Tardiness Bounds for Fixed-Priority Global Scheduling without Intra-Task Precedence Constraints*

Sergey Voronov, University of North Carolina at Chapel Hill, rdkl@cs.unc.edu
James H. Anderson, University of North Carolina at Chapel Hill, anderson@cs.unc.edu
Kecheng Yang, Texas State University, yangk@txstate.edu

Feb 2021

Abstract

Fixed-priority multiprocessor schedulers are often preferable to dynamic-priority ones because they entail less overhead, are easier to implement, and enable certain tasks to be favored over others. Under global fixed-priority (G-FP) scheduling, as applied to the standard sporadic task model, response times for low-priority tasks may be unbounded, even if the total task system utilization is low. In this paper, it is shown that this negative result can be circumvented if different jobs of the same task are allowed to execute in parallel. In particular, a response-time bound is presented for task systems that allow intra-task parallelism. This bound merely requires that the total utilization does not exceed the overall processing capacity—individual task utilizations need not be further restricted. This result implies that G-FP is optimal for scheduling soft real-time tasks that require bounded tardiness, if intra-task parallelism is allowed.

1 Introduction

Since the multicore revolution, the focus of real-time scheduling research has shifted from uniprocessors to multiprocessors. In work on this topic, the global earliest-deadline-first (G-EDF) and global fixed-priority (G-FP) schedulers have both been widely studied (e.g., [3, 5–7, 9, 19]). Although neither is optimal for scheduling hard real-time (HRT) systems where every deadline must be met, both preemptive and non-preemptive G-EDF are optimal for scheduling soft real-time (SRT) sporadic task systems that only require bounds on deadline *tardiness* [12]. That is, under each of these schedulers, deadlines can be missed by only a bounded amount of time for any feasible task system. *Feasible* in this context means that the underlying platform is not over-utilized, and no task over-utilizes a single processor [11].

Unfortunately, this SRT-optimality result does not extend to G-FP, as feasible task systems exist for which tardiness under it can increase without bound; this was shown previously for preemptive G-FP [11] and is shown herein for non-preemptive G-FP. This non-optimality result is regrettable because, in comparison to G-EDF, G-FP entails less overhead, is easer to implement, and enables certain tasks to be favored over others. Given this negative result, if certain tasks need to be prioritized over others, an obvious alternative would be to use a partitioning scheme instead. However, such schemes are also not optimal and can cause system capacity loss due to bin-packing-related issues.

*Work supported by NSF grants CNS 1409175, CPS 1446631, CNS 1563845, and CNS 1717589, ARO grant W911NF-17-1-0294, and funding from General Motors.

In this paper, we consider a different option: employing a relaxed variant of the standard sporadic task model in which successive jobs of the same task may execute in parallel. We are motivated to consider this relaxed model because of the nature of the counterexamples used to show the non-optimality of G-FP. In devising such counterexamples, the goal is to ensure that a certain low-priority task is unable to make use of processors made available to it in parallel, thereby causing its response times to grow without bound.

This relaxed task model has in fact been considered previously in work directed at using G-EDF in HRT [4] and SRT systems [14, 29]. The latter work showed that allowing intra-task parallelism enables much lower tardiness bounds to be derived. Following [29], we call this relaxed model the *npc-sporadic ("no precedence constraints") task model*. Under the npc-sporadic task model, G-EDF precludes response times from growing unboundedly, even if a task's execution time exceeds its period. All that is required is that the entire platform is not over-utilized—this is the only condition needed for SRT feasibility under this model.

This paper expands upon work directed at the npc-sporadic task model by considering the behavior of G-FP under this model. We show that, like G-EDF, G-FP ensures bounded response times for any feasible npc-sporadic task system. We elaborate on this result below, after first taking a closer look at the npc-sporadic model.

Applying the npc-sporadic task model. For the npc-sporadic task model to be applicable, successive jobs of the same task must be able to execute independently. Additionally, it must be acceptable for such jobs to produce output out of order; this tends to be a lesser concern that can be dealt with via buffering (recall that our focus here is applications that can tolerate some tardiness).

