A Schedulable Utilization Bound for the Multiprocessor EPDF Pfair Algorithm*

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

Abstract

The earliest-pseudo-deadline-first (EPDF) Pfair scheduling algorithm is less expensive than some other known Pfair algorithms, but is not optimal for scheduling recurrent real-time tasks on more than two processors. In prior work, sufficient per-task weight (*i.e.*, utilization) restrictions were established for ensuring that tasks either do not miss their deadlines or have bounded tardiness when scheduled under EPDF. Implicit in these restrictions is the assumption that the total system utilization may equal the total available processing capacity (*i.e.*, the total number of processors). This paper considers an orthogonal issue, namely, determining a sufficient restriction on the total utilization of a task set for it to be schedulable (*i.e.*, a schedulable utilization bound) under EPDF, assuming that there are no per-task weight restrictions. We prove that a task set with total utilization at most $\frac{3M+1}{4}$ is correctly scheduled under EPDF on M processors, regardless of how large each task's weight is. At present, we do not know whether this value represents the worst-case for EPDF, but we do provide a counterexample that shows that it cannot be improved to exceed 86% of the total processing capacity. The schedulable utilization bound we derive is expressed in terms of the maximum weight of any task, and hence, if this value is known, may be used to schedule task sets with total utilization greater than $\frac{3M+1}{4}$.

*Work supported by NSF grants CCR 9988327, ITR 0082866, CCR 0204312, CNS 0309825, and CNS 0615197, and by ARO grant W911NF-06-1-0425. The first author was also supported by an IBM Ph.D. fellowship.
Contents of this paper appeared in preliminary form in the following paper:
U. Devi and J. Anderson. Schedulable utilization bounds for EPDF fair multiprocessor scheduling. In *Proceedings of the 10th International Conference on Real-Time and Embedded Computing Systems and Applications, Springer-Verlag Lecture Notes in Computer Science*, pages 261–280, August 2004.

1 Introduction

We consider the scheduling of recurrent (*i.e.*, periodic, sporadic, or rate-based) real-time task systems on multiprocessor platforms consisting of M identical, unit-capacity processors. Such a task system consists of a collection of sequential tasks, each of which is invoked repeatedly; as explained in more detail later (in Sec. 2), each invocation is called a *job*, and jobs are subject to deadline constraints. Pfair scheduling, originally introduced by Baruah *et al.* [15], is the only known way of *optimally*[1] scheduling such multiprocessor task systems.[2] To ensure optimality, Pfair scheduling imposes a stronger constraint on the timeliness of tasks than that mandated by periodic scheduling. Specifically, each task must execute at an approximately uniform rate at all times, while respecting a fixed-size allocation quantum. A task's execution rate is defined by its *weight* (*i.e.*, *utilization*). Uniform rates are ensured by subdividing each task into quantum-length *subtasks* that are subject to intermediate deadlines, called *pseudo-deadlines*, determined based on the task's weight.

Viability of Pfair scheduling. Due to its theoretical optimality, Pfair scheduling is superior to other approaches under ideal conditions. However, in practice, context switches incurred due to its frequent task preemptions and migrations can negate the schedulability gain enabled by intelligent scheduling. The main overhead incurred due to a preemption or migration is a potential loss of cache affinity, which can cause task execution costs to worsen. Consequently, Pfair scheduling is often dismissed as an impractical approach. However, such a dismissive attitude is not always warranted due to the following reasons.

First, empirical evidence now exists that shows that Pfair scheduling is a practical alternative to other approaches. In particular, Calandrino *et al.* recently conducted an empirical evaluation of several multiprocessor scheduling approaches on a four-processor Linux testbed [22]. The approaches considered by them included partitioning, Pfair-based global scheduling, and non-Pfair-based global scheduling. Under partitioning, tasks are statically assigned to processors, while under global scheduling, they may migrate. The results presented in [22] show that there exist scenarios in which Pfair scheduling is either competitive with or outperforms other approaches.

Second, in dynamic task systems, in which tasks may leave or join the system or change weights, Pfair scheduling has some definitive advantages over other approaches. For such systems, partitioning is a cumbersome strategy due to the need to repartition when task-set changes occur, and if hard real-time guarantees are also needed, then non-Pfair-based global algorithms, such as global EDF, may also not be viable due to their low schedulability. It is worth noting that the evaluation of Calandrino *et al.* focused only on static systems.

Third, recent trends in processor architectures favor designs that can considerably mitigate migration overheads, and, hence, augur well for Pfair scheduling. The most significant such trend is the advent of multicore architectures, which are being adopted as a way of circumventing the impediments posed in continuing processor performance

[1] A real-time scheduling algorithm is said to be *optimal* iff it can schedule without deadline misses every feasible task system, *i.e.*, every task system for which some correct schedule exists.

[2] In recent work, Cho *et al.* [24] presented the *lowest local-remaining-execution first* (LLREF) algorithm, which is optimal on multiprocessors for scheduling periodic task systems. LLREF requires that the arrival time of every job be known *a priori*, so it cannot be used to schedule sporadic or rate-based tasks. Thus, it is not a viable approach for most of the application contexts considered later in this section that have motivated the research reported herein.

improvements in conventional single-core chips. Multicore architectures have multiple processing cores on a single chip, and in some designs, the different cores share one or more levels of on- or off-chip caches of considerable capacity. For example, the four cores of the Intel® Xeon® 5300 series chips share an 8-MB on-chip L2 cache [1]. In such systems, tasks do not significantly lose cache affinity when they migrate (as long as the miss rates of the shared caches are kept low), so overheads due to migrations are less of a concern.

Tie-breaking rules and their overhead. Under most known Pfair algorithms, subtasks are scheduled on an earliest-pseudo-deadline-first basis. However, to avoid deadline misses, ties among subtasks with the same deadline must be broken carefully. In [43], Srinivasan and Anderson observed that overheads associated with the tie-breaking rules of optimal Pfair algorithms may be unnecessary or unacceptable for many *soft* real-time task systems. PD^2 [6], the most efficient of the known optimal Pfair algorithms, requires two tie-break parameters, the *b-bit* and the *group deadline*. (These tie-break parameters are described in Sec. 3.) Though these parameters can be computed for each subtask in constant time, as explained below, there exist many scenarios in which eliminating them is preferable.

Under Pfair scheduling, up to M highest-priority subtasks are chosen for execution at the beginning of each time quantum. Hence, the use of the tie-breaking rules will incur, in the worst case, an overhead of $2 \cdot \mathcal{O}(\log N)$ comparisons per scheduling decision, and, hence, $2 \cdot \mathcal{O}(M \log N)$ comparisons per quantum, where N denotes the number of tasks. Though the inclusion of tie-breaking rules does not impact the asymptotic time complexity of scheduling, eliminating them may be preferable in embedded systems with slower processors or limited memory bandwidth. Other applications where this may be desirable as well include web-hosting systems [23] and server farms, packet processing in programmable multiprocessor-based network processors or routers [45], packet transmission on multiple, parallel outgoing router links [3], and optical networks that use wave-division multiplexing (WDM) to transmit multiple light packets simultaneously [11]. In the networking applications mentioned, packets must be processed and/or scheduled for transmission at link speeds. Since link speeds are typically much greater than processor speeds [25, 2], it may be undesirable or even unacceptable to include tie-breaking rules in scheduling packet-processing tasks. Tie-breaking rules are undesirable in web-hosting systems also for a similar reason. Finally, in some scalable network architectures for providing per-flow service guarantees, per-flow state is stored at the edge router only; the priority details of a packet are computed using flow state at the edge router and this information is transmitted in the packet header [46, 47, 32, 33]. This is in contrast to the traditional Integrated Services (IntServ) architecture [20], in which the state of every flow is stored and updated in every core router. The IntServ architecture is hence not scalable, but provides the ability to compute packet priorities in any router. In the scalable architectures, eliminating tie-breaking rules would help conserve scarce header bits.

Tie-breaking rules may especially be problematic in dynamic task systems. Some of the applications mentioned above are also dynamic. As explained in [43], it is possible to *reweight* each task whenever its utilization changes such that its next subtask deadline is preserved. If no tie-breaking information is maintained, such an approach entails very little computational overhead. However, utilization changes can cause tie-breaking information to change, so if tie-breaking rules are used, then reweighting may necessitate an $\mathcal{O}(N)$ cost for N tasks, due to the need to reorder the scheduler's priority queue. This cost may be prohibitive if reallocations are frequent. (This reweighting overhead is illustrated in more detail in Sec. 3.2.)

Prior research. The tie-breaking overheads described above motivated Srinivasan and Anderson to consider the viability of scheduling soft real-time task systems using the simpler *earliest-pseudo-deadline-first* (EPDF) Pfair algorithm, which uses no tie-breaking rules. A soft real-time task differs from a hard real-time task in that its deadlines may be *occasionally* missed. If a job or a subtask with a deadline at time d completes executing at time t, then it is said to have a *tardiness* of $\max(0, t - d)$. Srinivasan and Anderson succeeded in showing that EPDF is optimal on up to two processors [5], and that if each task's weight is at most $\frac{q}{q+1}$, then the algorithm guarantees a tardiness of at most q quanta for every subtask [43]. In later work [27, 26], we showed that this condition can be improved to $\frac{q+2}{q+3}$. With either condition, the total utilization of a task set may equal M, where M denotes the number of processors.

This research. In this paper, we consider the viability of EPDF for scheduling *hard* real-time task systems. Specifically, we seek to determine a schedulability test that is as accurate as possible, and which can be used to *a priori* show that a task system will not miss deadlines under EPDF. Such a test would enable less-expensive scheduling for hard real-time systems that satisfy the test and dynamic task systems where tie-breaking rules are problematic.

Furthermore, a schedulability test for EPDF can enable the use of the algorithm in some systems with heterogeneous timing constraints, specifically, those in which the hard real-time (HRT) component does not fully utilize the system and co-exists with soft real-time (SRT) and/or best-effort (BE) tasks. As pointed out in [39] and [21], such heterogeneous workloads are becoming increasingly common. One way of scheduling such a task system is to statically prioritize the HRT tasks over other tasks, and to schedule the HRT tasks using an appropriate scheduler. For the following reasons, apart from its simplicity and flexibility, EPDF may be a better choice for this scheduler than optimal Pfair algorithms or partitioning. First, since the utilization of the HRT component is likely to be low [39], and within what is permissible for EPDF, schedulability need not be sacrificed. Second, changes in the composition of HRT tasks can be more seamlessly supported (as described above and in Sec. 3.2). Third, because tasks execute more uniformly under Pfair scheduling, it may be possible to provide better bounds on response times for SRT and BE tasks.

Finally, we believe that studying EPDF is also of academic and theoretical interest, and may help in advancing the state-of-the-art in research on global scheduling algorithms. We believe that a study of EPDF could add to our general understanding of global scheduling algorithms, help in quantifying the "power" of tie-breaking rules, and lead to techniques that can be applied in the analysis of other global algorithms.

Motivated by the above reasons, in this paper, we address the following question, which is orthogonal to that addressed in prior work: If individual tasks cannot be subject to weight restrictions, then what would be a sufficient restriction on the total utilization of a task set for it to be scheduled correctly under EPDF? We answer this question by deriving a *schedulable utilization bound* for EPDF. (The notion of a schedulable utilization bound is formally defined in Sec. 2.)

Contributions. Our first contribution is to show that on M processors, EPDF can correctly schedule every task system with total utilization at most $\min(M, \frac{\lambda M(\lambda(1+W_{\max})-W_{\max})+1+W_{\max}}{\lambda^2(1+W_{\max})})$, where $\lambda = \max(2, \left\lceil \frac{1}{W_{\max}} \right\rceil)$. For $W_{\max} \geq \frac{1}{2}$, *i.e.*, $\lambda = 2$, this value reduces to $\frac{(2M+1)(2+W_{\max})-1}{4(1+W_{\max})}$, and as $W_{\max} \to 1.0$, it approaches $\frac{3M+1}{4}$, which, as $M \to \infty$, approaches $\frac{3M}{4}$, *i.e.*, 75% of the total processing capacity. As discussed in Sec. 2, for $W_{\max} > 1/2$, the schedulable utilization bound that we have derived for EPDF is greater than that of every known non-Pfair algorithm by 25%. Currently, it is not known whether $\frac{3M+1}{4}$ is the worst-case schedulable utilization, let alone, optimal utilization bound for EPDF. (These terms are defined formally in Sec. 2.) The closest that we have come to in assessing the accuracy of our result is a counterexample that shows that when $W_{\max} \geq \frac{5}{6}$, worst-case schedulable utilization cannot exceed 86%. We have considered extending our result for use in systems where tardiness may be permissible. As as second contribution, we discuss permissible system utilizations for a tardiness of $q > 0$ quanta.

Organization. The rest of the paper is organized as follows. Sec. 2 describes our task model and discusses the concept of a schedulable utilization bound. Sec. 3 provides an overview of Pfair scheduling. In Sec. 4, the utilization bound for EPDF mentioned above is derived. Sec. 5 concludes. Proofs omitted in the main paper are provided in an appendix.

2 Definitions

In this section, we describe our system model, formally define the notion of a schedulable utilization bound, and discuss some associated concepts and known utilization bounds for some algorithms.

Task model. A recurrent task system τ consists of N *periodic*, *sporadic*, or *rate-based* tasks. Each task T of τ is assigned a rational *weight* (*i.e.*, *utilization*) $wt(T) \in (0, 1]$ that denotes the fraction of a single processor it requires. For a periodic or a sporadic task T, $wt(T) = T.e/T.p$, where $T.e$ and $T.p$ are the *execution cost* and inter-arrival time or *period*, respectively, of T. A task is *light* if its weight is less than 1/2, and *heavy*, otherwise. Each task is sequential, and at any time may execute on at most one processor.

A periodic or sporadic task T may be invoked zero or more times; T is *periodic* if its consecutive invocations or arrivals are separated by exactly $T.p$ time units and is *sporadic* if the separation is at least $T.p$ time units. Each invocation or release of T is referred to as a *job* of T. The first job of T may be invoked or *released* at any time at or after time zero. Every job of T executes for at most $T.e$ time units. In this paper, we consider task systems in which every job of T has a *relative deadline* of $T.p$ time units, *i.e.*, should complete execution within $T.p$ time units of its release. Such systems are also referred to as *implicit-deadline systems*. (For ease of notation and description, we assume that each job of T executes for exactly $T.e$ time units.)

A rate-based task is a generalization of a sporadic task and can be used to model systems in which task arrivals are neither periodic nor sporadic but may be highly jittered [31]. The weight of a rate-based task denotes its average or expected arrival rate. No restriction is imposed on the instantaneous arrival rate of such a task, and an instantaneous rate that exceeds the expected is handled by simply postponing task deadlines. The *intra-sporadic* and *early-release* task models discussed in Sec. 3 can be used to characterize this behavior.

Schedulable utilization bounds. If $\mathcal{U}_{\mathcal{A}}(M, \alpha)$ is a schedulable utilization bound, or more concisely, *utilization bound*, for scheduling algorithm $\mathcal{A}$, then on M processors, $\mathcal{A}$ can correctly schedule every recurrent task system τ with $u_{\max}(\tau) \leq \alpha$ and $U_{sum}(\tau) \leq \mathcal{U}_{\mathcal{A}}(M, \alpha)$, where $u_{\max}(\tau)$ is the maximum utilization of any task in τ and $U_{sum}(\tau)$ is the total utilization of all the tasks in τ. If, in addition, there exists at least one task system with total utilization $\mathcal{U}_{\mathcal{A}}(M, \alpha)$ and $u_{\max} = \alpha$, whose task parameters can be slightly modified such that U_{sum} and $u_{\max}$ are greater than $\mathcal{U}_{\mathcal{A}}(M, \alpha)$ and α, respectively, by infinitesimal amounts, and the modified task system has a deadline miss under $\mathcal{A}$ on M processors, then $\mathcal{U}_{\mathcal{A}}(M, \alpha)$ is said to be the *minimax utilization*[3] of $\mathcal{A}$ for M and $u_{\max}$; otherwise, $\mathcal{U}_{\mathcal{A}}(M, \alpha)$ is a lower bound on $\mathcal{A}$'s minimax utilization for M and $u_{\max}$. The minimax utilization of $\mathcal{A}$ is also referred to as the *worst-case schedulable utilization*[4] of $\mathcal{A}$. Furthermore, if no task system with total utilization exceeding $\mathcal{U}_{\mathcal{A}}(M, \alpha)$ can be scheduled correctly by $\mathcal{A}$ when $u_{\max} = \alpha$ on M processors, then $\mathcal{U}_{\mathcal{A}}(M, \alpha)$ is said to be the *optimal utilization bound* of $\mathcal{A}$ for M and $u_{\max}$. Note that, while an optimal utilization bound is also a worst-case schedulable utilization, the converse may not hold. This is because it is possible that there exist task systems that are correctly scheduled but have a total utilization that is higher than that of some other task system that is barely schedulable. It should also be noted that when expressed using a fixed set of parameters, such as α and M, different values are not possible for the optimal bound and the worst-case schedulable utilization. In other words, if an optimal utilization bound exists or is known for M and α, then a worst-case schedulable utilization that is different from the optimal bound is not possible.

For implicit-deadline systems (considered in this paper), utilization bounds can be used for devising simple and fast schedulability tests and online admission-control tests. Given a utilization bound $\mathcal{U}_{\mathcal{A}}(M, \alpha)$ of Algorithm $\mathcal{A}$ and a system τ of N tasks, an $\mathcal{O}(N)$-time schedulability test for τ under $\mathcal{A}$ that verifies whether the total system utilization is at most $\mathcal{U}_{\mathcal{A}}(M, \alpha)$, and a similar $\mathcal{O}(1)$-time per-task online admission control, are straightforward. The downside of such utilization-based schedulability tests is that for many algorithms, optimal utilization bounds are not known. In such cases, the tests are only sufficient but not necessary (*i.e.*, are not *exact* tests), and hence, can be pessimistic, *i.e.*, incorrectly conclude that deadlines may be missed. On the other hand, more accurate tests, such as those based on time-demand analysis [35, 34] have higher time complexity, and hence, when timeliness is a concern, as in online admission-control tests, utilization-based tests are usually preferred. The concept of *load*, as defined in [9, 29], has also been used to obtain some useful schedulability tests [18, 16]. However, for implicit-deadline systems, load generally is the same as utilization and does not yield more accurate tests than those based on utilization, and hence, load-based tests are not of interest for the task models considered in this paper.

Uniprocessor utilization bounds. Optimal bounds or worst-case schedulable utilizations are known for several scheduling algorithms. (In the discussion of known utilization bounds that follows, tasks are assumed to be periodic or sporadic, with the relative deadline of each task equal to its period.) In the domain of uniprocessor scheduling, a utilization bound of 1.0 is optimal for preemptive earliest-deadline-first (EDF) scheduling, while $N(2^{1/N} - 1)$ is

[3]The phrase minimax utilization is due to Oh and Baker [38] and denotes the minimum utilization over all maximal task sets. A maximal task set is one that is schedulable but if the execution times of its tasks are increased slightly, then some deadline will be missed.

[4]Unless otherwise specified, worst-case schedulable utilizations are assumed to be with respect to M and $\alpha = u_{\max}$. It is possible to obtain other worst-case values if other or more task parameters, such as execution costs and periods, are considered.

the worst-case schedulable utilization for preemptive RM scheduling with respect to the number of tasks, N [36]. This value converges to $\ln 2 \approx 0.69$ as $N \to \infty$.

Multiprocessor utilization bounds. In general, multiprocessor scheduling algorithms use either a *partitioned* or *global* scheduling approach. Under partitioned scheduling, tasks are assigned to processors by defining a many-to-one mapping from the set of tasks to the set of processors. In other words, tasks are partitioned among processors, and a separate instance of a uniprocessor scheduling algorithm is used to schedule the tasks assigned to each processor. Therefore, the partitioning algorithm has to ensure that on each processor, the deadlines of the tasks assigned to it will be met. In general, this is done by ensuring that on each processor, the sum of the utilizations of the tasks assigned does not exceed the best known utilization bound of its scheduler. If $W_{\max}$, where $0 < W_{\max} \leq 1$, denotes the maximum weight of any task and EDF is the per-processor scheduling algorithm used, then the worst-case schedulable utilization for the partitioned approach with respect to the parameters M and $W_{\max}$ is $\frac{\beta M+1}{\beta+1}$, where $\beta = \left\lfloor \frac{1}{W_{\max}} \right\rfloor$ [37]. This value approaches $\frac{M+1}{2}$ as $W_{\max} \to 1.0$. Because EDF is an optimal uniprocessor scheduling algorithm, a higher utilization bound is not possible with any other per-processor scheduling algorithm.

Under global scheduling, a task may execute on any processor. The global approach can be differentiated further based on whether a preempted job is allowed to resume execution on a different processor. If each job is bound to a single processor only, then migrations are said to be *restricted*; otherwise, they are *unrestricted*. EDF, which is a restricted dynamic-priority algorithm,[5] has a worst-case schedulable utilization of $M - (M-1) \cdot u_{\max}$ when deployed globally. For all $M \geq 2$, this value approaches 1.0 as $u_{\max}$ approaches 1.0, implying that for each $M \geq 2$, there exist task systems with utilization arbitrarily close to 1.0 that cannot be correctly scheduled on M processors. This phenomenon, now referred to as the "Dhall" effect, was first noted by Dhall and Liu as early as 1978 [28]. RM, which is a static-priority scheduling algorithm, is also susceptible to this effect. It has been shown that within global scheduling, a utilization bound exceeding $\frac{M+1}{2}$ is impossible under either restricted dynamic-priority algorithms [13, 14] or static-priority algorithms [7, 8], regardless of the nature of migrations (restricted or not). Thus, the worst-case schedulable utilization for each multiprocessor scheduling algorithm considered so far converges to approximately 50% of the total processing capacity.

Most Pfair scheduling algorithms allow migrations, and hence, fall under the global category. As mentioned earlier, optimal scheduling on multiprocessors is possible with Pfair scheduling. Therefore, each of the optimal Pfair algorithms, PF [15], PD [17], and PD^2 [42], has an optimal utilization bound of M. *Weight monotonic* (WM) is another Pfair scheduling algorithm that has been proposed [12, 40]. WM uses static priorities based on task weights for scheduling subtasks and is not optimal. The current best known utilization bound for WM is $M/2$ [8].

[5]Under *restricted dynamic-priority* algorithms, the priority of a job cannot be changed once assigned. On the other hand, if job priorities can change with time, then the algorithm is said to be an *unrestricted dynamic-priority* algorithm.

3 Background on Pfair Scheduling

This section describes some basic concepts of Pfair scheduling, provides needed background, and summarizes results from prior work reported in [4, 5, 6, 15, 42, 43].[6]

3.1 Preliminaries

Pfair scheduling [15, 42] can be used to schedule a periodic, sporadic, *intra-sporadic* (IS), or *generalized-intra-sporadic* (GIS) (see below) task system τ on $M \geq 1$ processors. Periodic and sporadic tasks were described in Sec. 2.

