
$d_{i,j}$	absolute deadline of $\tau_{i,j}$
$\ell_{i,j}$	eligibility time, which is at most $r_{i,j}$, of $\tau_{i,j}$
$\Gamma^{(k)}$	subset of k tasks of τ with the highest execution costs
$\Gamma^{(k_1,k_2)}$	subset of k_2 tasks τ_i of $\Gamma^{(k_1)}$ with the highest values for $e_i - b_i$
$\Pi^{(k_1,k_2)}$	$\Gamma^{(k_1)} \setminus \Gamma^{(k_1,k_2)}$
PS	processor sharing schedule
PS_τ	processor sharing schedule for τ
$\mathsf{A}(\mathcal{S}, \tau_i, t_1, t_2)$	the total allocation to τ_i in schedule $\mathcal{S}$ for τ in $[t_1, t_2)$
$\mathsf{lag}(\tau_i, t, \mathcal{S})$	the lag, that is, the difference in allocation in comparison to that in a PS schedule, for τ_i at time t in $\mathcal{S}$
$\mathsf{LAG}(\tau, t, \mathcal{S})$	the total system lag for τ at time t in $\mathcal{S}$
$\tau_{\ell,j}$	job whose tardiness is presumed to exceed $x + e_\ell$ in the proof of Theorem 1
t_d	absolute deadline of $\tau_{\ell,j}$
Ψ	in the proof of Theorem 1, the set of all jobs of tasks in τ with deadline at or before t_d
$\overline{\Psi}$	set of all jobs of tasks in τ that are not in Ψ
$\mathsf{LAG}(\Psi, t, \mathcal{S})$	the total lag for jobs in Ψ at time t in $\mathcal{S}$
$\mathcal{B}(\tau, \Psi, t_d, \mathcal{S})$	the set of all blocking jobs in τ for Ψ with pending blocking work at t_d (due to non-preemptive sections that commenced execution before t_d and are pending at t_d)
$\mathsf{B}(\tau, \Psi, t_d, \mathcal{S})$	pending blocking work for Ψ at t_d due to jobs in $\mathcal{B}(\tau, \Psi, t_d, \mathcal{S})$
ρ	minimum number of jobs with deadlines at or after the end of a non-blocking, non-busy interval with continuously increasing LAG guaranteed to execute at the end of that interval

B.2 Table of Frequently-Used Expressions

Expression No. in the paper	Formula/Equation/Inequality
(2)	$\Gamma^{(k)} \stackrel{\text{def}}{=}$ Subset of k tasks of τ with the highest execution costs
(3)	$\Gamma^{(k_1,k_2)} \stackrel{\text{def}}{=}$ Subset of k_2 tasks τ_i of $\Gamma^{(k_1)}$ with the highest values for $e_i - b_i$
(4)	$\Pi^{(k_1,k_2)} \stackrel{\text{def}}{=} \Gamma^{(k_1)} \setminus \Gamma^{(k_1,k_2)}$
(5)	$\Gamma^{(k_1,k_2)} \cup \Pi^{(k_1,k_2)} = \Gamma^{(k_1)}$
(6)	$\Lambda = \begin{cases} U_{sum}(\tau) - 1, & U_{sum}(\tau) \text{ is integral} \\ \lfloor U_{sum}(\tau) \rfloor, & \text{otherwise} \end{cases}$
(11)	$\mathsf{LAG}(\tau, t_2, \mathcal{S}) = \mathsf{LAG}(\tau, t_1, \mathcal{S}) + \mathsf{A}(\mathsf{PS}_\tau, \tau, t_1, t_2) - \mathsf{A}(\mathcal{S}, \tau, t_1, t_2)$
(13)	$\mathsf{LAG}(\Psi, t, \mathcal{S}) \leq \sum_{\{\tau_i \in \tau \mid \text{ some } \tau_{i,j} \text{ is in } \Psi \text{ and is pending at } t^-\}} \mathsf{lag}(\tau_i, t, \mathcal{S})$
(17)	$\mathsf{LAG}(\Psi, t_2, \mathcal{S}) = \mathsf{LAG}(\Psi, t_1, \mathcal{S}) + \mathsf{A}(\mathsf{PS}_\tau, \Psi, t_1, t_2) - \mathsf{A}(\mathcal{S}, \Psi, t_1, t_2)$