Prior papers directed at G-EDF under the npc-sporadic task model mention several example applications that meet these requirements [14, 29]. A particularly compelling use case is computer-vision (or radar) object detection [29]. In contrast to object tracking, object detection may be performed on each frame of video independently. In recent work pertaining to real-time computer vision [31], this use case was considered in detail. In that work, it was shown that allowing intra-task parallelism enables *dramatically* improved response-time bounds for object detection.

Contributions. We consider the scheduling of npc-sporadic task systems on an identical multiprocessor platform under preemptive G-FP. We derive a response-time bound that shows that preemptive G-FP guarantees bounded response times (and hence tardiness) for any feasible npc-sporadic task system; that is, *preemptive G-FP is SRT-optimal*. We also show that our derived response-time bound is asymptotically tight.

This bound tends to grow as the core count and the number of higher-priority tasks increase. Thus, lower tardiness can be guaranteed by partitioning tasks among clusters of cores and scheduling globally only within a cluster. Using a clustered approach lessens tardiness at the expense of impinging on schedulability due to bin-packing-related issues. To elucidate this tradeoff, we conducted an experimental schedulability study in which different cluster sizes were considered on a 16-core platform. We found that using clusters of size four typically enabled relative tardiness bounds that were 60% of those under global scheduling with hardly any impact on schedulability. (A task's *relative tardiness* is given by its tardiness divided by its period.) In our experiments, we also compared relative tardiness bounds obtained from our analysis vs. observed average relative tardiness. We found that bounds for task sets with high total utilization tended to be four to ten times larger than observed relative tardiness.

Per-task tardiness bounds depend on the prioritization of the task system; a lower tardiness bound can be achieved with an appropriate choice of prioritization function. We evaluated several strategies for the priority ordering with respect to tardiness. We found that three of

these strategies significantly outperform the others, and considered these in more detail. In our evaluation, the correct choice of prioritization function ensured up to five times lower average tardiness

We present our tardiness analysis by initially focusing on the preemptive G-FP scheduler. However, as discussed later, this analysis can be adjusted to apply to non-preemptive G-FP, as well as to a further generalization of G-FP that employs preemption thresholds, and, in fact, to any work-conserving global scheduler. Thus, *non-preemptive G-FP is SRT-optimal as well* with respect to npc-sporadic task systems.

The results of this paper establish a rare context under which fixed-priority real-time scheduling is optimal in some sense. To our knowledge, the only other context where such a result has been shown is the uniprocessor scheduling of synchronous implicit-deadline periodic tasks with harmonic periods.

This work extends a paper [27] previously published in the Proceedings of the 26th International Conference on Real-Time Networks and Systems (RTNS 2018). In addition to the slightly modified original material, the extended version considers non-preemptive fixed-priority scheduling, generalized fixed-priority scheduling (G-FP with preemption thresholds), and task system prioritizations. It also includes an expanded experimental evaluation.

Paper organization. In the rest of the paper, we provide needed background (Sec. 2), prove some preliminary lemmas (Sec. 3), derive the response-time bound that is our main contribution (Sec. 4), establish its tightness (Sec. 5), consider task system prioritizations (Sec. 6), extend our results for the non-preemptive case (Sec. 7), as well as G-FP with preemption thresholds (Sec. 8) and any work-conserving scheduler (Sec. 9), present our experimental results (Sec. 10), and conclude (Sec. 11).