When scheduled under Pfair algorithms, it is required that for each task T, $T.e$ and $T.p$, be specified as integers, which are interpreted as integral number of quanta. Pfair algorithms allocate processor time in discrete quanta; the time interval $[t, t+1)$, where t is a nonnegative integer, is called *slot* t. (Hence, time t refers to the beginning of slot t.) All references to time are non-negative integers. The interval $[t_1, t_2)$, consists of slots $t_1, t_1+1, \ldots, t_2-1$. A task may be allocated time on different processors, but not in the same slot (*i.e.*, interprocessor migration is allowed but parallelism is not). Similarly, on each processor, at most one task may be allocated in each slot. The sequence of allocation decisions over time defines a *schedule* $\mathcal{S}$. Formally, $\mathcal{S} : \tau \times \mathbf{N} \mapsto \{0, 1\}$, where $\mathbf{N}$ is the set of nonnegative integers. $\mathcal{S}(T, t) = 1$ iff T is scheduled in slot t. On M processors, $\sum_{T \in \tau} \mathcal{S}(T, t) \leq M$ holds for all t.

Periodic, sporadic, and IS task models. In Pfair scheduling, each quantum of execution of each task is referred to as a *subtask*. Each task gives rise to a potentially infinite sequence of subtasks. The i^{th} subtask of T, where $i \geq 1$, is denoted T_i. If T is periodic or sporadic, then the k^{th} job of T, where $k \geq 1$, consists of subtasks $T_{(k-1)\cdot e+1}, \ldots, T_{k \cdot e}$, where $e = T.e$.

Each subtask T_i has an associated *pseudo-release* $r(T_i)$ and *pseudo-deadline* $d(T_i)$, defined as follows. (The prefix "pseudo-" is often omitted for conciseness.)

$$r(T_i) = \Theta(T_i) + \left\lfloor \frac{i-1}{wt(T)} \right\rfloor \qquad \wedge \qquad d(T_i) = \Theta(T_i) + \left\lceil \frac{i}{wt(T)} \right\rceil \tag{1}$$

In the above formulas, $\Theta(T_i) \geq 0$ denotes the *offset* of T_i and is used in modeling the late releases of sporadic and IS tasks. It is well known that the sporadic model generalizes the periodic model by allowing jobs to be released "late;" the IS model, proposed by Srinivasan and Anderson in [5, 42], is a further generalization that allows subtasks to be released late. The offsets of T's various subtasks are nonnegative and satisfy the following: $k > i \Rightarrow \Theta(T_k) \geq \Theta(T_i)$. T is *periodic* if $\Theta(T_i) = c$ holds for all i (and is *synchronous* also if $c = 0$), and is *IS*, otherwise. For a sporadic task, all subtasks that belong to the same job will have equal offsets. Examples are given in insets (a) and (b) of Fig. 1. Informally, the restriction on offsets implies that the separation between any pair of subtask releases is at least the separation between those releases if the task were periodic.

[6] To reasonably curtail the discussion, we have refrained from explaining every concept in depth. The interested reader is referred to the primary sources cited.

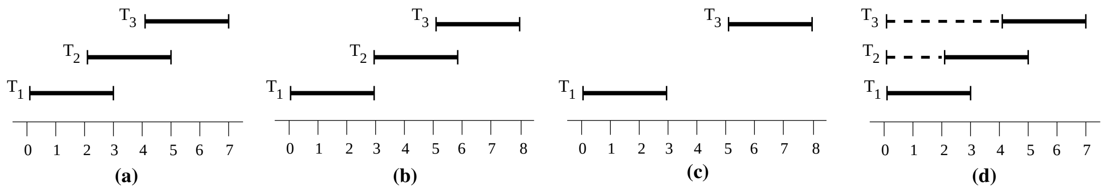

Figure 1. **(a)** PF-windows of the first job of a periodic (or sporadic) task T with weight $3/7$. This job consists of subtasks T_1, T_2, and T_3, each of which must be scheduled within its window. (This pattern repeats for every job.) **(b)** PF-windows of an IS task. Subtask T_2 is released one time unit late. Here, $\Theta(T_1) = 0$ while $\Theta(T_2) = \Theta(T_3) = 1$. **(c)** PF-windows of a GIS task. Subtask T_2 is absent and subtask T_3 is released one time unit late. **(d)** PF- and IS-windows of the first job of a GIS task with early releases. All the subtasks of this job are eligible when the job arrives. (The deadline-based priority definition of the Pfair scheduling algorithms and the prohibition of parallel execution of a task ensure that the subtasks execute in the correct sequence.) For each subtask, its PF-window consists of the solid part; the IS-window includes the dashed part, in addition. For example, T_2's PF-window is $[2, 5)$ and its IS-window is $[0, 5)$.

The interval $[r(T_i), d(T_i))$ is termed the *PF-window* of T_i and is denoted $\omega(T_i)$. The following lemma, concerning PF-window lengths, follows from (1).

Lemma 1 (Anderson and Srinivasan [6]) *The length of the PF-window of any subtask T_i of a task T, $|\omega(T_i)| = d(T_i) - r(T_i)$, is either $\left\lceil \frac{1}{wt(T)} \right\rceil$ or $\left\lceil \frac{1}{wt(T)} \right\rceil + 1$.*

GIS task model. When proving properties concerning Pfair scheduling algorithms, it is sometimes useful to "eliminate" certain subtasks. For example, if a deadline miss does not depend on the existence of a subtask, then ignoring such a subtask makes analysis easier. In [42], Srinivasan and Anderson introduced the *generalized intra-sporadic model* to facilitate such selective removal of subtasks. A GIS task system is obtained by omitting subtasks from a corresponding IS (or GIS) task system. However, the spacing between subtasks of a task that are not omitted may not be decreased in comparison to how they are spaced in a periodic task. Specifically, subtask T_i may be followed by subtask T_k, where $k > i + 1$ if the following holds: $r(T_k) - r(T_i)$ is at least $\left\lfloor \frac{k-1}{wt(T)} \right\rfloor - \left\lfloor \frac{i-1}{wt(T)} \right\rfloor$. That is, $r(T_k)$ is not smaller than what it would have been if $T_{i+1}, T_{i+2}, \ldots, T_{k-1}$ were present and released as early as possible. For the special case where T_k is the first subtask released by T, $r(T_k)$ must be at least $\left\lfloor \frac{k-1}{wt(T)} \right\rfloor$. Fig. 1(c) shows an example. In this example, though subtask T_2 is omitted, T_3 cannot be released before time 4. If a task T, after executing subtask T_i releases subtask T_k, then T_k is called the *successor* of T_i and T_i is called the *predecessor* of T_k. Note that a periodic task system is an IS task system, which in turn is a GIS task system; hence, any property established for the GIS task model applies to the other models, as well.

The early-release task model. The task models described so far are non-work-conserving in that, the second and later subtasks remain ineligible to be scheduled before their release times, even if they are otherwise ready and some processor is idle. The *early-release* (ER) task model is a work-conserving variant of the other models that allows subtasks to be scheduled before their release times [4]. Early releasing can be applied to subtasks in any of the task models considered so far, and unless otherwise specified, it should be assumed that early releasing is enabled. However, whether subtasks are actually released early is optional. To facilitate this, in this model, each subtask T_i has an eligibility time $\mathsf{e}(T_i)$ that specifies the first time slot in which T_i may be scheduled. This flexibility, in conjunction with the sporadic and the IS task models, which permit late job and subtask releases, can be used to schedule rate-based tasks. The interval $[e(T_i), d(T_i))$ is referred to as the *IS-window* of T_i. Fig. 1(d) gives an example. It is required that the following hold:

$$(\forall i \geq 1 :: e(T_i) \leq r(T_i) \ \wedge \ e(T_i) \leq e(T_{i+1})). \tag{2}$$

Concrete and non-concrete task systems. A task system is said to be *concrete* if the release and eligibility times are specified for each subtask of each task, and *non-concrete*, otherwise. Note that an infinite number of concrete task systems can be specified for every non-concrete task system. We indicate the type of the task system only when necessary.

Pfair and ERfair schedules. The notion of a Pfair schedule is obtained by comparing the allocations that each task receives in such a schedule to those received in an ideal fluid schedule. In an ideal fluid schedule, each task executes at a precise rate given by its utilization whenever it is active, and no deadlines, including pseudo-deadlines of subtasks, are missed. Let $\mathsf{A}(\mathsf{ideal}, T, t_1, t_2)$ and $\mathsf{A}(\mathcal{S}, T, t_1, t_2)$, denote the total allocation to T in the interval $[t_1, t_2)$ in the ideal schedule and an actual schedule, $\mathcal{S}$, respectively. Then, the "error" in allocation to T in $\mathcal{S}$ at time t with respect to the ideal schedule, referred to as the lag of T at t in $\mathcal{S}$, is given by

$$\mathsf{lag}(T, t, \mathcal{S}) = \mathsf{A}(\mathsf{ideal}, T, 0, t) - \mathsf{A}(\mathcal{S}, T, 0, t).$$

$\mathcal{S}$ is said to be *Pfair* iff the following holds.

$$(\forall t, T \in \tau :: -1 < \mathsf{lag}(T, t, \mathcal{S}) < 1)$$

Informally, each task's allocation error must always be less than one quantum. If early releases are allowed, then it is not required that the negative lag constraint, $\mathsf{lag}(T, t) > -1$, hold. A schedule for which $(\forall T, t : \mathsf{lag}(T, t) < 1)$ holds is said to be *ERfair*. The release times and deadlines in (1) are assigned such that scheduling each subtask by its deadline is sufficient to generate an ERfair schedule for τ; a Pfair schedule can be generated if each subtask is scheduled at or after its release time, as well. Further, ensuring that each task is scheduled in a Pfair or an ERfair manner is sufficient to ensure that the deadlines of all jobs are met in a periodic or sporadic task system. A schedule that is Pfair or ERfair exists for a GIS task system τ on M processors iff $\sum_{T \in \tau} wt(T) \leq M$ holds [15, 5].

If T is synchronous and periodic, then $\mathsf{A}(\mathsf{ideal}, T, 0, t)$ is given by $t \cdot wt(T)$. However, if T is GIS, then the allocation it receives in the ideal schedule may be less due to IS separations or omitted subtasks. To facilitate expressing $\mathsf{A}(\mathsf{ideal}, T, 0, t)$ for GIS tasks, let $\mathsf{A}(\mathsf{ideal}, T_i, 0, t)$ and $\mathsf{A}(\mathsf{ideal}, T_i, t)$ denote the ideal allocations to subtask T_i in $[0, t)$ and slot t, respectively. In [5], Anderson and Srinivasan showed that $\mathsf{A}(\mathsf{ideal}, T_i, t)$ is given by (3) below. An example is given in Fig. 2.

$$\mathsf{A}(\mathsf{ideal}, T_i, u) = \begin{cases} (\left\lfloor \frac{i-1}{wt(T)} \right\rfloor + 1) \times wt(T) - (i-1), & u = r(T_i) \\ i - (\left\lceil \frac{i}{wt(T)} \right\rceil - 1) \times wt(T), & u = d(T_i) - 1 \\ wt(T), & r(T_i) < u < d(T_i) - 1 \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

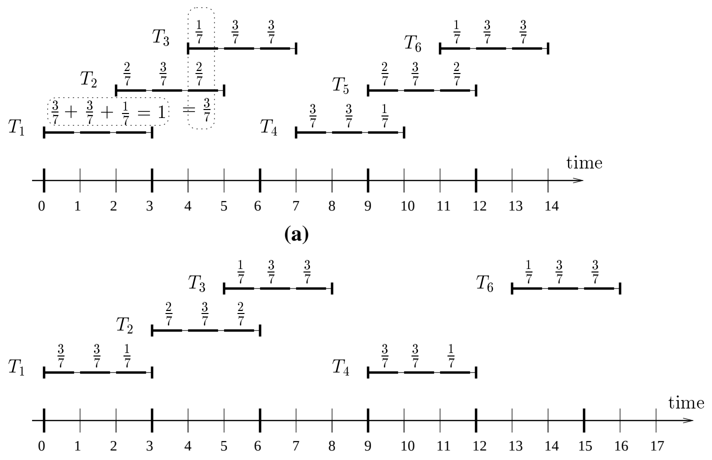

Figure 2. Per-slot ideal allocations to subtasks of a task T with weight $3/7$. These allocations are marked above the subtask windows. **(a)** T is synchronous, periodic. $\mathsf{A}(\mathsf{ideal}, T, t) = 3/7$ holds for every t. $\mathsf{A}(\mathsf{ideal}, T_2, 4) = \frac{2}{7}$ and $\mathsf{A}(\mathsf{ideal}, T_3, 4) = \frac{1}{7}$. **(b)** T is GIS. T_2's release is delayed by one time slot. T_4 is delayed by an additional time slot and T_5 is omitted. Here, $\mathsf{A}(\mathsf{ideal}, T_2, 4) = \frac{3}{7}$ and $\mathsf{A}(\mathsf{ideal}, T_3, 4) = 0$.

Let $\mathsf{A}(\mathsf{ideal}, T, t)$ denote the total allocation to task T in slot t. Then, $\mathsf{A}(\mathsf{ideal}, T, t)$ is given by $\sum_i \mathsf{A}(\mathsf{ideal}, T_i, t)$. For example, in Fig. 2(a), $\mathsf{A}(\mathsf{ideal}, T, 4) = \mathsf{A}(\mathsf{ideal}, T_2, 4) + \mathsf{A}(\mathsf{ideal}, T_3, 4) = 1/7 + 2/7 = 3/7$, since no subtasks other than T_2 and T_3 receive a non-zero allocation in slot 4.

As shown in Fig. 2, $\mathsf{A}(\mathsf{ideal}, T, u)$ usually equals $wt(T)$, but in certain slots, it may be less than $wt(T)$ due to omitted or delayed subtasks. Also, the total allocation that a subtask T_i receives in the slots that span its window is exactly one in the ideal schedule. These and similar properties have been proved formally in [41]. Later in this paper, we will use Lemma 2, and (4) and (5) given below.

$$(\forall u \geq 0 :: \mathsf{A}(\mathsf{ideal}, T, u) \leq wt(T)) \tag{4}$$

$$(\forall T_i :: \sum_{u=r(T_i)}^{d(T_i)-1} \mathsf{A}(\mathsf{ideal}, T_i, u) = 1) \tag{5}$$

Lemma 2 (A. Srinivasan [41]) *If $b(T_i) = 1$, then $\mathsf{A}(\mathsf{ideal}, T_{i+1}, r(T_{i+1})) \leq \rho(T)$, where $\rho(T) = \frac{T.e - \gcd(T.e, T.p)}{T.p}$.*

A task T's ideal allocation up to time t is simply

$$\mathsf{A}(\mathsf{ideal}, T, 0, t) = \sum_{u=0}^{t-1} \mathsf{A}(\mathsf{ideal}, T, u) = \sum_{u=0}^{t-1} \sum_i \mathsf{A}(\mathsf{ideal}, T_i, u),$$

and hence

$$\begin{aligned}
\mathsf{lag}(T, t, \mathcal{S}) &= \mathsf{A}(\mathsf{ideal}, T, 0, t) - \mathsf{A}(\mathcal{S}, T, 0, t) \\
&= \sum_{u=0}^{t-1} \mathsf{A}(\mathsf{ideal}, T, u) - \sum_{u=0}^{t-1} \mathcal{S}(T, u) \qquad (6) \\
&= \sum_{u=0}^{t-1} \sum_i \mathsf{A}(\mathsf{ideal}, T_i, u) - \sum_{u=0}^{t-1} \mathcal{S}(T, u). \qquad (7)
\end{aligned}$$

From (6), $\mathsf{lag}(T, t+1)$[7] is given by

$$\begin{aligned}\mathsf{lag}(T, t+1) &= \textstyle\sum_{u=0}^{t} (\mathsf{A}(\mathsf{ideal}, T, u) - \mathcal{S}(T, u)) \\ &= \mathsf{lag}(T, t) + \mathsf{A}(\mathsf{ideal}, T, t) - \mathcal{S}(T, t). \end{aligned} \tag{8}$$

Similarly, by (6) again, for any $0 \leq t' \leq t$,

$$\begin{aligned}\mathsf{lag}(T, t+1) &= \mathsf{lag}(T, t') + \textstyle\sum_{u=t'}^{t} (\mathsf{A}(\mathsf{ideal}, T, u) - \mathcal{S}(T, u)) \\ &= \mathsf{lag}(T, t') + \mathsf{A}(\mathsf{ideal}, T, t', t+1) - \mathsf{A}(\mathcal{S}, T, t', t+1). \end{aligned} \tag{9}$$

Another useful definition, the total lag for a task system τ in a schedule $\mathcal{S}$ at time t, $\mathsf{LAG}(\tau, t, \mathcal{S})$, or more concisely, $\mathsf{LAG}(\tau, t)$, is given by

$$\mathsf{LAG}(\tau, t) = \textstyle\sum_{T \in \tau} \mathsf{lag}(T, t). \tag{10}$$

Using (8)–(10), $\mathsf{LAG}(\tau, t+1)$ can be expressed as follows. In (12) below, $0 \leq t' \leq t$ holds.

$$\mathsf{LAG}(\tau, t+1) = \mathsf{LAG}(\tau, t) + \textstyle\sum_{T \in \tau} (\mathsf{A}(\mathsf{ideal}, T, t) - \mathcal{S}(T, t)) \tag{11}$$

$$\begin{aligned}\mathsf{LAG}(\tau, t+1) &= \mathsf{LAG}(\tau, t') + \sum_{u=t'}^{t} \sum_{T \in \tau} (\mathsf{A}(\mathsf{ideal}, T, u) - \mathsf{A}(\mathcal{S}, T, u)) \\ &= \mathsf{LAG}(\tau, t') + \mathsf{A}(\mathsf{ideal}, \tau, t', t+1) - \mathsf{A}(\mathcal{S}, \tau, t', t+1) \end{aligned} \tag{12}$$

(11) and (12) above can be rewritten as follows using (4).

$$\mathsf{LAG}(\tau, t+1) \leq \mathsf{LAG}(\tau, t) + \textstyle\sum_{T \in \tau} (wt(T) - \mathcal{S}(T, t)) \tag{13}$$

$$\mathsf{LAG}(\tau, t+1) \leq \mathsf{LAG}(\tau, t') + (t+1-t') \cdot \textstyle\sum_{T \in \tau} wt(T) - \mathsf{A}(\mathcal{S}, \tau, t', t+1) \tag{14}$$

$$= \mathsf{LAG}(\tau, t') + (t+1-t') \cdot \textstyle\sum_{T \in \tau} wt(T) - \sum_{u=t'}^{t} \sum_{T \in \tau} \mathcal{S}(T, u) \tag{15}$$

3.2 Tie-Break Parameters and Pfair Scheduling Algorithms

***b*-bit.** The *b-bit* or *boundary bit* is associated with each subtask T_i and is denoted $b(T_i)$. $b(T_i)$ is a tie-break parameter used by the known optimal Pfair algorithms and is defined by (16) below.

$$b(T_i) = \left\lceil \frac{i}{wt(T)} \right\rceil - \left\lfloor \frac{i}{wt(T)} \right\rfloor \tag{16}$$

From (1), it can be verified that if $\Theta(T_i) < \Theta(T_{i+1})$, then $d(T_i) \leq r(T_{i+1})$. Therefore, the PF-windows of T_i and T_{i+1} can overlap only if $\Theta(T_i) = \Theta(T_{i+1})$. It can also be verified that if $\Theta(T_i) = \Theta(T_{i+1})$, then $d(T_i) - r(T_{i+1}) = b(T_i)$. Hence, $b(T_i)$ determines whether the PF-window of T_i can overlap that of T_{i+1}. Observe that $b(T_i)$ is either zero or one. Therefore, the PF-windows of T_i and T_{i+1} are either disjoint or overlap by at most one slot. In Fig. 1, $b(T_2) = 1$, while $b(T_3) = 0$. Therefore, the PF-window of T_2 overlaps T_3's when $\Theta(T_3) = \Theta(T_2)$ as in insets (a), (b), and (d). Further, as shown in an appendix, the following lemma holds.

[7] The schedule parameter is omitted in the lag and LAG functions when unambiguous.

Lemma 3 *For all $i \geq 1$, $k \geq 1$, the following holds.*

$$r(T_{i+k}) \geq \begin{cases} d(T_i) + k - 1, & b(T_i) = 0 \\ d(T_i) + k - 2, & b(T_i) = 1 \end{cases}$$

Group deadline. Like the b-bit, the group deadline, denoted $D(T_i)$, is a parameter that is associated with each subtask T_i, and is used by some Pfair scheduling algorithms. For a subtask T_i of a heavy task T, $D(T_i)$ is defined as follows.

$$D(T_i) = \Theta(T_i) + \left\lceil \frac{\left\lceil \frac{i}{wt(T)} \right\rceil - i}{1 - wt(T)} \right\rceil$$

If $\Theta(T_i) = 0$, then $D(T_i) = \left\lceil \frac{d(T_i)-i}{1-wt(T)} \right\rceil$, and the formula has the interpretation that the group deadline of T_i is given by the deadline of the $(d(T_i) - i)^{\text{th}}$ subtask of a task with weight $1 - wt(T)$. The group deadline of each subtask of a light task is taken to be 0. The intuition behind group deadlines can be found in [6].

Optimal algorithms. Pfair scheduling algorithms function by choosing at the beginning of each time slot, at most M eligible subtasks for execution in that time slot. At present, three optimal Pfair scheduling algorithms, namely, PF [15], PD [17], and PD^2 [6, 42], that can correctly schedule any feasible GIS task system in polynomial time are known. All the three algorithms are based on a common approach: in each algorithm, subtask priorities are first determined on an *earliest-pseudo-deadline-first* basis; ties are then resolved using tie-breaking rules. Since PD^2 is the most efficient of the three algorithms, we limit ourselves to describing the priority definition of PD^2.

Algorithm PD^2. In determining the priorities of subtasks, PD^2 uses the b-bits and group deadlines of subtasks, apart from their pseudo-deadlines. Under PD^2, subtask T_i's priority is at least that of subtask U_j, denoted $T_i \preceq U_j$, if at least one of the following holds.

(i) $\mathsf{d}(T_i) < \mathsf{d}(U_j)$ **(ii)** $\mathsf{d}(T_i) = \mathsf{d}(U_j) \wedge \mathsf{b}(T_i) > \mathsf{b}(U_j)$ **(iii)** $\mathsf{d}(T_i) = \mathsf{d}(U_j) \wedge \mathsf{b}(T_i) = \mathsf{b}(U_j) = 1 \wedge \mathsf{D}(T_i) \geq \mathsf{D}(U_j)$

Any ties that may remain after applying the above rules can be resolved arbitrarily.