2 System Model

Task model. We consider the SRT scheduling of a system τ of n implicit-deadline npc-sporadic tasks, $\tau_1, \ldots, \tau_n$, on platform π with m identical unit-speed processors, $\pi_1, \ldots, \pi_m$. The npc-sporadic task model considered in this paper differs from the standard sporadic task model by relaxing intra-task precedence constraints: any two jobs, ready for execution, may be scheduled at the same time, even if they are produced by the same task. We use the following notation (we assume familiarity with terms commonly used in work on real-time scheduling): $C_i > 0$ denotes the worst-case execution time of task τ_i, T_i denotes its period, and $u_i = C_i/T_i$ denotes its utilization; $J_{i,j}$ denotes the j^{th} job released by τ_i, where $j \geq 1$, and $C_{i,j}$ denotes $J_{i,j}$'s actual execution time, which may be less than C_i. If a job is released at time t_r, has a deadline a time t_d, and completes at time t_c, then its *response time* is $t_c - t_r$ and its *tardiness* is $\max(0, t_c - t_d)$. We also assume that time is continuous. We denote the overall system utilization by $U = \sum_{i=1}^{n} u_i$, the total utilization of tasks $\tau_1, ..., \tau_\ell$ by $U_\ell = \sum_{i=1}^{\ell} u_i$, and the total utilization of all tasks in a specified set α by $U_\alpha = \sum_{\tau_i \in \alpha} u_i$.

Task constraints. Our objective is to derive a response-time bound for a task τ_k by focusing on a job of interest $J_{k,d}$. Note that if τ_k's response times are bounded, then its tardiness is bounded as well. If U exceeds the platform capacity of m, then at least one task will clearly have unbounded response times if all tasks release jobs as soon as possible and every job executes for its worst-case execution time. Therefore, we assume $U \leq m$. However, unlike the traditional sporadic task model, we do not require $u_i \leq 1$, which is necessary for bounded response times under that model but not under the npc-sporadic task model. Under the latter

model, a scheduler that can ensure bounded tardiness for any task system for which $U \leq m$ holds is *SRT-optimal*.

Scheduler. The main focus of this paper is preemptive G-FP (Secs. 4–6), but we will also consider its non-preemptive variant (Sec. 7) as well as a family of variants defined by *preemption thresholds* (Sec. 8). Furthermore, we will show that our results can be extended to any work-conserving scheduler (Sec. 9). To reduce redundancy, we begin by providing definitions (this section) and base lemmas (Sec. 3) for a broad class of schedulers. Scheduler-specific lemmas and theorems are presented in later sections. In this paper, we consider only schedulers that satisfy the following three assumptions:

SH1: The scheduler is global and work-conserving.

SH2: Ready jobs of the same npc-sporadic task are prioritized against each other on a FIFO basis.

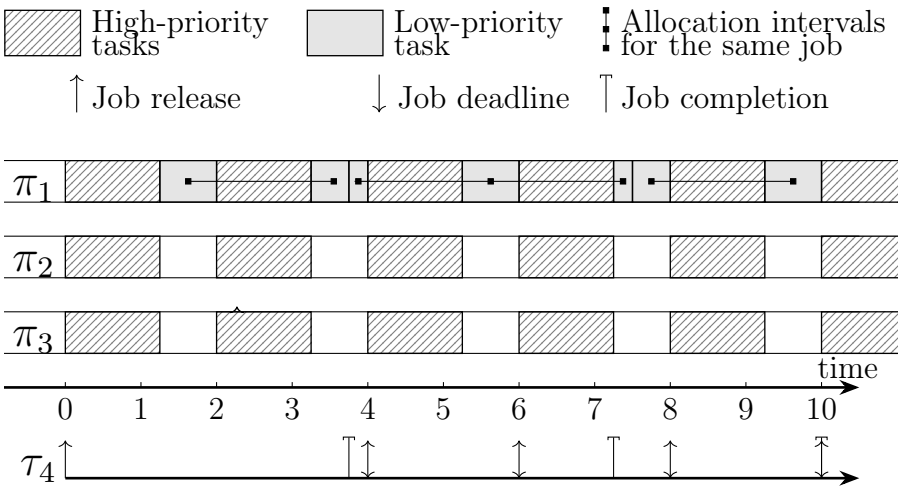
SH3: A job can be scheduled only a finite number of times within any finite time interval.

Assumption SH1 means that every job can be scheduled on every processor, and no processor can be idle, if there is a non-completed job. Assumption SH2 is implicit in the conventional sporadic task model (which precludes a job from starting until the previous job of the same task completes). Assumption SH3 reflects the practical reality that an "implementable" scheduler cannot preempt a job infinitely often. Informally speaking, Assumptions SH1 and SH2 ensure that every job will eventually be completed because the system is not over-utilized.

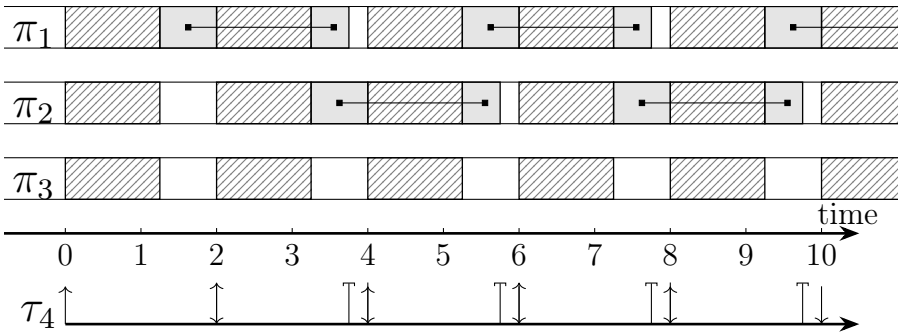
Sporadic vs. npc-sporadic task models. Although the potential for conventional sporadic tasks to have unbounded response times under preemptive G-FP has been shown previously [11], we provide examples illustrating this behavior below for both preemptive and non-preemptive G-FP to highlight various differences between the npc-sporadic and sporadic task models.

Example 1 (preemptive G-FP)**.** *Consider a task system with $m+1$ periodic tasks, each with a worst-case execution time of $1+\varepsilon$ and a period of 2 time units. Under the conventional sporadic model, the response time of the lowest-priority task is unbounded because an allocation of only $1-\varepsilon$ time units is available every 2 time units, while the task requires $1+\varepsilon$ time units, as illustrated in Fig. 1a for $m = 3$. The total utilization of this system is $(1+\varepsilon)(m+1)/2$, which approaches $(m+1)/2$ (roughly half-utilizing the platform) as $\varepsilon \to 0$. In contrast, under the npc-sporadic model, applying Theorem 1 in Sec. 4 to the task system in Fig. 1a yields a response-time bound for the lowest-priority task of 3.5 for small ε (its exact response time is $3+3\varepsilon$). A schedule for this case is shown in Fig. 1b.*

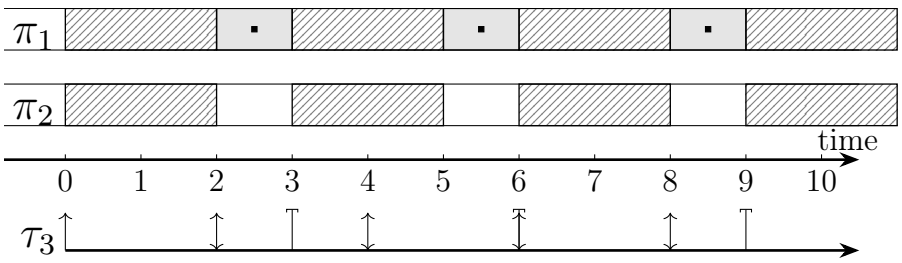
Example 2 (non-preemptive G-FP)**.** *Consider a task system with three periodic tasks, to be scheduled on two processors, with $(C_1, T_1) = (2, 3)$, $(C_2, T_2) = (2, 3)$, and $(C_3, T_3) = (1, 2)$. Despite the non-preemptivity of the scheduler, each job of τ_3 (the lowest-priority task) completes in one time unit, as illustrated in Fig. 1c, and thus never non-preemptively blocks any jobs from τ_1 and τ_2. The time available to τ_3 on each processor coincides, so under the conventional sporadic model, its response time must increase without bound, as illustrated. In contrast, under the npc-sporadic model, where intra-task parallelism is allowed, its response time is bounded, as implied by Fig. 1d.*