Overhead of tie-breaking rules. As explained in Sec. 1, in static task systems, the worst-case overhead incurred due to the use of tie-breaking rules is $2\mathcal{O}(M \log(N))$ additional comparison operations per time slot. The overhead incurred in addition in dynamic task systems was also explained in Sec. 1, and is illustrated using examples in Fig. 3.

Algorithm EPDF. EPDF is a derivative of the optimal algorithms in that it schedules subtasks on an earliest-pseudo-deadline-first basis but avoids using any tie-breaking rule. Under EPDF, ties among subtasks with equal deadlines are resolved arbitrarily. As mentioned earlier, in prior work, Srinivasan and Anderson have shown that EPDF is optimal on at most two processors [6]. They have also shown that on more than two processors, EPDF can correctly schedule task systems in which the maximum task weight is at most $1/(M-1)$ [44].

The next subsection presents some additional technical definitions and results that we will use in this paper.

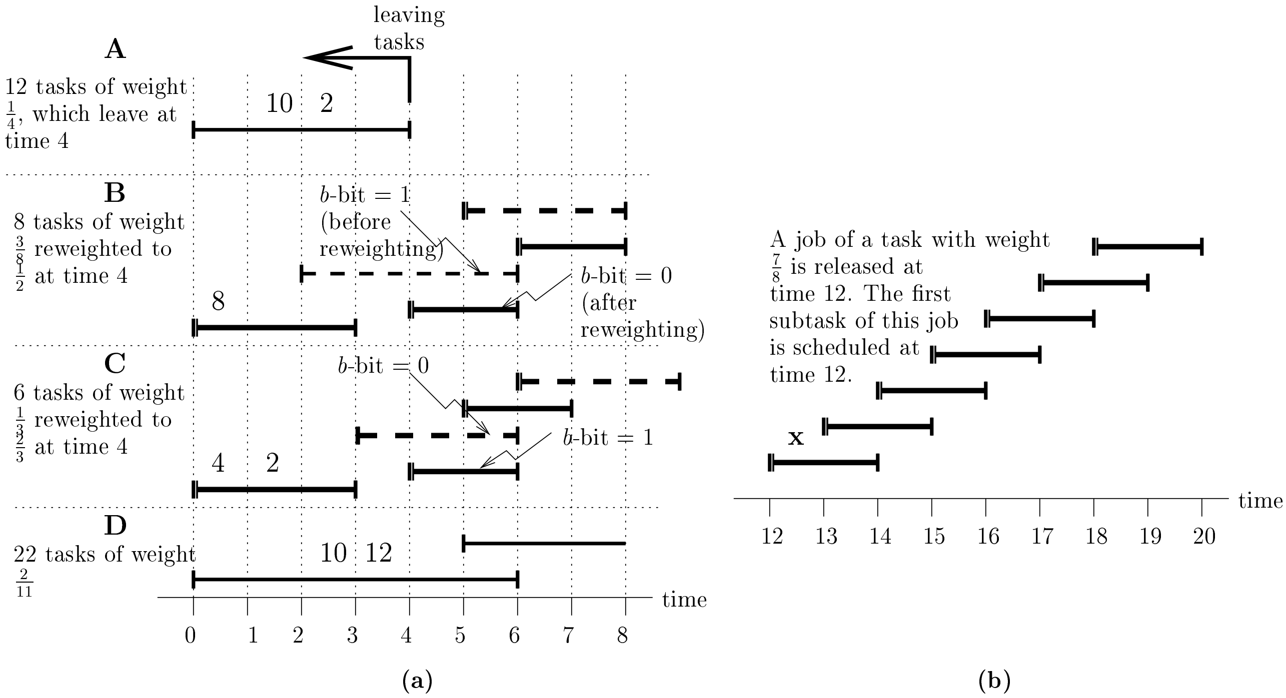

Figure 3. Examples to illustrate the complexities and overhead in reweighting due to the use of tie-breaking rules. **(a)** At time 4, tasks in Group A leave. The capacity that is made available is used to increase the weights of the tasks in Groups B and C. The last-released subtasks of the tasks in these groups have not been scheduled by time 4, and hence these tasks can be reweighted at this time. (The subtask windows before reweighting are shown using dashed lines.) For each task, this reweighting modifies the tie-break parameters for the next subtask to be scheduled, which is already in the ready queue. Note that, prior to reweighting, tasks in Group B have a higher PD^2 priority than those in Group C, whereas the reverse holds after the weight change. There is a similar change in the relative priority between subtasks in Groups B and D and between those in Groups C and D. Therefore, if tie-breaking rules are used, the position of a subtask whose tie-break parameters change may have to be adjusted in the priority queue at a cost of $\mathcal{O}(\log N)$. If tie-break parameters change for k subtasks, then the overhead is $\mathcal{O}(k \cdot (\log N))$, and if several subtasks are reweighted, as in this example, then the priority queue may have to be reconstructed at a cost of $\mathcal{O}(N)$. On the other hand, if no tie-breaking rule is used, then the priority queue need not be disturbed. In this example, deadlines are unaltered for the current (*i.e.*, active as in Def. 1) subtasks after the weight change. Altering the deadline for a subtask already in the priority queue can be avoided in one of two ways depending on whether the new deadline is before or after the old deadline. If earlier, the release time of the current subtask can be postponed (while leaving the eligibility time unaltered) so that the new and old deadlines align. If later, the reweighting request can be enacted after the current subtask is scheduled. **(b)** If tie-breaking rules are used, then because the first subtask depicted is scheduled at time 12, the task in this inset cannot leave or be reweighted until time 20, which is the group deadline for all the subtasks depicted. On the other hand, if no tie-breaking rule is used, then the task may leave at time 14 and the capacity allocated to it may be reclaimed. Similarly, the task may be reweighted at time 14 or 15 (depending on whether the reweighting request is initiated by the time the first subtask completes execution or later). (Refer to [44] and [19] for reweighting rules under optimal Pfair algorithms.)

3.3 Additional Technical Definitions

Active tasks. If subtasks are absent or are released late, then it is possible for a GIS (or IS) task to have no eligible subtasks and an allocation of zero during certain time slots. Tasks with and without subtasks in slot t are distinguished using the following definition of an *active* task.

Definition 1: A GIS task U is *active* in slot t if it has one or more subtasks U_j such that $e(U_j) \leq t < d(U_j)$. (A task that is active in t is not necessarily scheduled in that slot in an active schedule.)

Holes. If fewer than M tasks are scheduled at time t in $\mathcal{S}$, then one or more processors are idle in slot t. For each slot, each processor that is idle in that slot is referred to as a *hole*. Hence, if k processors are idle in slot t, then there are said to be k holes in t. The following lemma relates an increase in the total lag of τ, LAG, to the presence of holes, and is a generalization of one proved in [42].

Lemma 4 (Srinivasan and Anderson [42]) *If* $\mathsf{LAG}(\tau, t+\ell, \mathcal{S}) > \mathsf{LAG}(\tau, t, \mathcal{S})$, *where* $\ell \geq 1$, *then there is at least one hole in the interval* $[t, t+\ell)$.

Intuitively, if there is no idle processor in slots $t, \ldots, t+\ell-1$, then the total allocation in $\mathcal{S}$ in each of those slots to tasks in τ is equal to M. This is at least the total allocation that τ receives in any slot in the ideal schedule (assuming that the sum of the weights of all tasks in τ is at most M). Therefore, LAG cannot increase.

Task classification(from [42]). Tasks in τ may be classified as follows with respect to a schedule $\mathcal{S}$ and time interval $[t, t+\ell)$.[8]

$A(t, t+\ell)$**:** Set of all tasks that are scheduled in one or more slots in $[t, t+\ell)$.

$B(t, t+\ell)$**:** Set of all tasks that are not scheduled in any slot in $[t, t+\ell)$, but are active in one or more slots in the interval.

$I(t, t+\ell)$**:** Set of all tasks that are neither active nor scheduled in any slot in $[t, t+\ell)$.

As a shorthand, the notation $A(t)$, $B(t)$, and $I(t)$ is used when $\ell = 1$. $A(t, t+\ell)$, $B(t, t+\ell)$, and $I(t, t+\ell)$ form a partition of τ, *i.e.*, the following holds.

$$\left.\begin{array}{l} A(t,t+\ell) \cup B(t,t+\ell) \cup I(t,t+\ell) = \tau \qquad \wedge \\ A(t,t+\ell) \cap B(t,t+\ell) = B(t,t+\ell) \cap I(t,t+\ell) = I(t,t+\ell) \cap A(t,t+\ell) = \emptyset \end{array}\right\} \tag{17}$$

This classification of tasks is illustrated in Fig. 4(a) for $\ell = 1$. Using (10) and (17) above, we have the following.

$$\mathsf{LAG}(\tau, t+1) = \sum_{T \in A(t)} \mathsf{lag}(T, t+1) + \sum_{T \in B(t)} \mathsf{lag}(T, t+1) + \sum_{T \in I(t)} \mathsf{lag}(T, t+1) \tag{18}$$

The next definition identifies the last-released subtask at t of any task U.

Definition 2: Subtask U_j is the *critical subtask of* U *at* t iff $e(U_j) \leq t < d(U_j)$ holds, and no other subtask U_k of U, where $k > j$, satisfies $e(U_k) \leq t < d(U_k)$.

For example, in Fig. 4(a), T's critical subtask at both $t-1$ and t is T_{i+1}, and U's critical subtask at $t+1$ is U_{k+1}.

[8] For brevity, we let the task system τ and schedule $\mathcal{S}$ be implicit in these definitions.

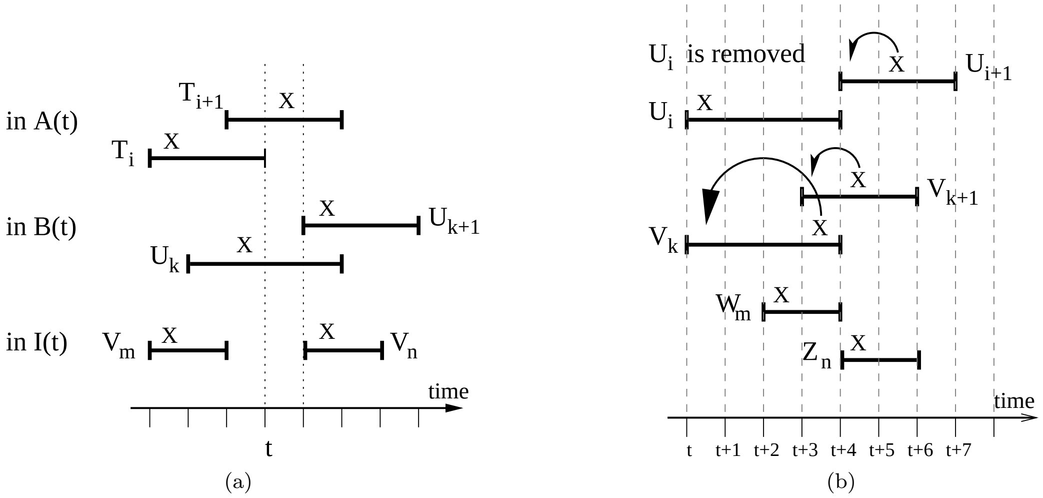

Figure 4. **(a)** Illustration of task classification at time t. IS-windows of two consecutive subtasks of three GIS tasks T, U, and V are depicted. The slot in which each subtask is scheduled is indicated by an "X." Because subtask T_{i+1} is scheduled at t, $T \in A(t)$. No subtask of U is scheduled at t. However, because the window of U_k overlaps slot t, U is active at t, and hence, $U \in B(t)$. Task V is neither scheduled nor active at t. Therefore, $V \in I(t)$. **(b)** Illustration of displacements. If U_i, scheduled at time t, is removed from the task system, then some subtask that is eligible at t, but scheduled later, can be scheduled at t. In this example, it is subtask V_k (scheduled at $t+3$). This displacement of V_k results in two more displacements, those of V_{k+1} and U_{i+1}, as shown. Thus, there are three displacements in all: $\Delta_1 = \langle U_i, t, V_k, t+3\rangle$, $\Delta_2 = \langle V_k, t+3, V_{k+1}, t+4\rangle$, and $\Delta_3 = \langle V_{k+1}, t+4, U_{i+1}, t+5\rangle$.

Displacements. To facilitate reasoning about Pfair algorithms, Srinivasan and Anderson formally defined displacements in [42]. Let τ be a GIS task system and let $\mathcal{S}$ be an EPDF schedule for τ. Then, removing a subtask, say T_i, from τ results in another GIS task system τ'. Suppose T_i is scheduled at t in $\mathcal{S}$. Then, T_i's removal can cause another subtask, say U_j, scheduled after t to shift left to t, which in turn can lead to other shifts, resulting in an EPDF schedule $\mathcal{S}'$ for τ'. Each shift that results due to a subtask removal is called a *displacement* and is denoted by a four-tuple $\langle X^{(1)}, t_1, X^{(2)}, t_2\rangle$, where $X^{(1)}$ and $X^{(2)}$ represent subtasks. This is equivalent to saying that subtask $X^{(2)}$ originally scheduled at t_2 in $\mathcal{S}$ displaces subtask $X^{(1)}$ scheduled at t_1 in $\mathcal{S}$. A displacement $\langle X^{(1)}, t_1, X^{(2)}, t_2\rangle$ is *valid* iff $e(X^{(2)}) \leq t_1$. Because there can be a cascade of shifts, there may be a chain of displacements. Such a chain is represented by a sequence of four-tuples. An example is given in Fig. 4(b).

The next two lemmas regarding displacements are proved in [41] and [42]. The first lemma states that in an EPDF schedule, a subtask removal can cause other subtasks to shift only to their left. According to the second lemma, if a subtask displaces to a slot with a hole, then its predecessor is scheduled in that slot prior to the displacement.

Lemma 5 (from [41]) *Let $X^{(1)}$ be a subtask that is removed from τ, and let the resulting chain of displacements in an* EPDF *schedule for τ be $\Delta_1, \Delta_2, \ldots, \Delta_k$, where $\Delta_i = \langle X^{(i)}, t_i, X^{(i+1)}, t_{i+1}\rangle$. Then $t_{i+1} > t_i$ for all $i \in [1, k]$.*

Lemma 6 (from [42]) *Let $\Delta = \langle X^{(i)}, t_i, X^{(i+1)}, t_{i+1}\rangle$ be a valid displacement in any* EPDF *schedule. If $t_i < t_{i+1}$ holds and there is a hole in slot t_i, then $X^{(i+1)}$ is the successor of $X^{(i)}$.*

4 Sufficient Schedulability Test for EPDF

In this section, we derive a schedulable utilization bound under EPDF for GIS task systems with early releases. The utilization bound that we derive is more accurate than that specified in Sec. 1, and can be used if the ρ values of tasks as defined in (19) can be computed.

Before getting started, we will set some needed notation in place and provide an overview of the approach used. Let ρ, $\rho_{\max}$, $W_{\max}$, and λ be defined as follows. (The task system τ will be omitted when it is unambiguous.)

$$\rho(T) \stackrel{\text{def}}{=} (T.e - \gcd(T.e, T.p))/T.p \tag{19}$$

$$\rho_{\max}(\tau) \stackrel{\text{def}}{=} \max_{T \in \tau}\{\rho(T)\} \tag{20}$$

$$W_{\max}(\tau) \stackrel{\text{def}}{=} \max_{T \in \tau}\{wt(T)\} \tag{21}$$

$$\lambda(\tau) \stackrel{\text{def}}{=} \max\left(2, \left\lceil \frac{1}{W_{\max}(\tau)} \right\rceil\right) \tag{22}$$

$\rho(T)$ is an upper bound on the ideal allocation that a subtask of T can receive in a slot that overlaps with the previous or the next subtask, such as, the allocation to T_2 in slot 2 or to T_3 in slot 4 in Fig. 2(a). This value is used in computing task lags, and hence, the LAG of a task system. $\lambda(\tau)$ denotes the minimum length of any window of any subtask of any task in τ excluding tasks with unit weight. As explained below, the minimum number of contiguous slots immediately preceding a deadline miss or an increase in LAG and that do not contain any hole is dependent on λ. Since our analysis relies on the number of such slots, the utilization bound derived is dependent on λ.

Overview of the derivation. In deriving a utilization bound for EPDF, we identify a condition that is necessary for a deadline miss, and determine a total task system utilization that is sufficient to ensure that this condition will not hold under any circumstance. We proceed as follows. If t_d denotes the time at which a deadline is missed by τ, then we note that $\mathsf{LAG}(\tau, t_d) \geq 1$ (Lemma 14(c)). While it is necessary that LAG be at least 1.0 for a deadline miss, this condition is be somewhat strong, and schedulability tests that are solely based on preventing it at all times turn out to be restrictive. A weaker condition is obtained by first noting that for a deadline to be missed at t_d, there should be no hole in any of the λ slots that immediately precede t_d (Lemma 14(g)).[9] This implies that the actual allocation exceeds the ideal by $M - U_{sum}$ in each of these slots, and hence, that LAG at t_d is at least 1.0 only if LAG at $t_d - \lambda$ is at least $\lambda \cdot (M - U_{sum}) + 1.0$. Thus, $\mathsf{LAG} \geq \lambda \cdot (M - U_{sum}) + 1.0$ is a weaker necessary condition for a deadline miss. (It should be pointed out that though weaker, this condition is nonetheless not sufficient, and hence, our utilization bound is neither worst-case nor optimal.)

We next determine a total utilization that is sufficient to ensure that LAG does not reach this value at any time. For this, we first make use of the fact that LAG can increase only across a slot with a hole (Lemma 4). Next, we note that for LAG to increase across a slot t, there should be no hole in any of the $\lambda - 1$ slots preceding it (Lemma 17), and express LAG at $t+1$ in terms of LAG at $t - \lambda + 1$, U_{sum}, and ρ (Lemma 19). This procedure is

[9] Lemma 14(g) only states that there is no hole in the last $\lambda - 1$ slots. Below, we explain how slot $t_d - \lambda$ is handled.

then applied recursively to express LAG at $t-\lambda+1$ in terms of LAG at an earlier time, and so on, with LAG at some time when it is less than 1.0 (such a time exists since LAG at time 0 is 0.0), serving as the base condition (Lemma 22). Thus, LAG at $t+1$ can be expressed in terms of just the task system parameters U_{sum}, $\rho_{\max}$, and λ, and solving for U_{sum} in $\mathsf{LAG}(t+1) < \lambda \cdot (M - U_{sum}) + 1$ yields a utilization bound.

In the above description, we have glossed over at least two complexities (and hence our description above may not *exactly* match what is stated in the lemmas referenced). First, if $W_{\max} = 1$, then it is possible for slot $(t_d - \lambda) = t_d - 2$ to have a hole, and hence, some extra reasoning (that is provided in Lemma 14(h) and in some lemmas in the appendix) is needed. (In fact, the reasoning used for task systems with $W_{\max} = 1$ can be generalized for $W_{\max} = \frac{1}{k}$, where k is an integer, to handle the case when slot $t_d - k - 1$ contains a hole. This generalization, along with some additional reasoning in Lemma 17, will help in expressing λ as $\left\lfloor \frac{1}{W_{\max}} \right\rfloor + 1$ instead of as in (22), and thereby an improved utilization bound can be provided when $W_{\max} = \frac{1}{k}$. Specifically, the drops in Fig. 7 will be at $W_{\max} = \frac{1}{k} + \epsilon$, instead of at $\frac{1}{k}$. However, formally presenting the generalization is quite tedious and will add to the length of the proof, so we have limited to presenting the general idea for $k = 1$.) Secondly, most properties discussed above apply only to minimal task systems, *i.e.*, a task system from which no subtask can be removed without causing a presumed deadline miss to be met. Minimal task systems are explained further later (following Def. 5).

With this overview, we will now move on to the derivation proper.

The utilization bound that we derive for EPDF is given by the following theorem.

Theorem 1 *Every GIS task system τ with $\sum_{T\in\tau} wt(T) \le \min\left(M, \frac{\lambda M(\lambda(1+\rho_{\max})-\rho_{\max})+1+\rho_{\max}}{\lambda^2(1+\rho_{\max})}\right)$, where $\rho_{\max}$ and λ are as defined in* (20) *and* (22)*, respectively, is correctly scheduled by* EPDF *on M processors.*

As a shorthand, we define $U(M,k,f)$ as follows.

Definition 3: $U(M,k,f) \stackrel{\text{def}}{=} \frac{kM(k(1+f)-f)+1+f}{k^2(1+f)}$.

To prove the above theorem, we use a setup similar to that used by Srinivasan and Anderson in [42] in establishing the optimality of PD^2. The overall strategy is to assume that the theorem to prove is false, identify a task system that is minimal or smallest in the sense of having the smallest number of subtasks and that misses a deadline, and show that such a task system cannot exist. (It should be pointed out that although the setup is similar, the details of the derivation and proof are significantly different.)

If Theorem 1 does not hold, then there exist a $W_{\max} \le 1$ and $\rho_{\max} < 1$, and a time t_d and a concrete task system τ defined as follows. (In these definitions, we assume that τ is scheduled on M processors.)

Definition 4: t_d is the earliest time at which any concrete task system with each task weight at most $W_{\max}$, $\rho(T)$ at most $\rho_{\max}$ for each task T, and total utilization at most $\min(M, U(M,\lambda,\rho_{max}))$ misses a deadline under EPDF, *i.e.*, some such task system misses a subtask deadline at t_d, and no such system misses a subtask deadline prior to t_d.

Definition 5: τ is a task system with the following properties.

(T0) τ is concrete task system with each task weight at most $W_{\max}$, $\rho(T)$ at most $\rho_{\max}$, and total utilization at most $\min(M, U(M, \lambda, \rho_{\max}))$.

(T1) A subtask in τ misses its deadline at t_d in $\mathcal{S}$, an EPDF schedule for τ.

(T2) No concrete task system satisfying (T0) and (T1) releases fewer subtasks in $[0, t_d)$ than τ.

(T3) No concrete task system satisfying, (T0), (T1), and (T2) has a larger rank than τ at t_d, where the *rank* of a task system τ at t is the sum of the eligibility times of all subtasks with deadlines at most t, *i.e.*, $rank(\tau, t) = \sum_{\{T_i: T \in \tau \ \wedge \ d(T_i) \leq t\}} e(T_i)$.

(T2) can be thought of as identifying a minimal task system in the sense of having a deadline miss with the fewest number of subtasks, subject to satisfying (T0) and (T1). It is easy to see that if Theorem 1 does not hold for all task systems satisfying (T0) and (T1), then some task system satisfying (T2) necessarily exists. (T3) further restricts the nature of τ by requiring subtask eligibility times to be spaced as much apart as possible. In proving some lemmas later, we will show that if the lemma does not hold, then either (T2) or (T3) is contradicted.

The following shorthand notation will be used hereafter.

Definition 6: α denotes the total utilization of τ, expressed as a fraction of M, *i.e.*, $\alpha \stackrel{\text{def}}{=} \frac{\sum_{T \in \tau} wt(T)}{M}$.

Definition 7: $\delta \stackrel{\text{def}}{=} \frac{\rho_{\max}}{1+\rho_{\max}}$.

The lemma below follows from (T0) and the definition of α.

Lemma 7 $0 \leq \alpha \leq \frac{\min(M, U(M, \lambda, \rho_{\max}))}{M} \leq 1$.

The next lemma is immediate from the definition of δ, (20), and (19). A proof is provided in an appendix.

Lemma 8 $0 \leq \delta < \frac{1}{2}$.

We now prove some properties about τ and $\mathcal{S}$. In proving some of these properties, we make use of the following three lemmas established in prior work by Srinivasan and Anderson.

Lemma 9 (Srinivasan and Anderson [42]) *If* $\mathsf{LAG}(\tau, t+1) > \mathsf{LAG}(\tau, t)$*, then* $B(t) \neq \emptyset$.

The following is an intuitive explanation for why Lemma 9 holds. Recall from Sec. 3 that $B(t)$ is the set of all tasks that are active but not scheduled at t. Because $e(T_i) \leq r(T_i)$ holds, by Definition 1 and (3), only tasks that are active at t may receive positive allocations in slot t in the ideal schedule. Therefore, if every task that is active at t is scheduled at t, then the total allocation in $\mathcal{S}$ cannot be less than the total allocation in the ideal schedule, and hence, by (11), LAG cannot increase across slot t.

Lemma 10 (Srinivasan and Anderson [42]) *Let* $t < t_d$ *be a slot with holes and let* $T \in B(t)$*. Then, the critical subtask at* t *of* T *is scheduled before* t.

To see that the above lemma holds, let T_i be the critical subtask of T at t. By its definition, the IS-window of T_i overlaps slot t, but T is not scheduled at t. Also, there is at least a hole in t. Because EPDF does not idle a processor while there is a task with an outstanding execution request, T_i is scheduled before t.

Lemma 11 (Srinivasan and Anderson [42]) *Let U_j be a subtask that is scheduled in slot t', where $t' \leq t < t_d$ and let there be a hole in t. Then $d(U_j) \leq t + 1$.*

This lemma is true because it can be shown that if $d(U_j) > t + 1$ holds, then U_j has no impact on the deadline miss at t_d. In other words, it can be shown that if the lemma does not hold, then the GIS task system obtained from τ by removing U_j also has a deadline miss at t_d, which is a contradiction to (T2).

Arguments similar to those used in proving the above lemma can be used to show the following. It is proved in an appendix.

Lemma 12 *Let $t < t_d$ be a slot with holes and let U_j be a subtask that is scheduled at t. Then $d(U_j) = t + 1$ and $b(U_j) = 1$.*

Finally, we will also use the following lemma, which is a generalization of Lemma 9. It is also proved in an appendix.

Lemma 13 *If $\mathsf{LAG}(\tau, t + 1) > \mathsf{LAG}(\tau, t - \ell)$, where $0 \leq \ell \leq \lambda - 2$ and $t \geq \ell$, then $B(t - \ell, t + 1) \neq \emptyset$.*

We are now ready to establish some properties concerning $\mathcal{S}$. These properties will be used later. Our ultimate goal is to show that one of Parts (c), (h), and (i) is contradicted, in turn contradicting the existence of τ, and thereby, proving Theorem 1.

Lemma 14 *The following properties hold for τ and $\mathcal{S}$.*

(a) *For all T_i, $d(T_i) \leq t_d$.*

(b) *Exactly one subtask of τ misses its deadline at t_d.*

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

(d) *Let T_i be a subtask that is scheduled at $t < t_d$ in $\mathcal{S}$. Then, $e(T_i) = \min(r(T_i), t)$.*

(e) $(\forall T_i :: d(T_i) < t_d \Rightarrow (\exists t :: e(T_i) \leq t < d(T_i) \wedge \mathcal{S}(T_i, t) = 1))$. *That is, every subtask with deadline before t_d is scheduled before its deadline.*

(f) *Let U_k be the subtask that misses its deadline at t_d. Then, U is not scheduled in any slot in $[t_d - \lambda + 1, t_d)$.*

(g) *There is no hole in any of the last $\lambda - 1$ slots,* i.e., *in slots $[t_d - \lambda + 1, t_d)$.*

(h) *There exists a time $v \leq t_d - \lambda$ such that the following both hold.*

(i) *There is no hole in any slot in $[v, t_d - \lambda)$.*

(ii) $\mathsf{LAG}(\tau, v) \geq (t_d - v)(1 - \alpha)M + 1$.

(i) *There exists a time $u \in [0, v)$, where v is as defined in part (h), such that $\mathsf{LAG}(\tau, u) < 1$ and $\mathsf{LAG}(\tau, t) \geq 1$ for all t in $[u + 1, v]$.*

The proofs of parts (a)–(d) are similar to those formally proved in [42] for the optimal PD^2 Pfair algorithm. We give informal explanations here. Part (e) follows directly from Definition 4 and (T1). The rest are proved in an appendix.

Proof of (a): This part holds because a subtask with deadline after t_d cannot impact the schedule for those with deadlines at most t_d. Therefore, even if all the subtasks with deadlines after t_d are removed, the deadline miss at t_d cannot be eliminated. In other words, the GIS task system that results due to the removal of subtasks from τ and that contains fewer subtasks than τ will also miss a deadline at t_d. This contradicts (T2).

Proof of (b): If several subtasks miss their deadlines at t_d, then even if all but one are removed, the remaining subtask will still miss its deadline, contradicting (T2).

Proof of (c): By part (a), all the subtasks in τ complete executing in the ideal schedule by t_d. Hence, the total allocation to τ in the ideal schedule up to t_d is exactly equal to the total number of subtasks in τ. By part (b), the total number of subtasks of τ scheduled in $\mathcal{S}$ in the same interval is fewer by exactly one subtask. Hence, the difference in allocations, $\mathsf{LAG}(\tau, t_d)$, is exactly one.

Proof of (d): Suppose $e(T_i)$ is not equal to $\min(r(T_i), t)$. Then, by (2) and because T_i is scheduled at t, it is before $\min(r(T_i), t)$. Simply changing $e(T_i)$ so that it equals $\min(r(T_i), t)$ will not affect how T_i or any other subtask is scheduled. So, the deadline miss at t_d will persist. However, this change increases the rank of the task system, and hence, (T3) is contradicted. ■

Overview of the rest of the proof of Theorem 1. If t_d and τ as given by Definitions 4 and 5, respectively, exist, then by Lemma 14(i), there exists a time slot $u < v$, where v is as defined in Lemma 14(h), across which LAG increases to at least one. To prove Theorem 1, we show that for every such u, either **(i)** there exists a time u', where $u+1 < u' \leq v$ or $u' = t_d$, such that $\mathsf{LAG}(\tau, u') < 1$, and thereby derive a contradiction to either Lemma 14(i) or Lemma 14(c), or **(ii)** v referred to above does not exist, contradicting Lemma 14(h). The crux of the argument we use in establishing the above is as follows. By Lemma 4, for LAG to increase across slot u, at least one hole is needed in that slot. As described earlier, we show that for every such slot, there are a sufficient number of slots without holes, and hence, that if (T0) holds, then the increase in LAG across slot u is offset by a commensurate decrease in LAG elsewhere that is sufficient to ensure that no deadline is missed.

In what follows, we state and prove several other lemmas that are required to accomplish this. We begin with a simple lemma that relates PF-window lengths to λ.

Lemma 15 *If $W_{\max} < 1$, then the length of the PF-window of each subtask of each task T in τ is at least λ; otherwise, it is at least $\lambda - 1$.*

Proof: Follows from the definition of λ in (22) and Lemma 1. ■

The next lemma shows that LAG does not increase across the first $\lambda - 1$ slots, *i.e.*, across slots $0, 1, \ldots, \lambda - 2$.

Lemma 16 $(\forall t : 0 \leq t \leq \lambda - 2 :: \mathsf{LAG}(\tau, t+1) \leq \mathsf{LAG}(\tau, t))$.

Proof: Contrary to the statement of the lemma, assume that $\mathsf{LAG}(\tau, t+1) > \mathsf{LAG}(\tau, t)$ for some $0 \leq t \leq \lambda - 2$. Then, by Lemma 4, there is at least one hole in slot t, and by Lemma 9, $B(t)$ is not empty. Let U be a task in $B(t)$ and U_j its critical subtask at t. Because there is a hole in t, by Lemma 10,

(B) U_j is scheduled before t,

and hence, by Lemma 11, $d(U_j) \leq t+1$ holds. We will consider two cases depending on $W_{\max}$. If $W_{\max} < 1$, then by Lemma 15, $|\omega(U_j)|$ (*i.e.*, the length of the PF-window of U_j) is at least λ, and hence, by Lemma 1, $r(U_j) = d(U_j) - |\omega(U_j)| \leq t + 1 - \lambda$ holds. By our assumption, $t < \lambda - 1$, and hence, $r(U_j) < 0$, which is impossible. Thus, the lemma holds when $W_{\max} < 1$. On the other hand, if $W_{\max} = 1$, then, by (22), $\lambda = 2$ holds, and hence, by our assumption, $t = 0$. Therefore, by (B), U_j is scheduled before time zero, which is also impossible. The lemma follows. ■

Lemma 4 showed that at least one hole is necessary in slot t for LAG to increase across t. The next lemma relates an increase in LAG to certain other conditions. We will use this lemma later to show that there is no hole in any of the $\lambda - 1$ slots preceding the one that we consider and across which LAG increases.

Lemma 17 *The following properties hold for* LAG *in* $\mathcal{S}$.

(a) *If* $\mathsf{LAG}(\tau, t+1, \mathcal{S}) > \mathsf{LAG}(\tau, t, \mathcal{S})$, *where* $\lambda - 1 \leq t < t_d - \lambda$, *then there is no hole in any slot in* $[t - \lambda + 2, t)$.

(b) *If* $\mathsf{LAG}(\tau, t+1, \mathcal{S}) > \mathsf{LAG}(\tau, t - \lambda + 2, \mathcal{S})$, *then there is no hole in slot* $t - \lambda + 1$.

Proof: We prove the two parts separately. We use Lemma 9 to conclude that $B(t)$ and $B(t - \lambda + 2, t+1)$ are non-empty in parts (a) and (b), respectively. For each part, we later show that unless the statement of that part holds, the critical subtask of any task in B can be removed without eliminating the deadline miss at t_d, which is a contradiction to (T2).

Proof of part (a). By the definition of λ in (22), $\lambda \geq 2$ holds. If $\lambda = 2$, then this part is vacuously true. Therefore, assume $\lambda \geq 3$, which, by (22), implies that

$$W_{\max} < \frac{1}{2}. \tag{23}$$

Because $\mathsf{LAG}(\tau, t+1) > \mathsf{LAG}(\tau, t)$, by Lemma 4,

(H) there is at least one hole in slot t,

and by Lemma 9, $B(t)$ is not empty. Let U be a task in $B(t)$ and let U_j be the critical subtask of U at t. By (H) and Lemma 10, U_j is scheduled before t. Let U_j be scheduled at $t' < t$, *i.e.*,

$$\mathcal{S}(U_j, t') = 1 \ \wedge \ t' < t. \tag{24}$$

Also, by Definition 2, $d(U_j) \geq t+1$ holds, while by (H), (24), and Lemma 11, $d(U_j) \leq t+1$ holds. Hence, we have

$$d(U_j) = t + 1. \tag{25}$$

By (23) and Lemma 15, the length of U_j's PF-window, $|\omega(U_j)| \geq \lambda$, and hence, by (25) and the definition of PF-window in Lemma 1, $r(U_j) \leq t - \lambda + 1$. We claim that U_j can be scheduled at ar after $t - \lambda + 2$. If there does not exist a predecessor for U_j, then U_j can be scheduled any time at or after $r(U_j)$, and hence, at or after $t - \lambda + 1$. If not, let U_h be the predecessor of U_j. Then, $h \leq j$, and hence, by Lemma 3, $d(U_h) \leq t - \lambda + 2$ holds. By Lemma 14(e), U_h does not miss its deadline, and hence, is scheduled at or before $t - \lambda + 1$. This implies that U_j is ready and can be scheduled at any time at or after $t - \lambda + 2$. Given this, if $t' > t - \lambda + 2$, then there is no hole in any slot in $[t - \lambda + 2, t')$. Hence, to complete the proof, it is sufficient to show that there is no hole in any slot in $[t', t)$. Assume to the contrary, and let v be the earliest slot with at least one hole in $[t', t)$. Then, by (24) and Lemma 11, $d(U_j) \leq v + 1 \leq t$, which contradicts (25). Thus, v does not exist, and there is no slot with one or more holes in $[t', t)$.

Proof of part (b). By the statement of this part of the lemma, $\mathsf{LAG}(\tau, t{+}1) > \mathsf{LAG}(\tau, t{-}\lambda{+}2)$. Hence, Lemma 13 applies with $\ell = \lambda - 2$. Therefore, $B(t - \lambda + 2, t + 1)$ is not empty. By definition, a task in $B(t - \lambda + 2, t + 1)$ is not scheduled anywhere in $[t - \lambda + 2, t + 1)$. Let U be a task in $B(t - \lambda + 2, t + 1)$ and let $t - \lambda + 2 \leq v \leq t$ be the latest slot in the interval $[t - \lambda + 2, t + 1)$ in which U is active, and let U_j be its critical subtask at v. Therefore, by Definition 2,

$$d(U_j) \geq v + 1 \geq t - \lambda + 3. \tag{26}$$

By Lemma 4, there is at least one hole in the interval $[t - \lambda + 2, t + 1)$. Hence, because U is in $B(t - \lambda + 2, t + 1)$, and thus is not scheduled anywhere in the interval concerned, U_j is scheduled at or before $t - \lambda + 1$ (as opposed to after the interval under consideration). Therefore, if there is a hole in $t - \lambda + 1$, then by Lemma 11, it follows that $d(U_j) \leq t - \lambda + 1$, which contradicts (26). Thus, there cannot be any hole in $t - \lambda + 1$, which completes the proof of this part. ■

The next lemma bounds the lag of each task at time $t + 1$, where t is a slot with one or more holes.

Lemma 18 *Let $t < t_d$ be a slot with one or more holes in $\mathcal{S}$. Then the following hold.*

(a) $(\forall T \in A(t) :: \mathsf{lag}(T, t + 1) \leq \rho(T))$

(b) $(\forall T \in B(t) :: \mathsf{lag}(T, t + 1) \leq 0)$

(c) $(\forall T \in I(t) :: \mathsf{lag}(T, t + 1) = 0)$

Proof: Parts (b) and (c) have been proved formally in [43]. To see why they hold, note that no task in $B(t)$ or $I(t)$ is scheduled at t. Because there is a hole in t, the critical subtask of a task in $B(t)$ is scheduled before t; similarly, the latest subtask of a task in $I(t)$ with release time at or before t should have completed execution by t. Hence, such tasks cannot be behind with respect to the ideal schedule.

Proof of part (a). Let T be a task in $A(t)$ and let T_i be its subtask scheduled at t. Then, in $\mathcal{S}$, T_i and all prior subtasks of T are scheduled in $[0, t + 1)$, *i.e.*, each of these subtasks receives its entire allocation of one quantum in $[0, t + 1)$. Because there is a hole in t, by Lemma 12, we have

$$d(T_i) = t + 1 \ \wedge \ b(T_i) = 1. \tag{27}$$

By (5) and the final part of (3), this implies that T_i and all prior subtasks of T receive a total allocation of one quantum each in $[0, t+1)$ in the ideal schedule also. Therefore, any difference in the allocations received in the ideal schedule and $\mathcal{S}$ is only due to subtasks that are released later than T_i. This is illustrated in Fig. 5.

Obviously, in $\mathcal{S}$, every subtask that is released later than T_i receives an allocation of zero in $[0, t+1)$. Therefore, the lag of T at $t+1$ is equal to the allocations that these later subtasks receive in the ideal schedule in the same interval. To determine this value, let T_j be the successor of T_i. By Lemma 3, if $j > i+1$, then $r(T_j) \geq d(T_i) = t+1$ holds. Hence, among the later subtasks, only T_{i+1} may receive a non-zero allocation in $[0, t+1)$. By (27), $b(T_i) = 1$, and hence, by Lemma 3 again, $r(T_{i+1})$ is at least $d(T_i) - 1$, which, by (27), is at least t. Hence, in the ideal schedule, the allocation to T_{i+1} in the interval $[0, t+1)$ may be non-zero only in slot $r(T_{i+1})$.

Thus, if $r(T_{i+1}) \geq t+1$ or T_{i+1} is absent, then $\mathsf{lag}(T, t+1)$ is zero. Otherwise, it is given by the allocation that T_{i+1} receives in slot $t = r(T_{i+1})$ in the ideal schedule. Because $b(T_i) = 1$ holds, by Lemma 2, $\mathsf{A}(\mathsf{ideal}, T_{i+1}, r(T_{i+1})) \leq \rho(T)$. Hence, $\mathsf{lag}(T, t+1) \leq \rho(T)$. This is illustrated in Fig. 5. ■

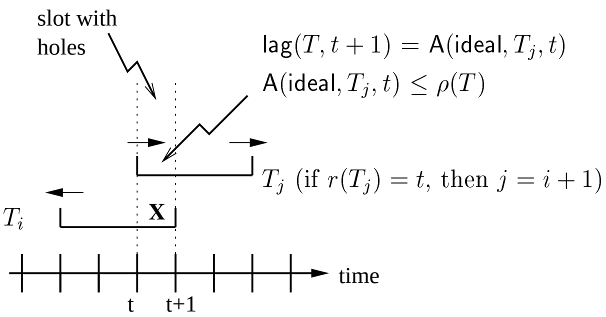

Figure 5. Lemma 18. PF windows of a subtask T_i and its successor T_j are shown. If $r(T_j) = t$, then $j = i+1$ holds. T_i is scheduled in slot t (indicated by an "X") and there are one or more holes in that slot. Arrows over the window end-points indicate that the end-point could extend along the direction of the arrow. T_i and all prior subtasks of T complete executing at or before $t+1$. Therefore, the lag of T at t is at most the allocation that T_{i+1} receives in slot $t+1$ in the ideal schedule.

The next lemma gives an upper bound on LAG at $t+1$ in terms of LAG at t and $t - \lambda + 1$, where t is a slot with holes.

Lemma 19 *Let t, where $\lambda - 1 \leq t < t_d - \lambda$, be a slot with at least one hole. Then, the following hold.* **(a)** $\mathsf{LAG}(\tau, t+1) \leq \mathsf{LAG}(\tau, t) \cdot \delta + \alpha M \cdot \delta$. **(b)** *If there is no hole in any slot in $[t - \lambda + 1, t)$, that is, there is no hole in any of the $\lambda - 1$ slots preceding t, then* $\mathsf{LAG}(\tau, t+1) \leq \mathsf{LAG}(\tau, t - \lambda + 1) \cdot \delta + (\lambda\alpha \cdot M - (\lambda - 1)M) \cdot \delta$.

Proof: By the statement of the lemma, there is at least one hole in slot t. Therefore, by Lemma 18, only tasks in set $A(t)$, *i.e.*, tasks that are scheduled in slot t, may have a positive lag at $t+1$. Let x denote the number of tasks scheduled at t, *i.e.*, $x = \sum_{T \in \tau} \mathcal{S}(T, t) = |A(t)|$. Then, by (18), we have

$$\begin{aligned}
\mathsf{LAG}(\tau, t+1) &\leq \textstyle\sum_{T \in A(t)} \mathsf{lag}(T, t+1) \\
&\leq \textstyle\sum_{T \in A(t)} \rho(T) && \text{(by Lemma 18)} \\
&\leq \textstyle\sum_{T \in A(t)} \rho_{\max} && \text{(by (20))} \\
&= |A(t)| \cdot \rho_{\max} \\
&= x \cdot \rho_{\max}. && (28)
\end{aligned}$$

Using (13), $\mathsf{LAG}(\tau, t+1)$ can be expressed as follows.

$$\mathsf{LAG}(\tau, t+1) \leq \mathsf{LAG}(\tau, t) + \textstyle\sum_{T \in \tau} (wt(T) - \mathcal{S}(T, t))$$

$$\begin{aligned}
&= \mathsf{LAG}(\tau, t) + \textstyle\sum_{T \in \tau} wt(T) - x && (\textstyle\sum_{T\in\tau} \mathcal{S}(T,t) = x) \\
&= \mathsf{LAG}(\tau, t) + \alpha M - x && \text{(by Def. 6)} \qquad (29)
\end{aligned}$$

By (28) and (29), we have

$$\mathsf{LAG}(\tau, t+1) \leq \min(x \cdot \rho_{\max}, \mathsf{LAG}(\tau, t) + \alpha M - x).$$

Because $\rho_{\max} \geq 0$, and hence, $x \cdot \rho_{\max}$ is non-decreasing with increasing x, whereas $\mathsf{LAG}(\tau,t) + \alpha M - x$ is decreasing, $\mathsf{LAG}(\tau, t+1)$ is maximized when $x \cdot \rho_{\max} = \mathsf{LAG}(\tau, t) + \alpha M - x$, *i.e.*, when $x = \mathsf{LAG}(\tau, t) \cdot \frac{1}{1+\rho_{\max}} + \alpha M \frac{1}{1+\rho_{\max}}$. Therefore, using either (28) or (29), we have

$$\begin{aligned}
\mathsf{LAG}(\tau, t+1) &\leq \mathsf{LAG}(\tau, t) \cdot \left(\tfrac{\rho_{\max}}{1+\rho_{\max}}\right) + \alpha M \cdot \left(\tfrac{\rho_{\max}}{1+\rho_{\max}}\right) \\
&= \mathsf{LAG}(\tau, t) \cdot \delta + \alpha M \cdot \delta. && \text{(by Def. 7)} \qquad (30)
\end{aligned}$$

The above establishes part (a).

By the statement of the lemma, $t \geq \lambda - 1$, and hence, $t - \lambda + 1 \geq 0$ holds. Therefore, using (15), $\mathsf{LAG}(\tau, t)$ can be expressed as follows.

$$\mathsf{LAG}(\tau, t) \leq \mathsf{LAG}(\tau, t-\lambda+1) + (\lambda - 1) \cdot \textstyle\sum_{T\in\tau} wt(T) - \sum_{u=t-\lambda+1}^{u=t-1} \sum_{T\in\tau} \mathcal{S}(T, u)$$

If there is no hole in any slot in $[t - \lambda + 1, t)$, then $\sum_{T\in\tau} \mathcal{S}(T, u) = M$, for all u in $[t - \lambda + 1, t)$. Hence,

$$\begin{aligned}
\mathsf{LAG}(\tau, t) &\leq \mathsf{LAG}(\tau, t-\lambda+1) + (\lambda - 1) \cdot \textstyle\sum_{T\in\tau} wt(T) - \sum_{u=t-\lambda+1}^{u=t-1} \sum_{T\in\tau} \mathcal{S}(T, u) \\
&= \mathsf{LAG}(\tau, t-\lambda+1) + (\lambda - 1) \cdot \textstyle\sum_{T\in\tau} wt(T) - (\lambda-1) \cdot M && \text{(there is no hole in } [t-\lambda+1, t) \text{ by assumption)} \\
&\leq \mathsf{LAG}(\tau, t-\lambda+1) + (\lambda - 1)\alpha \cdot M - (\lambda-1)M && \text{(by Def. 6)} \qquad (31)
\end{aligned}$$

Substituting (31) in (30), we have $\mathsf{LAG}(\tau, t+1) \leq \mathsf{LAG}(\tau, t-\lambda+1) \cdot \delta + (\lambda\alpha \cdot M - (\lambda-1)M) \cdot \delta$, establishing part (b). ■

The next two lemmas are purely mathematical and are proved in an appendix. They will be used in proving a later lemma.

Lemma 20 *A solution to the recurrence*

$$\begin{aligned}
L_0 &\leq \delta + \delta \cdot \alpha M \\
L_k &\leq \delta \cdot L_{k-1} + \delta \cdot (\lambda\alpha M - (\lambda-1)M),
\end{aligned}$$

for all $k \geq 1$ and $0 \leq \delta < 1$, is given by

$$L_n \leq \delta^{n+1}(1 + \alpha M) + (1 - \delta^n)\left(\frac{\delta}{1-\delta}\right)(\lambda\alpha M - (\lambda-1)M), \quad \textit{for all } n \geq 0.$$

Lemma 21 $\frac{M(\lambda-\delta-(\lambda-1)\delta^{n+1})+\delta^{n+2}-\delta^{n+1}+1-\delta}{M(\lambda-(\lambda-1)\delta^{n+1}-\delta^{n+2})} \geq \frac{M(\lambda-\delta)+\frac{1}{\lambda}}{\lambda M}$ *holds for all $n \geq 0, 0 \leq \delta \leq 1/2$, and $M \geq 1$.*

We will next show how to bound LAG at the end of an interval that does not contain λ consecutive slots without holes, after LAG increases to one at the beginning of the interval.

Lemma 22 *Let t, where $\lambda - 1 \leq t < t_d - \lambda$, be a slot across which* LAG *increases to at least one, i.e.,*

$$\lambda - 1 \leq t < t_d - \lambda \ \wedge \ \mathsf{LAG}(\tau, t) < 1 \ \wedge \ \mathsf{LAG}(\tau, t+1) \geq 1. \tag{32}$$

Let u, where $t < u \leq t_d - \lambda$, be such that there is at least one hole in at least one slot in every λ consecutive slots in the interval $[t+1, u)$. Then, $\mathsf{LAG}(\tau, u) < (\lambda - \lambda\alpha)M + 1$.

Proof: Because $\mathsf{LAG}(\tau, t+1) > \mathsf{LAG}(\tau, t)$ holds by (32), by Lemmas 4 and 17, we have (C1) and (C2) below, respectively. ((C2) holds by part (a) of Lemma 17 when $\lambda > 2$ and by part (b) of the same Lemma when $\lambda = 2$. By (22), $\lambda \geq 2$ holds.)

(C1) There is at least one hole in slot t.
(C2) There is no hole in slot $t - 1$.

By the statement of the lemma (that there is at least one hole in at least one slot in every λ consecutive slots in $[t+1, u)$) and (C1), there is at least one hole in at least one slot in every λ consecutive slots in $[t, u)$. Therefore, at most $\lambda - 1$ consecutive slots in the interval can be without any hole. Let

$$n \geq 0 \tag{33}$$

denote the number of slots in $[t, u - \lambda + 1)$ that each begin a block of $\lambda - 1$ consecutive slots with no holes. Note that the last slot in the last block is before $u - 1$. If $n > 0$, let $t_1, t_2, \ldots, t_n$, where $t_1 < t_2 < \ldots < t_n < u - \lambda + 1$ be such slots. Also note that if there is no hole in $[u - \lambda + 1, u)$ (provided $u - \lambda + 1 > t + 1$ holds), then there is at least one hole in $u - \lambda$. By (C1), $t_1 > t$ holds. Further, $t_n + \lambda - 1 < u$ and there is no hole in $[t_i, t_i + \lambda - 1)$ for each i. Because there is at least one hole in at least one slot in any consecutive λ slots in $[t, u)$, there is at least one hole in $t_i - 1$ for all $1 \leq i \leq n$. Similarly, there is at least one hole in $t_i + \lambda - 1$ for all $1 \leq i \leq n$. Also,

$$t_{i+1} - t_i \geq \lambda, \quad \text{for all } 1 \leq i \leq n - 1. \tag{34}$$

We divide the interval $[t - 1, u)$ into $n + 1$ non-overlapping subintervals that each start at t or at one of the n slots that begin blocks of $\lambda - 1$ consecutive slots without holes in $[t, u - \lambda + 1)$, namely, $t_1, \ldots, t_n$, as shown in Fig. 6. Note that $t - 1$ exists by the statement of the lemma, and by (C2), there is no hole in slot $t - 1$. Also, as noted earlier, $t_1 > t$.

The subintervals denoted $I_0, I_1, \ldots, I_n$ are defined as follows.

$$I_0 \stackrel{\text{def}}{=} \begin{cases} [t - 1, t_1), & \text{if } t_1 \text{ exists, } i.e., \ n > 0 \\ [t - 1, u), & \text{otherwise} \end{cases} \tag{35}$$

$$I_n \stackrel{\text{def}}{=} [t_n, u), \qquad \text{if } n > 0 \tag{36}$$

$$I_k \stackrel{\text{def}}{=} [t_k, t_{k+1}), \quad \text{for all } 1 \leq k < n \tag{37}$$

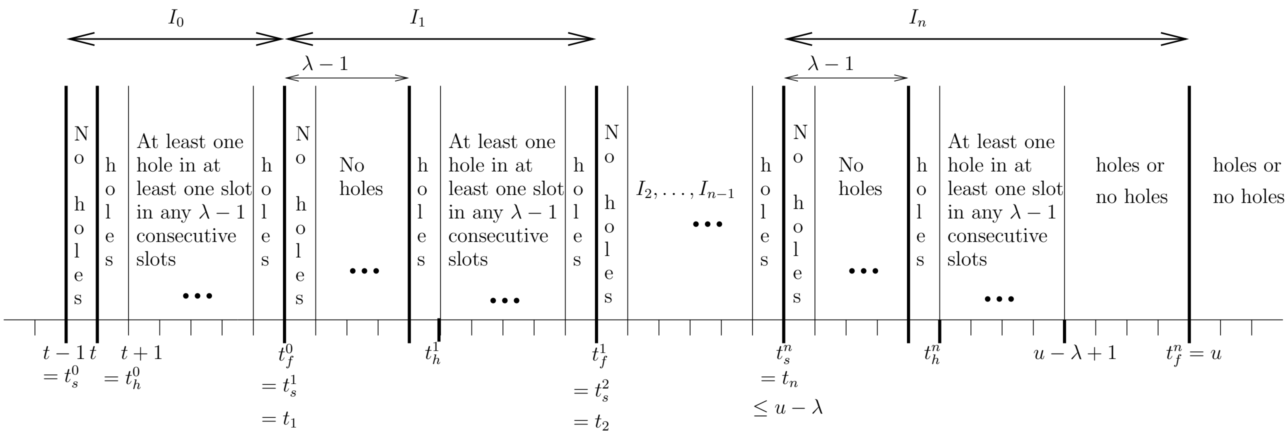

Figure 6. Lemma 22. $\mathsf{LAG}(\tau, t) < 1$ and $\mathsf{LAG}(\tau, t+1) \geq 1$. There is at least one hole in at least one slot in every λ consecutive slots in $[t-1, u)$. Each interval I_k, where $1 \leq k \leq n$, has one set of $\lambda - 1$ consecutive slots without holes. The objective is to determine an upper bound on $\mathsf{LAG}(\tau, u)$.

Before proceeding further, we introduce some more notation. We denote the start and end times of I_k, where $0 \leq k \leq n$, by t_s^k and t_f^k, respectively, *i.e.*, I_k is denoted as follows.

$$I_k \stackrel{\text{def}}{=} [t_s^k, t_f^k), \qquad \text{for all } k = 0, 1, \ldots, n \tag{38}$$

t_h^k is defined as follows for $0 \leq k \leq n$.

$$t_h^k \stackrel{\text{def}}{=} \begin{cases} t+1, & k = 0 \\ t_s^k + \lambda, & \text{otherwise} \end{cases} \tag{39}$$

LAG at t_h^k is denoted L_k, *i.e.*,

$$L_k \stackrel{\text{def}}{=} \mathsf{LAG}(\tau, t_h^k), \qquad \text{for all } k = 0, 1, \ldots, n \tag{40}$$

Note that each subinterval I_k, where $1 \leq k \leq n$, contains a block of $\lambda - 1$ consecutive slots with no holes that begins at t_s^k, I_n may contain a second such block beginning at $u - \lambda + 1$, and the following holds (refer to Fig. 6).

(C3) For all k, $0 \leq k \leq n$, there is **(i)** no hole in any slot in $[t_s^k, t_h^k - 1)$, **(ii)** at least one hole in $t_h^k - 1$, and **(iii)** at least one hole in at least one slot in $[\hat{t}, \hat{t} + \lambda - 1)$, for all $\hat{t}$, where $t_h^k - 1 \leq \hat{t} \leq \min(t_f^k - 1, u - \min(\lambda - 1, t_f^n - t_h^n + 1))$.

Our goal now is to derive bounds for LAG at t_f^k, for all $0 \leq k \leq n$. Towards this end, we first establish the following claim, which states that LAG at any time in the interval $[t_h^k, t_f^k]$ is at most L_k for all k. Intuitively, this claim follows from the following: **(i)** there is at least one hole in $t_h^k - 1$ and **(ii)** at most $\lambda - 2$ consecutive slots are without holes in $[t_h^k, \min(t_f^k, u - \lambda + 1))$. Due to the above facts, it follows that for every slot with a hole in $[t_h^k, t_f^k)$, there is at least one hole in at least one of the $\lambda - 1$ slots that precede it, which is used with part (a) or (b) of Lemma 17 to arrive at the required claim.

Claim 1 $(\forall k : 0 \leq k \leq n :: (\forall t' : t_h^k \leq t' \leq t_f^k :: \mathsf{LAG}(\tau, t') \leq L_k))$.

For each k, the proof is by induction on t'.

Base Case: $t' = t_h^k$**.** The claim holds by the definition in (40).

Induction Step: Assuming that the claim holds at all times in the interval $[t_h^k, t']$, where $t_h^k \leq t' < t_f^k$, we show that it holds at $t'+1$. By this induction hypothesis, we have the following.

$$t_h^k \leq t' < t_f^k \tag{41}$$

$$(\forall \hat{t} : t_h^k \leq \hat{t} \leq t' :: \mathsf{LAG}(\tau, \hat{t}) \leq L_k) \tag{42}$$

We consider the following two cases. In each case, if there is no hole in t', then by Lemma 4, $\mathsf{LAG}(\tau, t'+1) \leq \mathsf{LAG}(\tau, t') \leq L_k$. Hence, assume that there is a hole in t'.

Case 1: $\boldsymbol{t' \leq t_h^k + \lambda - 3}$. For this case, $t' - \lambda + 2 \leq t_h^k - 1$ holds. By (C3)(ii), there is a hole in slot $t_h^k - 1$. By (41), $t_h^k - 1 < t'$. Thus, there is at least one hole in the interval $[t' - \lambda + 2, t')$, *i.e.*, in one of the $\lambda - 2$ slots immediately preceding t'. Therefore, by the contrapositive of Lemma 17(a), $\mathsf{LAG}(\tau, t'+1) \leq \mathsf{LAG}(\tau, t')$, which by (42), is at most L_k.

Case 2: $\boldsymbol{t' > t_h^k + \lambda - 3}$. In this case,

$$t_h^k - 2 < t' - \lambda + 1 \tag{43}$$

holds. Because $t_f^k \leq t_f^n$ for all $0 \leq k \leq n$, we have $t_f^k \leq u$ by (35) and (36), when $n = 0$ and $n > 0$, respectively, and hence, by (41), we have $t' < u$. Therefore, $t' - \lambda + 1 < u - \lambda + 1 \leq u - \min(\lambda - 1, t_f^n - t_h^n + 1)$ holds. Also, because $\lambda \geq 2$, by (41), $t' - \lambda + 1 < t_f^k - 1$ follows. Thus, we have $t' - \lambda + 1 < \min(t_f^k - 1, u - \min(\lambda - 1, t_f^n - t_h^n + 1))$. Therefore, by (43), (C3)(iii) applies for $\hat{t} = t' - \lambda + 1$, and it follows that there is at least one hole in $[t' - \lambda + 1, t')$. If there is a hole in $[t' - \lambda + 2, t')$, then by the contrapositive of Lemma 17(a), $\mathsf{LAG}(\tau, t'+1) \leq \mathsf{LAG}(\tau, t')$, which by (42), is at most L_k. On the other hand, if there is no hole in $[t' - \lambda + 2, t')$, then there is a hole in $t' - \lambda + 1$ because there is at least one hole in $[t' - \lambda + 1, t')$. Thus, by the contrapositive of Lemma 17(b), $\mathsf{LAG}(\tau, t'+1) \leq \mathsf{LAG}(\tau, t' - \lambda + 2)$. By (43), $t' - \lambda + 2 \geq t_h^k$, and hence, by (42), $\mathsf{LAG}(\tau, t' - \lambda + 2) \leq L_k$. The claim follows for this case. ■

Having shown that $\mathsf{LAG}(\tau, t_f^k)$ is at most L_k, we now bound L_k. We start by determining a bound for L_0. By (40), $L_0 = \mathsf{LAG}(\tau, t_h^0)$, which by (39) is $\mathsf{LAG}(\tau, t+1)$. Thus,

$$\begin{aligned} L_0 &= \mathsf{LAG}(\tau, t+1) \\ &\leq \delta \cdot \mathsf{LAG}(\tau, t) + \delta \cdot \alpha M && \text{(by Lemma 19(a), because (C1) holds and by the statement of this lemma, } \lambda - 1 \leq t < t_d - \lambda) \\ &< \delta + \delta \cdot \alpha M, && (\mathsf{LAG}(\tau, t) < 1 \text{ by (32)).} \end{aligned} \tag{44}$$

We next determine an upper bound for L_k, where $1 \le k \le n$. Notice that by our definition of I_k in (37), we have $t_s^k = t_f^{k-1}$. Thus, $\mathsf{LAG}(\tau, t_s^k) = \mathsf{LAG}(\tau, t_f^{k-1})$, and hence, by Claim 1, we have

$$\mathsf{LAG}(\tau, t_s^k) \le L_{k-1}. \tag{45}$$

By (39), for $k > 0$, $t_h^k = t_s^k + \lambda$. By (34) (and because $t_i = t_s^i$) $t_s^k + \lambda - 1 < t_s^{k+1}$. Also, $t_s^{k+1} \le u$ holds by (37) and (36). By the statement of the lemma $u \le t_d - \lambda$. Thus,

$$t_s^k + \lambda - 1 < t_d - \lambda. \tag{46}$$

Also, $t_s^1 = t_1 > t$, and hence,

$$t_s^k + \lambda - 1 > t \ge \lambda - 1, \tag{47}$$

where the last inequality is from the statement of the lemma. By (C3)(ii), there is a hole in slot $t_h^k - 1 = t_s^k + \lambda - 1$, and by (C3)(i), no hole in any slot in $[t_s^k, t_h^k - 1)$, *i.e.*, in $[t_s^k, t_s^k + \lambda - 1)$. Therefore, because (46) and (47) also hold, Lemma 19(b) applies with $t_s^k + \lambda - 1$, and hence, $\mathsf{LAG}(\tau, t_s^k + \lambda) \le \delta \cdot \mathsf{LAG}(\tau, t_s^k) + \delta \cdot (\lambda\alpha \cdot M - (\lambda - 1) \cdot M)$, which by (40), (45), and $t_h^k = t_s^k + \lambda$ implies that

$$L_k = \mathsf{LAG}(\tau, t_h^k) = \mathsf{LAG}(\tau, t_s^k + \lambda) \le \delta \cdot L_{k-1} + \delta \cdot (\lambda\alpha \cdot M - (\lambda - 1) \cdot M), \qquad \text{for all } 1 \le k \le n. \tag{48}$$

By (36) and (38), we have $u = t_f^n$. Therefore, by Claim 1 and (33), we have

$$\mathsf{LAG}(\tau, u) = \mathsf{LAG}(\tau, t_f^n) \le L_n, \tag{49}$$

and hence, an upper bound on $\mathsf{LAG}(\tau, u)$ can be determined by solving the recurrence given by (44) and (48), which is restated below for convenience.

$$\begin{aligned} L_0 &< \delta + \delta \cdot \alpha M \\ L_k &\le \delta \cdot L_{k-1} + \delta \cdot (\lambda\alpha \cdot M - (\lambda - 1) \cdot M) \quad , k \ge 1 \end{aligned}$$

By Lemma 20, a solution to the above recurrence is given by

$$L_k < \delta^{k+1}(1 + \alpha M) + (1 - \delta^k)\left(\frac{\delta}{1 - \delta}\right)(\lambda\alpha \cdot M - (\lambda - 1) \cdot M).$$

Therefore, by (49), $\mathsf{LAG}(\tau, u) \le L_n < \delta^{n+1}(1 + \alpha M) + (1 - \delta^n)\left(\frac{\delta}{1-\delta}\right)(\lambda\alpha \cdot M - (\lambda - 1) \cdot M)$.

If L_n is at least $(\lambda - \lambda\alpha)M + 1$, then $\delta^{n+1}(1 + \alpha M) + (1 - \delta^n)\left(\frac{\delta}{1-\delta}\right)(\lambda\alpha \cdot M - (\lambda - 1) \cdot M) > (\lambda - \lambda\alpha)M + 1$, which on rearranging terms implies that

$$\begin{aligned} \alpha &> \frac{M(\lambda - \delta - (\lambda - 1)\delta^{n+1}) + \delta^{n+2} - \delta^{n+1} + 1 - \delta}{M(\lambda - (\lambda - 1)\delta^{n+1} - \delta^{n+2})} \\ &\ge \frac{M(\lambda - \delta) + \frac{1}{\lambda}}{\lambda M} && \text{(by Lemma 21, because } 0 \le \delta < 1/2 \text{ by Lemma 8)} \\ &= \frac{\lambda M(\lambda(1 + \rho_{\max}) - \rho_{\max}) + 1 + \rho_{\max}}{\lambda^2 M(1 + \rho_{\max})} && \text{(by Def. 7)} \\ &= \frac{U(M, \lambda, \rho_{\max})}{M} && \text{(by Def. 3)} \\ &\ge \frac{\min(M, U(M, \lambda, \rho_{\max}))}{M}. \end{aligned} \tag{50}$$

Because (50) is in contradiction to Lemma 7, we conclude that $L_n < (\lambda - \lambda\alpha)M + 1$. Hence, by (49), $\mathsf{LAG}(\tau, u) \leq L_n < (\lambda - \lambda\alpha)M + 1$. ■

Lemma 23 *Let $\lambda - 1 \leq t < t_d - \lambda$ be a slot such that $\mathsf{LAG}(\tau, t) < 1 \;\wedge\; \mathsf{LAG}(\tau, t+1) \geq 1$ and let u be the earliest time after t such that $u = t_d - \lambda$ or there is no hole in any slot in the interval $[u, u + \lambda)$. (Note that this implies that there is at least one hole in at least one slot in every λ consecutive slots in $[t+1, u)$.) Then, at least one of the following holds.*

$$t < u \leq t_d - 2\lambda, \text{ there is a hole in } u-1, (\forall \hat{t} : u \leq \hat{t} < u+\lambda :: \mathsf{LAG}(\tau, \hat{t}) < (\lambda - \lambda\alpha)M + 1), \text{ and } \mathsf{LAG}(\tau, u+\lambda) < 1. \quad (51)$$

$$\mathsf{LAG}(\tau, t_d) < 1. \quad (52)$$

$$\text{There exists at least one slot with a hole before } t_d - \lambda \text{ and } (\forall v : t' + 1 \leq v \leq t_d - \lambda :: \mathsf{LAG}(\tau, v) < (t_d - v) \cdot (1-\alpha)M + 1), \text{ where } t' \text{ is the latest slot with a hole before } t_d - \lambda. \quad (53)$$

Proof: Because $\mathsf{LAG}(\tau, t+1) > \mathsf{LAG}(\tau, t)$ holds (by the statement of the lemma), by Lemma 4, we have the following.

(D1) There is at least one hole in t.

We consider two cases depending on u. (By the definition of u in the statement of the lemma, if $u < t_d - \lambda$ holds, then there is no hole in u.)

Case 1: $u \leq t_d - \lambda$ and there is no hole in u. By Lemma 14(g),

(D2) there is no hole in any slot in $[t_d - \lambda + 1, t_d)$.

If $u = t_d - \lambda$, then by the condition of this case, there is no hole in $t_d - \lambda$, and hence, by (D2), in any slot in $[t_d - \lambda, t_d)$, *i.e.*, in $[u, u+\lambda)$. On the other hand, if $u < t_d - \lambda$, then by the definition of u, there is no hole in $[u, u+\lambda)$. Thus, in both cases, we have the following.

(D3) There is no hole in any slot in $[u, u+\lambda)$.

By the definition of u, there is at least one hole in at least one slot in any λ consecutive slots in the interval $[t+1, u)$. Therefore, by Lemma 22, we have

$$\mathsf{LAG}(\tau, u) < (\lambda - \lambda\alpha)M + 1. \quad (54)$$

To prove the lemma for this case, we consider the following two subcases.

Subcase 1(a): $u \leq t_d - 2\lambda$. For this subcase, we first show that

(D4) there is at least one hole in $u - 1$.

Because (D3) and $u < t_d - \lambda$ (which is implied by the condition of this subcase, since $\lambda \geq 2$) hold, the absence of holes in $u - 1$ would imply that there is no hole in any slot in $[u-1, u+\lambda)$. This is a contradiction of the fact that

u is the earliest time after t such that either there is no hole in any slot in $[u, u+\lambda)$ or $u = t_d - \lambda$. Thus, there is at least one hole in $u - 1$.

We next show that $\mathsf{LAG}(\tau, u+\lambda) < 1$ holds. By (15), we have

$$\begin{array}{rcll} \mathsf{LAG}(\tau, u+\lambda) & \leq & \mathsf{LAG}(\tau, u) + \lambda \cdot \sum_{T\in\tau} wt(T) - \sum_{v=u}^{v=u+\lambda-1} \sum_{T\in\tau} \mathcal{S}(T, v) & \\ & = & \mathsf{LAG}(\tau, u) + \lambda \cdot \sum_{T\in\tau} wt(T) - \lambda M & \text{(by (D3), there is no hole in } [u, u+\lambda)) \\ & < & (\lambda - \lambda\alpha)M + 1 + \lambda \cdot \sum_{T\in\tau} wt(T) - \lambda M & \text{(by (54), } \mathsf{LAG}(\tau, u) < (\lambda - \lambda\alpha)M + 1) \\ & = & (\lambda - \lambda\alpha)M + 1 + \lambda\alpha M - \lambda M & \text{(by Def. 6)} \\ & = & 1. & \end{array} \quad (55)$$

Further, by (D3), there is no hole in any slot in $[u, u+\lambda)$, hence, by Lemma 4, LAG cannot increase across any slot in this interval. Therefore, by (54), $(\forall \hat{t} : u < \hat{t} < u+\lambda :: \mathsf{LAG}(\tau, \hat{t}) \leq \mathsf{LAG}(\tau, u) < (\lambda - \lambda\alpha)M + 1)$ holds, which together with (D4), (54) and (55), establishes condition (51) for this subcase.

Subcase 1(b): $\boldsymbol{t_d - 2\lambda < u \leq t_d - \lambda}$**.** For this subcase, we first show that there is no hole in any slot in $[u, t_d)$. If $u = t_d - \lambda$, then, by (D3), there is no hole in $[u, t_d)$. Otherwise, because $u > t_d - 2\lambda$ holds, we have $u + \lambda \geq t_d - \lambda + 1$. Thus, by (D3), there is no hole in any slot in $[u, t_d - \lambda + 1)$, and hence, by (D2), no hole in $[u, t_d)$. By (14), $\mathsf{LAG}(\tau, t_d) \leq \mathsf{LAG}(\tau, u) + (t_d - u) \cdot \sum_{T\in\tau} wt(T) - \sum_{v=u}^{t_d-1} \mathsf{A}(\mathcal{S}, \tau, v)$. Therefore, by Definition 6, $\mathsf{LAG}(\tau, t_d) \leq \mathsf{LAG}(\tau, u) + (t_d - u)\alpha M - \sum_{v=u}^{t_d-1} \mathsf{A}(\mathcal{S}, \tau, v)$. Since there is no hole in $[u, t_d)$ in $\mathcal{S}$, $\mathsf{A}(\mathcal{S}, \tau, v) = M$, for every v in $[u, t_d)$. Hence, $\mathsf{LAG}(\tau, t_d) \leq \mathsf{LAG}(\tau, u) + (t_d - u)(\alpha M - M)$, which by (54), implies $\mathsf{LAG}(\tau, t_d) < (\lambda - \lambda\alpha)M + 1 + (t_d - u)(\alpha M - M)$. Since, by the condition of this case, $u \leq t_d - \lambda$ holds, we have $t_d - u \geq \lambda$. Hence, because $\alpha \leq 1$ (by Lemma 7), $\mathsf{LAG}(\tau, t_d) < 1$ holds. Thus, (52) holds for this subcase.

Case 2: $\boldsymbol{u = t_d - \lambda}$ **and there is a hole in slot** $\boldsymbol{t_d - \lambda}$**.** Let u' denote the latest slot with at least one hole in $[t, t_d - \lambda)$. Because (D1) holds, u' exists, and the following holds.

> **(D5)** $u' < t_d - \lambda$ and there is no hole in any slot in $[u'+1, t_d - \lambda)$.

Because no λ consecutive slots in $[t, t_d - \lambda)$ are without holes (implied by the statement of the lemma, since $u = t_d - \lambda$ for this case), applying Lemma 22 (with $u = u' + 1$), we have

$$\mathsf{LAG}(\tau, u'+1) < \lambda(1-\alpha)M + 1. \quad (56)$$

By (D5), $u' < t_d - \lambda$, and hence, $u' + 1 \leq t_d - \lambda$ holds. Therefore, $\lambda \leq (t_d - (u'+1))$, and hence, because $\alpha \leq 1$ by Lemma 7,

$$\lambda(1-\alpha)M \leq (t_d - (u'+1))(1-\alpha)M \quad (57)$$

holds. Therefore, by (56), we have

$$\begin{array}{rcll} \mathsf{LAG}(\tau, u'+1) & < & \lambda(1-\alpha)M + 1 & (58) \\ & \leq & (t_d - (u'+1))(1-\alpha)M + 1 \qquad \text{(by (57))}. & (59) \end{array}$$

For all $\hat{t}$, where $u' + 2 \le \hat{t} \le t_d - \lambda$, by (15), we have the following.

$$\begin{array}{rcll}
\mathsf{LAG}(\tau, \hat{t}) & \le & \mathsf{LAG}(\tau, u'+1) + (\hat{t} - u' - 1) \cdot \sum_{T \in \tau} wt(T) - \sum_{v=u'+1}^{\hat{t}-1} \sum_{T \in \tau} \mathcal{S}(T, v) & \\
& < & (\lambda - \lambda\alpha)M + 1 + (\hat{t} - u' - 1) \cdot \sum_{T \in \tau} wt(T) - \sum_{v=u'+1}^{\hat{t}-1} \sum_{T \in \tau} \mathcal{S}(T, v) & \text{(by (58))} \\
& = & (\lambda - \lambda\alpha)M + 1 + (\hat{t} - u' - 1) \cdot \sum_{T \in \tau} wt(T) - (\hat{t} - u' - 1) \cdot M & \text{(by (D5) and the definition of } \hat{t}\text{, there in no hole in } [u'+1, \hat{t})) \\
& = & (\lambda - \lambda\alpha)M + 1 + \alpha(\hat{t} - u' - 1) \cdot M - (\hat{t} - u' - 1) \cdot M & \text{(by Def. 6)} \\
& \le & (\lambda - \lambda\alpha)M + 1 & \text{(by Lemma 7)} \\
& \le & (t_d - \hat{t}) \cdot (1 - \alpha)M + 1 & \text{(because } \hat{t} \le t_d - \lambda) \qquad (60)
\end{array}$$

By the definition of u', (59), the definition of $\hat{t}$, and (60), condition (53) holds for this case. ■

By part (i) of Lemma 14, there exists a t, where $0 \le t < v$ and v is as defined in Lemma 14(h), such that $\mathsf{LAG}(\tau, t) < 1$ and $\mathsf{LAG}(\tau, s) \ge 1$, for all s in $[t+1, v]$. Since by Lemma 16, $t \ge \lambda - 1$ holds, Lemma 23 applies for t. Let $t' = u + \lambda$, where u is as defined in Lemma 23. If (51) holds, then

(E) $u \le t_d - 2\lambda$ and there is a hole in $u - 1$,

and $\mathsf{LAG}(\tau, t') < 1$ holds. Hence, if $t' < v$ holds, then the existence of t, and hence Lemma 14(i), is contradicted. On the other hand, if $t' \ge v$ holds, then, because, by (E), there is a hole in $u - 1 < t_d - 2\lambda$, and by the definition of v in Lemma 14(h), which states that there is no hole in any slot in $[v, t_d - \lambda)$, we have $u - 1 < v$, *i.e.*, $v \ge u$. Therefore, we have, $u \le v \le t'$. However, by (51), LAG at every time between u and $u + \lambda - 1 = t' - 1$ is less than $(\lambda - \lambda\alpha)M + 1$, and at time $u + \lambda = t'$ is less than one, and hence, is less than $(\lambda - \lambda\alpha)M + 1$ (as $\alpha \le 1$). This contradicts Lemma 14(h). If (52) holds, then Lemma 14(c) is contradicted. Finally, if (53) holds, then at every time s after the latest slot with a hole before $t_d - \lambda$ and until $t_d - \lambda$, $\mathsf{LAG}(\tau, s) < (t_d - s) \cdot (1 - \alpha)M + 1$, which contradicts part (h) of Lemma 14. Therefore, our assumption that τ misses a deadline at t_d is incorrect, which in turn proves Theorem 1.

According to Theorem 1, $\min(M, U(M, \lambda, \rho_{\max}))$ is a schedulable utilization bound for EPDF on M processors. We now show that every GIS task system with total utilization at most $\min(M, U(M, \lambda, W_{\max}))$ is also correctly scheduled under EPDF. For this, first note that the first derivative of $U(M, \lambda, f) = \frac{\lambda M(\lambda(1+f)-f)+1+f}{\lambda^2(1+f)}$ with respect to f is given by $\frac{-M}{\lambda(1+f)^2}$, which is negative for all $M \ge 1$, $\lambda \ge 1$, and $0 \le f \le 1$. Hence, $U(M, \lambda, f)$ is a decreasing function of f for $0 \le f \le 1$. By (19), $\rho(T) < wt(T)$ holds for each task T. Therefore, $0 \le \rho_{\max} < W_{\max} \le 1$ holds, and hence, $U(M, \lambda, W_{\max}) < U(M, \lambda, \rho_{\max})$ holds, by which we have the following corollary to Theorem 1.

Corollary 1 *A GIS task system τ is schedulable on M processors under* EPDF *if the total utilization of τ is at most* $\min(M, \frac{\lambda M(\lambda(1+W_{\max})-W_{\max})+1+W_{\max}}{\lambda^2(1+W_{\max})})$, *where $W_{\max}$ and λ are as defined in* (21) *and* (22), *respectively.*

Comparison with EDF. Fig. 7 shows how the utilization bound derived for EPDF compares to the best known bounds of partitioned EDF (p-EDF) [37], global EDF (g-EDF) [30, 10], and a variant of global EDF, called fp-EDF [30], in which tasks with utilizations greater than 1/2 are statically given higher priority than the remaining

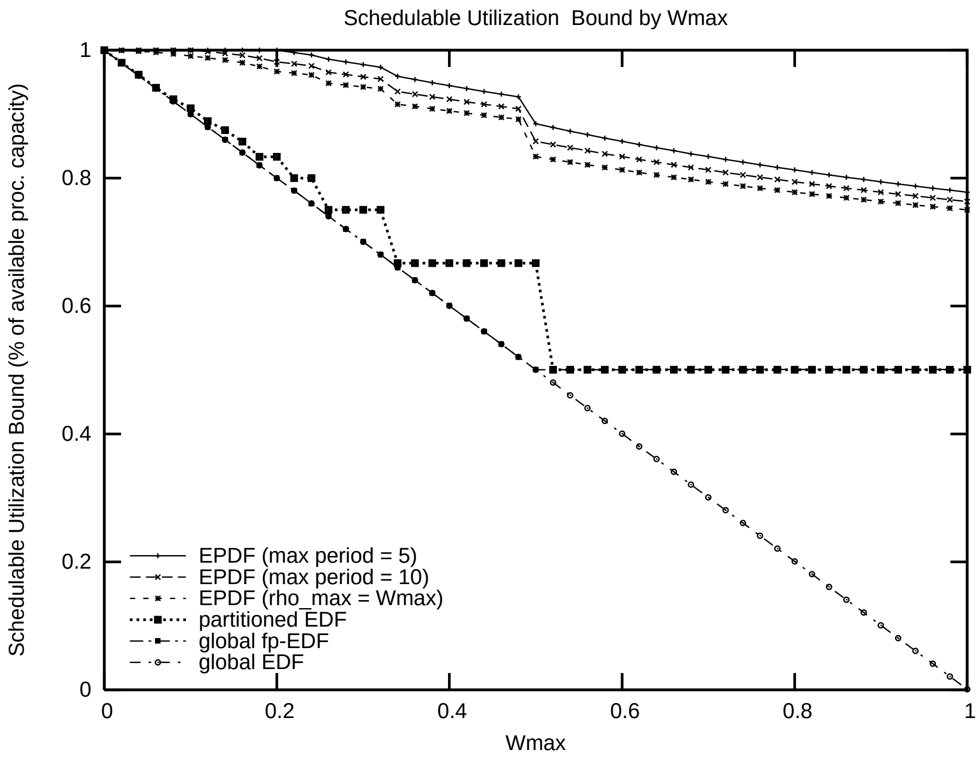

Figure 7. Schedulable utilization Bound by $W_{\max}$. The order of the curves coincides with that in the legend.

tasks, and the remaining tasks are scheduled under EDF [13]. As discussed in Sec. 1, for any M, there exist task sets with utilization roughly equal to 1.0 that can miss deadlines under g-EDF, and hence, the worst-case utilization bound under this algorithm approaches 0.0% of the available processing capacity. In general, there does not exist a correlation between $W_{\max}$ and $\rho_{\max}$. That is, $W_{\max}(\tau) > W_{\max}(\tau') \not\Rightarrow \rho_{\max}(\tau) > \rho_{\max}(\tau')$, where τ and τ' are two distinct task systems. For example, though $\frac{7}{12} > \frac{55}{96}$, $\rho(\frac{7}{12}) = \frac{1}{2}$, which is less than $\rho(\frac{55}{96}) = \frac{9}{16}$. Thus, a general characterization of any utilization bound as a function of $\rho_{\max}$ may not convey useful information. Hence, in Fig. 7, each bound is plotted as a function of $W_{\max}$ only. Also, $\rho_{\max}(\tau)$ cannot be determined if only $W_{\max}(\tau)$ is known. However, since $\gcd(T.e, T.p) \geq 1$ holds for every T, by (19) and (20), $\rho_{\max}(\tau)$ is at most $W_{\max}(\tau) - 1/p_{\max}(\tau)$, where $p_{\max}$ is the maximum period of any task. Hence, for an arbitrary task system, a utilization bound higher than that given by $\min(M, U(M, \lambda, W_{\max}))$ is possible if at least $p_{\max}$ is also known. Thus, in Fig. 7, utilization bounds for EPDF for tasks systems with $p_{\max} = 5$ and $p_{\max} = 10$ computed using Theorem 1 assuming $\rho_{\max} = W_{\max} - 1/p_{\max}$ are plotted, as is the $U(M, \lambda, \rho_{\max})$ bound. As can be seen from the plots, the utilization bound of EPDF is higher than those of p-EDF and fp-EDF by around 25% when $W_{\max} > 1/2$.

Note that as is generally the case with utilization bounds, the bound for EPDF also decreases with increasing $W_{\max}$. This bound is also piece-wise continuous with discontinuities whenever λ changes, *i.e.*, at $W_{\max} = 1/p$, where p is an integer. In other words, when considering $W_{\max}$ for increasing values, there is a sharper decrease in the utilization bound at $W_{\max} = 1/p$ than at other points, with the decrease becoming more pronounced as p decreases. In fact, due to scale, the discontinuities are not perceptible for $p > 3$ (*i.e.*, $W_{\max} < 0.3\bar{3}$) in the curves in Fig. 7, and the drop in the bound is more pronounced for $p = 2$ (*i.e.*, at $W_{\max} = 0.5$) than for $p = 3$ (*i.e.*, at $W_{\max} = 0.3\bar{3}$). These discontinuities are due to the following reason. The bound given by Corollary 1 decreases with increasing $W_{\max}$ and decreasing λ; λ is not an independent variable, but a decreasing piece-wise constant function of $W_{\max}$, and assumes a constant value of p for $W_{\max}$ in the subinterval $[\frac{1}{p}, \frac{1}{p-1})$, where $p \geq 2$. Further, the EPDF bound is a continuous decreasing function of $W_{\max}$ for a given λ, and has a term proportional to $\frac{-1}{\lambda}$. Therefore, it incurs an extra drop that is proportional to the change in $1/\lambda$ when λ decreases. For example, at $W_{\max} = 0.5$, $1/\lambda$ increases by 33.3% to 0.5 (from $0.3\bar{3}$), which is higher than the 25% increase at $W_{\max} = 0.3\bar{3}$ (from 0.25 to $0.3\bar{3}$). Hence, the drop is higher at $W_{\max} = 0.5$. In the context of the derivation of the bound, when λ decreases by one, the number of contiguous slots without holes decreases by one, resulting in a drop of $M - U_{sum}$ to the permissible accumulation of LAG, which in turn leads to a drop in the utilization bound in a non-continuous manner. Also, the percentage by which the total permissible LAG drops (due to a decrease of one slot without a hole) increases with decreasing number of slots without holes, and hence the highest drop at $\lambda = 2$ (which value is encountered first when $W_{\max} = 0.5$).

Similar characteristic as noted in the EPDF plots and described above can be observed in the plot for p-EDF also. In the case of p-EDF, the utilization bound depends on the minimum number of tasks, say β, that can necessarily be assigned to each processor by a partitioning algorithm. β can easily be seen to be $\left\lfloor \frac{1}{W_{\max}} \right\rfloor$, and for reasonable partitioning heuristics, the utilization bound is roughly equal to $\frac{\beta M}{\beta+1}$, which is a constant $p-1$ for $W_{\max} \in (\frac{1}{p}, \frac{1}{p-1}]$. Also, the bound is an increasing function of β and, hence, since β decreases by one at $W_{\max} = 1/p$, the bound exhibits discontinuous drops at $W_{\max} = 1/p$. To understand the result, consider a task set with $W_{\max} = \frac{1}{3} + \epsilon = \frac{1}{3-\epsilon'}$. In this case $\beta = \left\lfloor \frac{1}{W_{\max}} \right\rfloor = 2$, and since at least two tasks of utilization $W_{\max}$ of every such task set can be assigned to each processor, it can be shown that the total utilization of all the tasks assigned to each processor is at least $2 \cdot (1/3 + \epsilon)$. The utilization bound in this case is therefore at least $2 \cdot (1/3 + \epsilon) \cdot M \geq 2 \cdot 1/3 \cdot M$. Since $\beta = \left\lfloor \frac{1}{W_{\max}} \right\rfloor \Rightarrow W_{\max} > \frac{1}{\beta+1}$, the above argument can be generalized to show that for an arbitrary $W_{\max}$, the total utilization of tasks assigned to each processor is at least $\beta \cdot 1/(\beta+1)$ and that the utilization bound is at least $\frac{\beta M}{\beta+1}$.

In Sec. 3, we mentioned that Srinivasan and Anderson have shown that EPDF can correctly schedule every task system with $W_{\max} \leq \frac{1}{M-1}$ [44]. With $W_{\max} = \frac{1}{M-1}$, the utilization bound given by Corollary 1 is $M - \frac{1}{M-1} + \frac{1}{(M-1)^2}$, which approaches M asymptotically. The discrepancy for lower values of M is due to the approximation of $\rho_{\max}$ by $W_{\max}$. Hence, the results of this paper are useful only when $W_{\max} > \frac{1}{M-1}$ holds.

How accurate is the utilization bound given by Theorem 1? Is it the worst-case for EPDF? As yet, we do not fully know the answers to these questions. The closest we have come towards providing any answer is the following task system, which can miss deadlines under EPDF.

Counterexample. Consider a task system that consists of $2n + 1$ tasks of weight $\frac{1}{2}$, n tasks of weight $\frac{3}{4}$, and n tasks of weight $\frac{5}{6}$ scheduled on $3n$ processors, where $n \geq 2$. There is an EPDF schedule for this task system in which a deadline miss occurs at time 12. One such schedule is shown in Fig. 8. The total utilization of this task system is $\frac{31n}{12} + \frac{1}{2}$, which approaches 86.1% of $3n$, the total processing capacity, as n approaches infinity. The

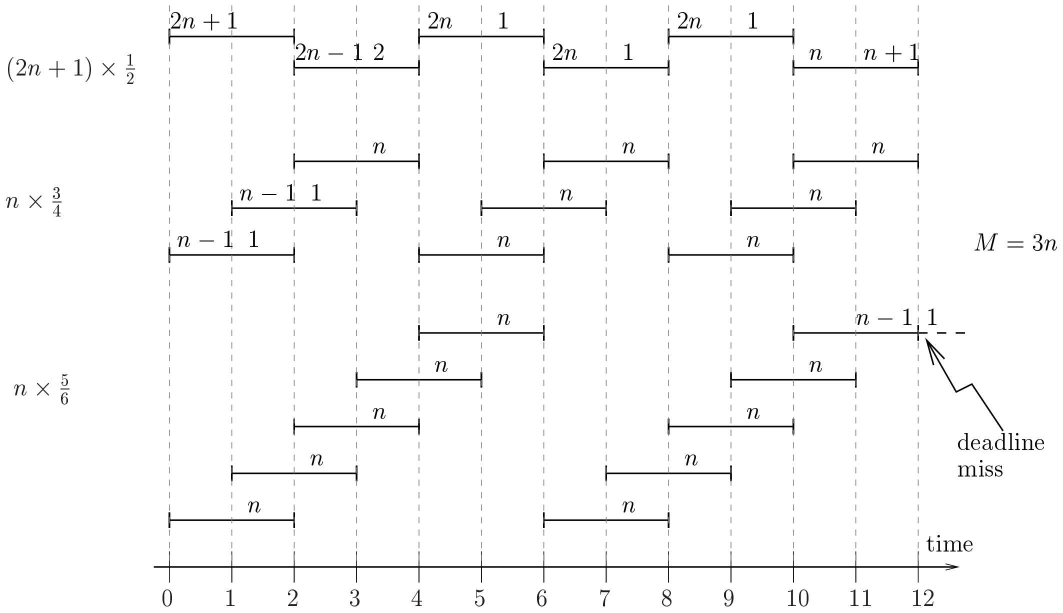

Figure 8. An EPDF schedule with a deadline miss for a task system with $2n+1$ tasks of weight $\frac{1}{2}$, n tasks with weight $\frac{3}{4}$, and n tasks with weight $\frac{5}{6}$ on $M = 3n$ processors, where $n \geq 2$. The number of tasks for each weight scheduled in each time slot is indicated. As $n \rightarrow \infty$, the total utilization of this task system approaches 86.1%.

utilization bound given by Theorem 1 for a task system with task parameters as in this example is slightly above 80% for large M. Hence, at least asymptotically, the value derived does not seem to be too pessimistic. However, the same cannot be said when M is small. For instance, for the example under consideration, the total utilizations when $n = 2$ and $n = 3$ (*i.e.*, $M = 6$ and $M = 9$) are 94.4% and 91.6%, respectively, while the estimated values are only 84.1% and 82.7%. Further, though this single task system is too little data to draw general conclusions, the difficulty in identifying counterexamples and some of the approximations made in the analysis make us strongly believe that our result is not tight.

Utilization restriction for a tardiness of q quanta. Having determined a sufficient utilization restriction for schedulability under EPDF, we were interested in determining a sufficient utilization restriction for a tardiness of $q \geq 1$ quanta. Extending the technique used in this paper, we have found that if the total utilization of τ is at most $\min(M, \frac{((q+1)W_{\max}+(q+2))M+((2q+1)W_{\max}+1)}{2(q+1)W_{\max}+2})$, then no subtask of τ misses its deadline by more than q quanta. For a tardiness of at most one, this imposes a sufficient utilization restriction of 83.3%. A proof can be found in [26].

5 Conclusion

We have determined a utilization bound for the non-optimal earliest-pseudo-deadline-first (EPDF) Pfair scheduling algorithm on multiprocessors, and thereby, presented a sufficient schedulability test for EPDF. In general, on $M > 2$ processors, the utilization bound derived is at least $\frac{3M+1}{4}$, and is higher than that known for every non-Pfair algorithm by around 25% when task utilizations are not restricted. Our bound is expressed as a decreasing function of the maximum weight, $W_{\max}$, of any task, and hence, if this value is known, may be used to schedule task systems with total utilization exceeding $\frac{3M+1}{4}$. EPDF is more efficient than optimal Pfair algorithms and may be preferable for systems instantiated on less-powerful platforms or systems whose task composition can change at run-time.

Currently, it is not known whether the utilization bound derived is the worst-case for EPDF. We have only presented a counterexample that shows that the worst-case value for $W_{\max} \leq \frac{5}{6}$ is at most 86.1%, which also suggests that a significant improvement to the result may not be likely. We have also considered extending the result to allow non-zero tardiness.

References

[1] Quad-core Intel®Xeon®processor 5300 series. http://download.intel.com/products/processor/xeon/dc53kprodbrief.pdf. Product Brief.

[2] V. Agarwal, M. S. Hrishikesh, S. Keckler, and D. Burger. Clock rate versus ipc: The end of the road for conventional microarchitectures. In *Proceedings of the 27th International Symposium on Computer Architecture*, pages 248–259, June 2000.

[3] J. Anderson, S. Baruah, and K. Jeffay. Parallel switching in connection-oriented networks. In *Proceedings of the 20th IEEE Real-time Systems Symposium*, pages 200–209. IEEE, December 1999.

[4] J. Anderson and A. Srinivasan. Early-release fair scheduling. In *Proceedings of the 12th Euromicro Conference on Real-Time Systems*, pages 35–43, June 2000.

[5] J. Anderson and A. Srinivasan. Pfair scheduling: Beyond periodic task systems. In *Proceedings of the 7th International Conference on Real-Time Computing Systems and Applications*, pages 297–306, December 2000.

[6] 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.

[7] 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.

[8] B. Andersson and J. Jonsson. The utilization bounds of partitioned and pfair static-priority scheduling on multiprocessors are 50%. In *Proceedings of the 15th Euromicro Conference on Real-Time Systems*, pages 33–40, July 2003.

[9] T. Baker, N. Fisher, and S. Baruah. Algorithms for determining the load of a sporadic task system. Technical Report TR-051201, Department of Computer Science, Florida State University, December 2005.

[10] 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.

[11] E. Bampis and G.N. Rouskas. The scheduling and wavelength assignment problem in optical WDM networks. *Journal of Lightwave Technology*, 20(5):782–789, May 2002.

[12] S. Baruah. Fairness in periodic real-time scheduling. In *Proceedings of the 16th IEEE Real-Time Systems Symposium*, pages 200–209, December 1995.

[13] 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.

[14] S. Baruah and J. Carpenter. Multiprocessor fixed-priority scheduling with restricted inter-processor migrations. In *Proceedings of the 15th Euromicro Conference on Real-Time Systems*, pages 195–202, July 2003.

[15] 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.

[16] S. Baruah and N. Fisher. The partitioned multiprocessor scheduling of sporadic task systems. In *Proceedings of the 26th Real-Time Systems Symposium*, pages 321–329, December 2005.

[17] S. Baruah, J. Gehrke, and C. G. Plaxton. Fast scheduling of periodic tasks on multiple resources. In *Proceedings of the 9th International Parallel Processing Symposium*, pages 280–288, April 1995.

[18] S.K. Baruah, R.R. Howell, and L.E. Rosier. Feasibility problems for recurring tasks on one processor. *Theoretical Computer Science*, 20(3):3–20, 1993.

[19] A. Block, J. Anderson, and G. Bishop. Fine-grained task reweighting on multiprocessors. In *Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 429–435, August 2005.

[20] R. Braden, D. Clark, and S. Shenker. Integrated Services in the Internet Architecture: an Overview. RFC 1633 (Informational), June 1994.

[21] B. Brandenburg and J. Anderson. Integrating hard/soft real-time tasks and best-effort jobs on multiprocessors. In *Proceedings of the 19th Euromicro Conference on Real-Time Systems*, July 2007. To appear.

[22] J. Calandrino, H. Leontyev, A. Block, U. Devi, and J. Anderson. LITMUS$^{\mathrm{RT}}$: A testbed for empirically comparing real-time multiprocessor schedulers. In *Proceedings of the 27th IEEE Real-Time Systems Symposium*, pages 111–123, December 2006.

[23] A. Chandra, M. Adler, P. Goyal, and P. Shenoy. Surplus fair scheduling: A Proportional-share CPU scheduling algorithm for symmetric multiprocessors. In *Proceedings of the 4th USENIX Symposium on Operating System Design and Implemetation*, pages 45–58, June 2000.

[24] H. Cho, B. Ravindran, and E. Jensen. An optimal real-time scheduling algorithm for multiprocessors. In *Proceedings of the 27th IEEE Real-Time Systems Symposium*, 2006.

[25] K. Coffman and A. Odlyzko. The size and growth rate of the internet. http://www.firstmonday.org/issues/issue3_10/coffman/ and http://www.dtc.umn.edu/~odlyzko/doc/internet.size.pdf, March 2001.

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

[27] 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).

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

[29] N. Fisher, T. Baker, and S. Baruah. Algorithms for determining the demand-based load of a sporadic task system. In *Proceedings of the 12th IEEE International Conference on Real-Time and Embedded Computing Systems and Applications*, August 2006.

[30] 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.

[31] 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.

[32] J. Kaur and H. Vin. Core-stateless guaranteed rate scheduling algorithms. In *Proceedings of IEEE INFOCOM*, pages 1484–1492. IEEE, April 2001.

[33] J. Kaur and H. Vin. Core-stateless guaranteed throughput networks. In *Proceedings of IEEE INFOCOM*, pages 2155–2165. IEEE, April 2003.

[34] 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.

[35] J. P. Lehoczky, L. Sha, and Y. Ding. Rate-monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings of the 11th IEEE Real-Time Systems Symposium*, pages 166–171, December 1989.

[36] 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.

[37] J.M. Lopez, M. Garcia, J.L. Diaz, and D.F. Garcia. Worst-case utilization bound for EDF scheduling on real-time multiprocessor systems. In *Proceedings of the 12th Euromicro Conference on Real-Time Systems*, pages 25–34, June 2000.

[38] D. Oh and T. P. Baker. Utilization bounds for N-processor rate monotone scheduling with static processor assignment. *Real-Time Systems: The International Journal of Time-Critical Computing*, 15:183–192, 1998.

[39] R. Rajkumar. Resource Kernels: Why Resource Reservation should be the Preferred Paradigm of Construction of Embedded Real-Time Systems. Keynote talk, 18th Euromicro Conference on Real-Time Systems, Dresden, Germany, July 2006.

[40] S. Ramamurthy and M. Moir. Static-priority periodic scheduling on multiprocessors. In *Proceedings of the 21st IEEE Real-Time Systems Symposium*, pages 69–78, December 2000.

[41] A. Srinivasan. *Efficient and Flexible Fair Scheduling of Real-Time Tasks on Multiprocessors.* PhD thesis, University of North Carolina at Chapel Hill, December 2003.

[42] 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.

[43] 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.

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

[45] A. Srinivasan, P. Holman, J. Anderson, S. Baruah, and J. Kaur. Multiprocessor scheduling in processor-based router platforms: Issues and ideas. In *Proceedings of the Second Workshop on Network Processors*, pages 48–62, February 2003.

[46] I. Stoica, S. Shenker, and H. Zhang. Core -stateless fair queueing: Achieving approximately fair bandwidth allocations in high speed networks. In *SIGCOMM*, pages 118–130, 1998.

[47] I. Stoica and H. Zhang. Providing guaranteed services without per flow management. In *SIGCOMM*, pages 81–94, 1999.

Appendix: Proofs Omitted in the Main Text

Lemma 3 *For all $i \geq 1$, $k \geq 1$, the following holds.*

$$r(T_{i+k}) \geq \begin{cases} d(T_i) + k - 1, & b(T_i) = 0 \\ d(T_i) + k - 2, & b(T_i) = 1 \end{cases}$$

Proof: We consider two cases based on $b(T_i)$.

Case 1: $b(T_i) = 0$. In this case, by (16), $\frac{i}{wt(T)}$ is an integer. Hence,

$$
\begin{aligned}
r(T_{i+k}) &= \Theta(T_{i+k}) + \left\lfloor \frac{i+k-1}{wt(T)} \right\rfloor && \text{(by (1))} \\
&= \Theta(T_{i+k}) + \frac{i}{wt(T)} + \left\lfloor \frac{k-1}{wt(T)} \right\rfloor && \text{(because } \tfrac{i}{wt(T)} \text{ is an integer)} \\
&= \Theta(T_{i+k}) + \left\lceil \frac{i}{wt(T)} \right\rceil + \left\lfloor \frac{k-1}{wt(T)} \right\rfloor \\
&> \Theta(T_{i+k}) + \left\lceil \frac{i}{wt(T)} \right\rceil + \frac{k-1}{wt(T)} - 1 \\
&\geq \Theta(T_i) + \left\lceil \frac{i}{wt(T)} \right\rceil + \frac{k-1}{wt(T)} - 1 && \text{(because } \Theta(T_{i+k}) \geq \Theta(T_i)\text{)} \\
&= d(T_i) + \frac{k-1}{wt(T)} - 1 && \text{(by (1))} \\
&\geq d(T_i) + k - 2 && \text{(because } wt(T) \leq 1 \text{ and } k \geq 1\text{).}
\end{aligned}
$$

Because, $r(T_{i+k})$ is an integer, the above implies that $r(T_{i+k}) \geq d(T_i) + k - 1$.

Case 2: $b(T_i) = 1$. In this case,

$$
\begin{aligned}
r(T_{i+k}) &= \Theta(T_{i+k}) + \left\lfloor \frac{i+k-1}{wt(T)} \right\rfloor && \text{(by (1))} \\
&> \Theta(T_{i+k}) + \frac{i+k-1}{wt(T)} - 1 \\
&= \Theta(T_{i+k}) + \frac{i}{wt(T)} + \frac{k-1}{wt(T)} - 1 \\
&> \Theta(T_{i+k}) + \left\lceil \frac{i}{wt(T)} \right\rceil + \frac{k-1}{wt(T)} - 2 \\
&\geq \Theta(T_i) + \left\lceil \frac{i}{wt(T)} \right\rceil + \frac{k-1}{wt(T)} - 2 && \text{(because } \Theta(T_{i+k}) \geq \Theta(T_i)\text{)} \\
&= d(T_i) + \frac{k-1}{wt(T)} - 2 && \text{(by (1))} \\
&\geq d(T_i) + k - 3 && \text{(because } wt(T) \leq 1 \text{ and } k \geq 1\text{).}
\end{aligned}
$$

For this case again, the lemma holds because $r(T_{i+k})$ is an integer. ■

Lemma 8 $0 \leq \delta < \frac{1}{2}$.

Proof: By Definition 7, $\delta = \frac{\rho_{\max}}{1+\rho_{\max}}$. The first derivative of δ with respect to $\rho_{\max}$, $\delta'(\rho_{\max})$, is $\frac{1}{(1+\rho_{\max})^2}$, which is positive, and hence, δ is increasing with $\rho_{\max}$ (though the rate of increase decreases with $\rho_{\max}$). By (20) and (19), $0 \leq \rho_{\max} < W_{\max}$ holds. Hence, $\rho_{\max}$ lies in the range $[0, 1)$. $\delta(0) = 0$ and $\delta(1) = \frac{1}{2}$, from which, the lemma follows. ■

Lemma 13 *If* $\mathsf{LAG}(\tau, t+1) > \mathsf{LAG}(\tau, t-\ell)$, *where* $0 \leq \ell \leq \lambda - 2$ *and* $t \geq \ell$, *then* $B(t-\ell, t+1) \neq \emptyset$.

Proof: We will refer to sets $A(t-\ell, t+1)$, $B(t-\ell, t+1)$, and $I(t-\ell, t+1)$ as A, B, and I, respectively. By (12), $\mathsf{LAG}(\tau, t+1) = \mathsf{LAG}(\tau, t-\ell) + \sum_{T \in \tau}(\mathsf{A}(\mathsf{ideal}, T, t-\ell, t+1) - \mathsf{A}(\mathcal{S}, T, t-\ell, t+1))$. Because, tasks in I are neither active nor scheduled in the interval $[t-\ell, t+1)$, by (17), $\mathsf{LAG}(\tau, t+1) = \mathsf{LAG}(\tau, t-\ell) + \sum_{T \in A \cup B}(\mathsf{A}(\mathsf{ideal}, T, t-$

$\ell, t+1) - \mathsf{A}(\mathcal{S}, T, t-\ell, t+1))$ holds. Because $\ell \leq \lambda - 2$ holds by the statement of the lemma, $t+1-(t-\ell) \leq \lambda - 1$ holds. Therefore, by (4), for each T in $A \cup B$, $\mathsf{A}(\mathsf{ideal}, T, t-\ell, t+1) \leq (\lambda - 1) \cdot wt(T) \leq 1$ holds. The last inequality holds because, by (22), either $\lambda = 2$ or $W_{\max} \leq \frac{1}{\lambda - 1}$. Hence, $\mathsf{LAG}(\tau, t+1) \leq \mathsf{LAG}(\tau, t-\ell) + |A \cup B| - \sum_{T \in A \cup B} \mathsf{A}(\mathcal{S}, T, t-\ell, t+1)$. By the statement of the lemma, $\mathsf{LAG}(\tau, t+1) > \mathsf{LAG}(\tau, t-\ell)$. Therefore, it follows that $|A \cup B| - \sum_{T \in A \cup B} \mathsf{A}(\mathcal{S}, T, t-\ell, t+1) > 0$, *i.e.*,

$$\sum_{T \in A \cup B} \mathsf{A}(\mathcal{S}, T, t-\ell, t+1) < |A \cup B|. \tag{61}$$

Now, if B is empty, then, $|A \cup B| = |A|$ holds. Because, by definition, every task in A is scheduled at least once in the interval $[t-\ell, t+1)$, $\sum_{T \in A \cup B} \mathsf{A}(\mathcal{S}, T, t-\ell, t+1) \geq |A| = |A \cup B|$, which is a contradiction to (61). Therefore, $B(t-\ell, t+1) \neq \emptyset$. ■

Lemma 24 *Let T be a unit-weight task. Then, $d(T_i) = r(T_i) + 1$ and $|\omega(T_i)| = 1$ hold for every subtask T_i of T.*

Proof: Follows directly from $wt(T) = 1$, (1), and the definition of $\omega(T_i)$. ■

In some of the proofs that follow, we identify tasks that receive an allocation of zero at time t in the ideal schedule using the following definition.

Definition 8: A GIS task U is a *Z-task* at time t iff there exists no subtask U_j such that $r(U_j) \leq t < d(U_j)$. The set of all tasks that are Z-tasks at t is denoted $Z(t)$.

By (3), and because $\mathsf{A}(\mathsf{ideal}, U, t) = \sum_i \mathsf{A}(\mathsf{ideal}, U_i, t)$ holds, U is a Z-task at t iff $\mathsf{A}(\mathsf{ideal}, U, t) = 0$. Also, note that while $T \in I(t) \Rightarrow T \in Z(t)$, the converse does not hold. This is because, a task may have one or more subtasks whose IS-windows include slot t, and hence, are active at t, even if their release times are after t, *i.e.*, even if their PF-windows do not include slot t.

The next two lemmas concern tasks with unit weight that are Z-tasks at some time t. The first lemma says that if a subtask of a unit-weight task is scheduled before its release time, then that task is a Z-task during some earlier time.

Lemma 25 *Let T_i be a subtask of a unit-weight task T scheduled at some time $t < t_d$. If $r(T_i) > t$, then T is a Z task in at least one slot in $[0, t+1)$.*

Proof: Follows easily from the fact that unit-weight tasks have PF-windows of length one and no subtask is scheduled before time 0. ■

Lemma 26 *Let T be a unit-weight task and suppose T is not scheduled at some time $t < t_d$, where there is at least one hole in t. Then, there exists a time $t' \leq t$, such that there are no holes in $[t', t)$ and there exists at least one slot $\hat{t}$ in $[t', t+1)$, such that T is a Z-task in $\hat{t}$.*

Proof: Because T is not scheduled at t, by (17), either $T \in B(t)$ or $T \in I(t)$. If $T \in I(t)$, then T is a Z-task at t. Therefore, the lemma is trivially satisfied with $\hat{t} = t' = t$. So assume $T \in B(t)$ for the rest of the proof. Let T_j be the critical subtask at t of T. Then, by Definition 2, we have

$$d(T_j) \geq t + 1. \tag{62}$$

Also, because there is a hole in t, by Lemma 10, T_j is scheduled before t, say t'', *i.e.*,

$$t'' < t \ \wedge \ \mathcal{S}(T_j, t'') = 1. \tag{63}$$

Because there is a hole in t, by Lemma 11, $d(T_j) \leq t + 1$ holds, which, by (62), implies that $d(T_j) = t + 1$ holds. Because $wt(T) = 1$, by Lemma 24, we have $r(T_j) = t$. Thus,

$$r(T_j) = t \ \wedge \ d(T_j) = t + 1. \tag{64}$$

Because (63) and (64) hold, by Lemma 25, T is a Z-task at some time at or before t'', *i.e.*, before t. Let z be the latest such time. Then, we have

$$0 \leq z < t \ \wedge \ (\not\exists T_k : r(T_k) \leq z \ \wedge \ d(T_k) > z) \wedge \ (\forall z' : z < z' \leq t :: (\exists T_k : r(T_k) \leq z' \ \wedge \ d(T_k) > z')). \tag{65}$$

Because $wt(T) = 1$ holds, by Lemma 24, (65) implies the following.

$$0 \leq z < t \ \wedge \ (\not\exists T_k : r(T_k) = z \wedge \ d(T_k) = z + 1) \wedge \ (\forall z' : z < z' \leq t :: (\exists T_k : r(T_k) = z' \ \wedge \ d(T_k) = z' + 1)) \tag{66}$$

We next claim the following. (Informally, Claim 2 says that every subtask of T with release time in $[z + 1, t]$ is scheduled before its release time, and there is no hole in any slot between where the subtask is scheduled and its deadline.)

Claim 2 *If $r(T_k) = z' \ \wedge \ d(T_k) = z' + 1$ holds, where $z < z' \leq t$, then there exists a u', where $u' \leq t'' - (t - z') < z'$, such that T_k is scheduled at u', and there is no hole in any slot in $[u', z')$.*

The proof is by induction on decreasing values for z'. So, we begin with $z' = t$.

Base Case: $z' = t$. By (64), we have $r(T_j) = t$ and $d(T_j) = t + 1$, and by (63), we have $\mathcal{S}(T_j, t'') = 1$. Further, $t'' - (t - z') = t'' - (t - t) = t''$. By (63), $t'' < t = z'$. Therefore, to show that the claim holds for this case with $u' = t''$, we only have to show that there is no hole in any slot in $[t'', t)$. Assume to the contrary that there is a hole in slot t_h in $[t'', t)$. Then, by Lemma 11, $d(T_j) \leq t_h + 1 < t + 1$, which is in contradiction to (64). Hence, the assumption does not hold.

Induction Hypothesis: Assume that for all z', where $z'' \leq z' \leq t$ and $t \geq z'' > z + 1$, $(r(T_k) = z' \ \wedge$ $d(T_k) = z' + 1) \Rightarrow \ (\exists u' : u' \leq t'' - (t - z') < z' :: (\mathcal{S}(T_k, u') = 1 \ \wedge$ there are no holes in $[u', z'))$ holds. (Informally, the claim holds for all z' in $[z'', t]$.)

Induction Step: We now show that the following holds, *i.e.*, the claim holds, for $z'' - 1$.

$$\begin{aligned}(r(T_k) = z'' - 1 \ \wedge \ d(T_k) = z'') \Rightarrow (\exists u' : u' \leq t'' - (t - (z'' - 1)) < z'' - 1 ::\\ (\mathcal{S}(T_k, u') = 1 \ \wedge \ \text{there is no hole in any slot in } [u', z'' - 1)))\end{aligned} \tag{67}$$

By (66), there exists a subtask T_ℓ with $r(T_\ell) = z''$ and $d(T_\ell) = z''+1$. By the induction hypothesis, T_ℓ is scheduled at or before $t'' - (t - z'')$. Let T_k be a subtask such that $r(T_k) = z'' - 1$ and $d(T_k) = z''$. Then, because $r(T_k) < r(T_\ell)$ holds, $k < \ell$ holds. Also, $d(T_k) < d(T_\ell)$. Hence, by (2), T_k is scheduled before T_ℓ, *i.e.*, T_k is scheduled at or before $t'' - (t - z'') - 1 = t'' - (t - (z'' - 1))$. Because $t'' - (t - (z'' - 1)) = z'' - 1 - (t - t'')$, by (63), we have $t'' - (t - (z'' - 1)) < z'' - 1$. Thus, we have shown that if the antecedent of the implication in (67) holds, then the first subexpression of the consequent is satisfied. To see that there are no holes in $[t'' - (t - (z'' - 1)), z'' - 1)$, assume to the contrary that there is a hole in slot t_h in this interval. Then, by Lemma 11, $d(T_k) \leq t_h + 1 \leq z'' - 1$, which is in contradiction to $d(T_k) = z''$. Thus, the claim holds for $z'' - 1$. ■

(66) and Claim 2 imply that $(\forall z' : z < z' \leq t ::$ there are no holes in $z' - 1)$. Therefore, it immediately follows that there are no holes in $[z, t)$. Also, we defined z such that T is a Z-task at z. Therefore, the lemma follows. ■

Proof of parts (f), (g), (h), and (i) of Lemma 14.

Lemma 14 *The following properties hold for τ and $\mathcal{S}$.*

- **(a)** *For all T_i, $d(T_i) \leq t_d$.*
- **(b)** *Exactly one subtask of τ misses its deadline at t_d.*
- **(c)** $\mathsf{LAG}(\tau, t_d) = 1$.
- **(d)** *Let T_i be a subtask that is scheduled at $t < t_d$ in $\mathcal{S}$. Then, $e(T_i) = \min(r(T_i), t)$.*
- **(e)** $(\forall T_i :: d(T_i) < t_d \;\Rightarrow\; (\exists t :: e(T_i) \leq t < d(T_i) \;\wedge\; \mathcal{S}(T_i, t) = 1))$. *That is, every subtask with deadline before t_d is scheduled before its deadline.*
- **(f)** *Let U_k be the subtask that misses its deadline at t_d. Then, U is not scheduled in any slot in $[t_d - \lambda + 1, t_d)$.*
- **(g)** *There is no hole in any of the last $\lambda - 1$ slots,* i.e., *in slots in $[t_d - \lambda + 1, t_d)$.*
- **(h)** *There exists a time $v \leq t_d - \lambda$ such that the following both hold.*
  - **(i)** *There is no hole in any slot in $[v, t_d - \lambda)$.*
  - **(ii)** $\mathsf{LAG}(\tau, v) \geq (t_d - v)(1 - \alpha)M + 1$.
- **(i)** *There exists a time $u \in [0, v)$ where v is as defined in part (h), such that $\mathsf{LAG}(\tau, u) < 1$ and $\mathsf{LAG}(\tau, t) \geq 1$ for all t in $[u + 1, v]$.*

As mentioned in the main text, parts (a), (b), (c), and (d) are proved in [42]. Part (e) follows directly from Definition 4 and (T1). The other parts are proved below.

Proof of (f): If U_k is the first subtask of U, then it is trivial that U is not scheduled at any time before t_d. So assume U_k is not the first subtask of U and let U_j be U_k's predecessor. We first show that the deadline of U_j is at or before $t_d - \lambda + 1$. Because U_k misses its deadline at t_d, $d(U_k) = t_d$ holds. If $W_{\max} < 1$, then by Lemma 15, $|\omega(U_k)| \geq \lambda$. On the other hand, if $W_{\max} = 1$, then by (22), $\lambda = 2$. Hence, if $wt(U) < 1$, then by Lemma 1, $|\omega(U_k)| \geq 2 = \lambda$. Thus, if either $W_{\max} < 1$ or $W_{\max} = 1 \;\wedge\; wt(U) < 1$, then $r(U_k) \leq t_d - \lambda$ holds. Since U_k is U_j's predecessor, $k \leq j-1$ holds, and by Lemma 3, $d(U_j) \leq t_d - \lambda + 1$ follows. Finally, if $W_{\max} = 1$ and $wt(U) = 1$, then because the deadlines of any two consecutive subtasks of a unit-weight task are separated by at least one time slot, $d(U_j) \leq t_d - 1 = t_d - \lambda + 1$ follows.

Thus, by part (e), U_j and every prior subtask of U is scheduled before $t_d - \lambda + 1$. Obviously, no U_ℓ, where $\ell > k$, is scheduled before U_k is scheduled. Therefore, no subtask of U is scheduled in any slot in $[t_d - \lambda + 1, t_d)$. ■

Proof of (g): Let U_k denote the subtask that misses its deadline at t_d. By Lemma 15, $|\omega(U_k)| \geq \lambda - 1$ holds, Therefore, $r(U_k) \leq t_d - \lambda + 1$. Further, by part (f), no subtask of U is scheduled in any slot in $[t_d - \lambda + 1, t_d)$. Hence, U_k is ready and eligible to be scheduled in $[t_d - \lambda + 1, t_d)$. Thus, if there is a hole in this interval, then EPDF would schedule U_k there, which contradicts the fact that U_k misses its deadline. ■

Proof of (h): We prove this part using part (g) and Lemma 26. Let τ_1 and N_1 be defined as follows.

$$\tau_1 \stackrel{\text{def}}{=} \{T \mid T \in \tau \ \wedge \ wt(T) = 1\} \tag{68}$$

$$|\tau_1| \stackrel{\text{def}}{=} N_1 \tag{69}$$

Then, by (T0), we have

$$N_1 \leq M, \tag{70}$$

and by Definition 6,

$$\sum_{T \in \tau \setminus \tau_1} wt(T) = \alpha M - N_1. \tag{71}$$

By part (g), there is no hole in any slot in $[t_d - \lambda + 1, t_d)$. Therefore, by (70), at least $M - N_1$ tasks with weight less than one (*i.e.*, tasks in $\tau \setminus \tau_1$) are scheduled in each slot in $[t_d - \lambda + 1, t_d)$, *i.e.*,

$$\left(\forall t : t_d - \lambda + 1 \leq t < t_d :: \sum_{T \in \tau \setminus \tau_1} \mathcal{S}(T, t) \geq M - N_1 \right). \tag{72}$$

Let $\mathcal{T}$ denote the set of all subtasks of tasks not in τ_1 that are scheduled in some slot in $[t_d - \lambda + 1, t_d)$, *i.e.*,

$$\mathcal{T} = \{T_i \mid T \in \tau \setminus \tau_1 \ \wedge \ T_i \text{ is scheduled in the interval } [t_d - \lambda + 1, t_d)\}. \tag{73}$$

Let T_i be some subtask in $\mathcal{T}$. If $W_{\max} < 1$, then by Lemma 15, $|\omega(T_i)| \geq \lambda$ holds. Otherwise, since, $T \notin \tau_1$, $wt(T) < 1$ holds, and hence, by Lemma 1, $|\omega(T_i)| \geq 2 = \lambda$ holds. (The last equality follows from (22), since $W_{\max} = 1$.) Thus, in both cases, the length of the PF-window of every subtask in $\mathcal{T}$ is at least λ. By part (a), $d(T_i) \leq t_d$. Hence, because $|\omega(T_i)| \geq \lambda$ holds, it follows that $r(T_i) \leq t_d - \lambda$. Therefore, by Lemma 3, the deadline of the predecessor, if any, say, T_h, of T_i is at or before $t_d - \lambda + 1$. Hence, by part (e), T_h and all prior subtasks of T are scheduled before $t_d - \lambda + 1$. Thus, at most one subtask of every task not in τ_1 is scheduled in $[t_d - \lambda + 1, t_d)$. Furthermore, as shown above, the release time of each such subtask is at or before $t_d - \lambda$.

By (72), $|\mathcal{T}| \geq (\lambda - 1) \cdot (M - N_1)$, and by the discussion above, every subtask in $\mathcal{T}$ is of a distinct task, and the release time of each such subtask is at or before $t_d - \lambda$. By (69), in every time slot, at least $M - N_1$ processors are available for scheduling tasks not in τ_1. Therefore, the fact that no subtask in $\mathcal{T}$ is scheduled at $t_d - \lambda$ implies that either there is no hole in $t_d - \lambda$, that is, M other subtasks of tasks in τ are scheduled there, or there is a hole in $t_d - \lambda$ and the predecessor of every subtask in $\mathcal{T}$ (that is at least $(\lambda - 1) \cdot (M - N_1)$ distinct tasks in $\tau \setminus \tau_1$) is scheduled there. Let $h \geq 0$ denote the number of holes in slot $t_d - \lambda$. If $h > 0$ holds, then by the discussion above, $(\lambda - 1) \cdot (M - N_1) < M$ holds, and at least $(\lambda - 1) \cdot (M - N_1)$ tasks in $\tau \setminus \tau_1$ are scheduled in $t_d - \lambda$. Hence, at most $M - (\lambda - 1) \cdot (M - N_1) - h = (\lambda - 1)N_1 - (\lambda - 2)M - h$ tasks in τ_1 are scheduled at $t_d - \lambda$. Let

$$N_1^s \stackrel{\text{def}}{=} \sum_{T \in \tau_1} \mathcal{S}(T, t_d - \lambda). \tag{74}$$

Then, by the above discussion,

$$h > 0 \Rightarrow N_1^s \leq (\lambda - 1)N_1 - (\lambda - 2)M - h < (\lambda - 1)N_1 - (\lambda - 2)M. \tag{75}$$

Let τ_1^z denote the subset of all tasks in τ_1 that are not scheduled at $t_d - \lambda$. If τ_1^z is not empty and $h > 0$, then let Y be a task in τ_1^z. Then, by Lemma 26, there exists a time $u \leq t_d - \lambda$ such that Y is a Z-task at u and there is no hole in any slot in $[u, t_d - \lambda)$. Let v be defined as follows.

$$v \stackrel{\text{def}}{=} \begin{cases} t_d - \lambda, & \text{if } \tau_1^z = \emptyset \ \vee \ h = 0 \\ \begin{array}{l}\min_{Y \in \tau_1^z}\{u \leq t_d - \lambda \mid \text{there is no hole in } [u, t_d - \lambda) \\ \text{and } Y \text{ is a } Z\text{-task in at least one slot in } [u, t_d - \lambda + 1)\}\end{array}, & \text{if } \tau_1^z \neq \emptyset \ \wedge \ h > 0 \end{cases}$$

v satisfies the following.

$$\text{Every task in } \tau_1^z \text{ is a } Z\text{-task in at least one slot in the interval } [v, t_d - \lambda + 1). \tag{76}$$

$$\text{There is no hole in } [v, t_d - \lambda). \tag{77}$$

To complete the proof, we are left with determining a lower bound on $\mathsf{LAG}(\tau, v)$. By (12), we have

$$\begin{aligned}
\mathsf{LAG}(\tau, v) &= \mathsf{LAG}(\tau, t_d) - \textstyle\sum_{T \in \tau} \sum_{u=v}^{t_d - 1} (\mathsf{A}(\mathsf{ideal}, T, u) - \mathcal{S}(T, u)) && \text{, by (12)} \\
&= 1 - \textstyle\sum_{T \in \tau} \sum_{u=v}^{t_d - 1} (\mathsf{A}(\mathsf{ideal}, T, u) - \mathcal{S}(T, u)) && \text{, by part (c)} \\
&= 1 - \textstyle\sum_{T \in \tau} \sum_{u=v}^{t_d - 1} \mathsf{A}(\mathsf{ideal}, T, u) + \sum_{T \in \tau} \sum_{u=v}^{t_d - 1} \mathcal{S}(T, u) \\
&= 1 - (\textstyle\sum_{T \in \tau} \sum_{u=v}^{t_d - 1} \mathsf{A}(\mathsf{ideal}, T, u)) + (t_d - v)M - h \\
& && \text{, there are } h \text{ holes in } t_d - \lambda\text{; by part (g), there is no hole in} \\
& && [t_d - \lambda + 1, t_d)\text{, and by (77), there is no hole in } [v, t_d - \lambda) \\
&= 1 - \textstyle\sum_{T \in \tau_1} \sum_{u=v}^{t_d - \lambda} \mathsf{A}(\mathsf{ideal}, T, u) - \sum_{T \in \tau \setminus \tau_1} \sum_{u=v}^{t_d - \lambda} \mathsf{A}(\mathsf{ideal}, T, u) \\
&\quad - \textstyle\sum_{T \in \tau} \sum_{u=t_d - \lambda + 1}^{t_d - 1} \mathsf{A}(\mathsf{ideal}, T, u) + (t_d - v)M - h \\
&\geq 1 - \textstyle\sum_{T \in \tau_1} \sum_{u=v}^{t_d - \lambda} \mathsf{A}(\mathsf{ideal}, T, u) - \sum_{T \in \tau \setminus \tau_1} \sum_{u=v}^{t_d - \lambda} wt(T) \\
&\quad - \textstyle\sum_{T \in \tau} \sum_{u=t_d - \lambda + 1}^{t_d - 1} wt(T) + (t_d - v)M - h && \text{, by (4)} \\
&= 1 - \textstyle\sum_{T \in \tau_1} \sum_{u=v}^{t_d - \lambda} \mathsf{A}(\mathsf{ideal}, T, u) \\
&\quad -(\alpha M - N_1)(t_d - v - \lambda + 1) - (\lambda - 1)\alpha M + (t_d - v)M - h && \text{, by (71) and Def. 6} \\
&= 1 - \textstyle\sum_{T \in (\tau_1 \setminus \tau_1^z)} \sum_{u=v}^{t_d - \lambda} \mathsf{A}(\mathsf{ideal}, T, u) - \sum_{T \in \tau_1^z} \sum_{u=v}^{t_d - \lambda} \mathsf{A}(\mathsf{ideal}, T, u) \\
&\quad -(\alpha M - N_1)(t_d - v - \lambda + 1) - (\lambda - 1)\alpha M + (t_d - v)M - h \\
&\geq 1 - \textstyle\sum_{T \in (\tau_1 \setminus \tau_1^z)} \sum_{u=v}^{t_d - \lambda} wt(T) - \sum_{T \in \tau_1^z} \sum_{u=v}^{t_d - \lambda} \mathsf{A}(\mathsf{ideal}, T, u)
\end{aligned}$$

$$
\begin{aligned}
& -(\alpha M - N_1)(t_d - v - \lambda + 1) - (\lambda - 1)\alpha M + (t_d - v)M - h && \text{, by (4)} \\
= \;& 1 - (N_1^s)(t_d - v - \lambda + 1) - \textstyle\sum_{T \in \tau_1^z} \sum_{u=v}^{t_d-\lambda} \mathsf{A}(\mathsf{ideal}, T, u) \\
& -(\alpha M - N_1)(t_d - v - \lambda + 1) - (\lambda - 1)\alpha M + (t_d - v)M - h && \text{, by the definitions of } \tau_1 \text{ and } \tau_1^z\text{, (69), and (74)} \\
= \;& 1 - \textstyle\sum_{T \in \tau_1^z} \sum_{\{u \mid v \le u \le t_d - \lambda \;\wedge\; T \text{ is not a } Z\text{-task at } u\}} \mathsf{A}(\mathsf{ideal}, T, u) \\
& -(N_1^s)(t_d - v - \lambda + 1) - (\alpha M - N_1)(t_d - v - \lambda + 1) - (\lambda - 1)\alpha M + (t_d - v)M - h \\
& && \text{, by Def. 8 (}Z\text{-task) and (3)} \\
\ge \;& 1 - \textstyle\sum_{T \in \tau_1^z} \sum_{\{u \mid v \le u \le t_d - \lambda \;\wedge\; T \text{ is not a } Z\text{-task at } u\}} wt(T) \\
& -(N_1^s)(t_d - v - \lambda + 1) - (\alpha M - N_1)(t_d - v - \lambda + 1) - (\lambda - 1)\alpha M + (t_d - v)M - h \\
& && \text{, by (4)} \\
\ge \;& 1 - \textstyle\sum_{T \in \tau_1^z} (t_d - v - \lambda) wt(T) - (N_1^s)(t_d - v - \lambda + 1) \\
& -(\alpha M - N_1)(t_d - v - \lambda + 1) - (\lambda - 1)\alpha M + (t_d - v)M - h && \text{, by (76)} \\
= \;& 1 - (N_1 - N_1^s)(t_d - v - \lambda) - (N_1^s)(t_d - v - \lambda + 1) \\
& -(\alpha M - N_1)(t_d - v - \lambda + 1) - (\lambda - 1)\alpha M + (t_d - v)M - h && \text{, by the definitions of } \tau_1 \text{ and } \tau_1^z\text{, (69), and (74)} \\
= \;& 1 + N_1 - N_1^s - \alpha M(t_d - v) + M(t_d - v) - h && \text{, simplifying} \\
\ge \;& \begin{cases} 1 + N_1 - N_1^s - \alpha M(t_d - v) + M(t_d - v), & h = 0 \\ 1 - (\lambda - 2)N_1 + (\lambda - 2)M - \alpha M(t_d - v) + M(t_d - v), & h > 0 \end{cases} && \text{, by the first inequality in (75)} \\
\ge \;& 1 + (M - \alpha M)(t_d - v). && \text{, because } N_1 - N_1^s \ge 0\text{, and, by (69) and (70), } N_1 \le M
\end{aligned}
$$

■

Proof of (i): Follows from the facts that $\mathsf{LAG}(\tau, 0) = 0$ and there exists a $v \le t_d - \lambda$ such that $LAG(\tau, v) \ge (t_d - v)(1 - \alpha)M + 1$ (from part (h)). Note that, by Lemma 7, $(t_d - v)(1 - \alpha)M + 1 \ge 1$. ■

Lemma 20 *A solution to the recurrence*

$$
\begin{aligned}
L_0 &< \delta + \delta \cdot \alpha M \\
L_k &\le \delta \cdot L_{k-1} + \delta \cdot (\lambda \alpha M - (\lambda - 1)M),
\end{aligned} \tag{78}
$$

for all $k \ge 1$ and $0 \le \delta < 1$, is given by

$$
L_n < \delta^{n+1}(1 + \alpha M) + (1 - \delta^n)\left(\frac{\delta}{1 - \delta}\right)(\lambda \alpha M - (\lambda - 1)M), \quad \textit{for all } n \ge 0. \tag{79}
$$

Proof: The proof is by induction on n.

Base Case: Holds trivially for $n = 0$.

Induction Hypothesis: Assume that (79) holds for $L_0, \ldots, L_k$.

Induction Step: Using (78), we have

$$
\begin{aligned}
L_{k+1} &\le \delta L_k + \delta(\lambda \alpha M - (\lambda - 1)M) \\
&< \delta^{k+2}(1 + \alpha M) + \delta(1 - \delta^k)\left(\tfrac{\delta}{1-\delta}\right)(\lambda \alpha M - (\lambda - 1)M) + \delta(\lambda \alpha M - (\lambda - 1)M) && \text{(by (79) (induction hypothesis))}
\end{aligned}
$$

$$= \delta^{k+2}(1+\alpha M) + \delta(\lambda\alpha M - (\lambda-1)M)((1-\delta^k)\left(\tfrac{\delta}{1-\delta}\right)+1)$$
$$= \delta^{k+2}(1+\alpha M) + (1-\delta^{k+1})\left(\tfrac{\delta}{1-\delta}\right)((\lambda\alpha M - (\lambda-1)M),$$

which proves the lemma. ■

Lemma 21 $\frac{M(\lambda-\delta-(\lambda-1)\delta^{n+1})+\delta^{n+2}-\delta^{n+1}+1-\delta}{M(\lambda-(\lambda-1)\delta^{n+1}-\delta^{n+2})} \geq \frac{M(\lambda-\delta)+\frac{1}{\lambda}}{\lambda M}$ *holds for all* $n \geq 0, 0 \leq \delta \leq 1/2$*, and* $M \geq 1$*.*

Proof: First, note the following.

$$\begin{array}{lll}
 & \frac{M(\lambda-\delta-(\lambda-1)\delta^{n+1})+\delta^{n+2}-\delta^{n+1}+1-\delta}{M(\lambda-(\lambda-1)\delta^{n+1}-\delta^{n+2})} \geq \frac{M(\lambda-\delta)+\frac{1}{\lambda}}{\lambda M} & \\
\Leftrightarrow & (\lambda^2+1)\delta^{n+2} - (\lambda^2-\lambda+1)\delta^{n+1} + \lambda M\delta^{n+2}(1-\delta) + \lambda^2(1-\delta) - \lambda \quad \geq 0 & \text{, simplifying} \\
\Leftarrow & (\lambda^2+1)\delta^{n+2} - (\lambda^2-\lambda+1)\delta^{n+1} + \lambda\delta^{n+2}(1-\delta) + \lambda^2(1-\delta) - \lambda \quad \geq 0 & \text{, because } M \geq 1 \text{ and } 0 \leq \delta \leq 1/2 \\
\Leftrightarrow & -\lambda\delta^{n+3} + (\lambda^2+\lambda+1)\delta^{n+2} - (\lambda^2-\lambda+1)\delta^{n+1} - \lambda^2\delta + \lambda^2 - \lambda \quad \geq 0 &
\end{array}$$

From this, it suffices to show that $h(\delta,\lambda) \stackrel{\text{def}}{=} -\lambda\delta^{n+3} + (\lambda^2+\lambda+1)\delta^{n+2} - (\lambda^2-\lambda+1)\delta^{n+1} - \lambda^2\delta + \lambda^2 - \lambda \geq 0$, for $0 \leq \delta \leq 1/2$, $n \geq 0$, and $\lambda \geq 2$. The first derivative of $h(\delta,\lambda)$ with respect to λ is given by $h'(\delta,\lambda) = \delta^{n+2}(1-\delta) + (1-\delta^{n+1})(2\lambda(1-\delta)-1)$. Because $\delta \leq 1/2$ and $\lambda \geq 2$, $h'(\delta,\lambda)$ is positive for all λ. Hence, $h(\delta,\lambda)$ is increasing with λ and it suffices to show that $h(\delta,2) \geq 0$ holds. $h(\delta,2) = -2\delta^{n+3}+7\delta^{n+2}-3\delta^{n+1}-4\delta+2 = -(2\delta-1)(\delta^{n+1}(\delta-3)+2)$. Because $-(2\delta-1) \geq 0$, it suffices to show that $g(\delta) \stackrel{\text{def}}{=} \delta^{n+1}(\delta-3)+2$ is at least zero. The first derivative of $g(\delta)$ with respect to δ is given by $g'(\delta) = \delta^n((n+2)\delta - 3(n+1))$. The roots of $g'(\delta)$ are $\delta = 0$ and $\delta = \frac{3(n+1)}{n+2}$. $\frac{3(n+1)}{n+2} \geq \frac{3}{2}$ holds for all $n \geq 0$. Therefore, $g(\delta)$ is either increasing or decreasing for δ in $[0, \frac{1}{2}]$. $g(0) = 2$ and $g(\frac{1}{2})$ lies in $[\frac{3}{4}, 2]$ for all $n \geq 0$. Therefore, $g(\delta)$ is positive in $[0, \frac{1}{2}]$, and hence, $h(\delta) \geq 0$ in that interval. ■

Lemma 12 *Let* $t < t_d$ *be a slot with holes and let* U_j *be a subtask that is scheduled at* t*. Then* $d(U_j) = t+1$ *and* $b(U_j) = 1$*.*

Proof: Let U_j and t be as defined in the statement of the lemma. We first show that $d(U_j)$ cannot be less than $t+1$. Because U_j is scheduled at t and $t \leq t_d - 1$ holds, $d(U_j) < t+1 \Rightarrow d(U_j) < t_d$. Hence, by Lemma 14(e), U_j is scheduled before its deadline, and so, $d(U_j) < t+1$ implies that U_j is scheduled before t which contradicts the fact that U_j is scheduled at t. Therefore,

$$d(U_j) \geq t+1. \tag{80}$$

We now show that $d(U_j) = t+1 \ \wedge\ b(U_j) = 1$ holds. The proof is by contradiction, hence, assume that $b(U_j) = 0 \ \vee\ d(U_j) \neq t+1$ holds. By (80), this implies that $(b(U_j) = 0 \ \wedge\ d(U_j) = t+1) \ \vee\ d(U_j) > t+1$. Let U_k be the successor, if any, of U_j. By Lemma 3, if $d(U_j) > t+1$ holds, then $r(U_k) \geq d(U_j) - 1 > t$ holds, and if $b(U_j) = 0 \ \wedge$ $d(U_j) = t+1$ holds, then $r(U_k) \geq d(U_j) = t+1$ holds. Thus, in both cases, $r(U_k) \geq t+1$ holds. Because U_j is scheduled at t in $\mathcal{S}$, U_k is scheduled at or after $t+1$. Hence, by Lemma 14(d), $e(T_k) \geq t+1$ holds. Now, if U_j is removed, then because there is at least a hole in t, by Lemma 6, only U_j's successor, U_k, can shift to t. However, if U_k exists, then its eligibility time is at least $t+1$, and hence, U_k cannot shift to t. Thus, regardless of whether U_k exists, the removal of U_j will not cause any subtask scheduled after t to shift to t, and hence, will not prevent the deadline miss at t_d. This contradicts (T2). Hence, our assumption is incorrect. The lemma follows. ■