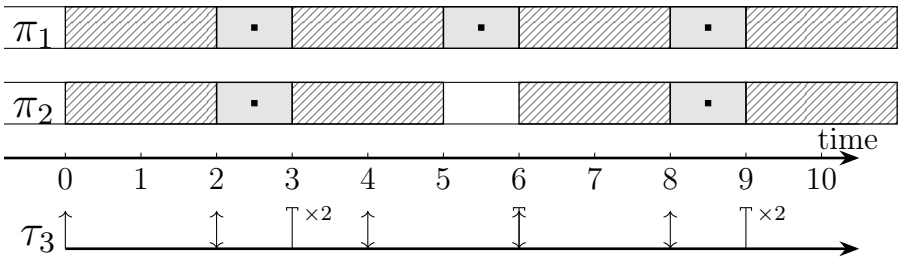
(a) Ex. 1, sporadic tasks, and preemptive G-FP.



(b) Ex. 1, npc-sporadic tasks, and preemptive G-FP.



(c) Ex. 2, sporadic tasks, and non-preemptive G-FP.



(d) Ex. 2, npc-sporadic tasks, and non-preemptive G-FP.

Figure 1: Schedules for the task systems in Exs. 1 and 2.

Definitions. In considering the problem of deriving response-time bounds, we make use of some additional terms and notation, which we introduce next.

Definition 1. We let $r_{i,j}$ and $R_{i,j}$ denote the release and response times, respectively, of job $J_{i,j}$. For conciseness, we sometimes use r and R in reference to the job of interest instead of $r_{k,d}$ and $R_{k,d}$ when there is no ambiguity.

Definition 2. At time t, job $J_{i,j}$ is *ready* if $t \geq r_{i,j}$ and it has not completed yet. If $t < r_{i,j}$, then $J_{i,j}$ is *unreleased*.

As in [13], we use the concept of lag, which we define by considering an "ideal" platform π' consisting of n processors, $\pi'_1, \ldots, \pi'_n$, with speeds $u_1, \ldots, u_n$, respectively. A processor's speed corresponds to the job execution rate on it. Note that such a speed might differ from 1.0. In the ideal schedule, each task τ_i only executes jobs on processor π'_i with speed u_i. Under the npc-sporadic task model with implicit deadlines, every job executes in the ideal schedule from its release until its completion without interference from other jobs or tasks (different tasks run on different processors). Note that if $C_{i,j} < C_i$ holds, then $J_{i,j}$ completes in the ideal schedule before its deadline, whereas, if $C_{i,j} = C_i$ holds, then it completes exactly at its deadline. Thus, at most one job from every task is scheduled at any time in the ideal schedule.

Definition 3. We denote as $\mathcal{I}$ the ideal schedule of the task set τ on π' as described above. Note that $\mathcal{I}$ is a hypothetical schedule that is used only for proofs (it does not exist in reality because it may require processors that differ from those in the given platform, but $\mathcal{I}$ is well defined). Also, we denote as $\mathcal{S}$ the *canonical* schedule produced by the considered scheduler on the actual platform π with m unit-speed processors; a canonical schedule is a schedule that satisfies several scheduler-specific assumptions and is defined for every scheduler later. These assumptions can be specified later because we do not use any of them within Secs. 2 and 3.

Definition 4. For a given schedule $\mathcal{H}$ (either $\mathcal{I}$ or $\mathcal{S}$) at a given time instant t, we define function $\mathsf{sched}(\mathcal{H}, t, J_{i,j})$ such that $\mathsf{sched}(\mathcal{H}, t, J_{i,j}) = s$ if $J_{i,j}$ is scheduled on some processor of speed s, and $\mathsf{sched}(\mathcal{H}, t, J_{i,j}) = 0$ if $J_{i,j}$ is not scheduled on any processor.

We also define $\mathcal{A}(\mathcal{H}, t_1, t_2, J_{i,j})$, the overall processor capacity allocated to job $J_{i,j}$ in $\mathcal{H}$ within the interval $[t_1, t_2)$, as follows,

$$\mathcal{A}(\mathcal{H}, t_1, t_2, J_{i,j}) = \int_{t_1}^{t_2} \mathsf{sched}(\mathcal{H}, t, J_{i,j}) dt.$$

Because $\mathsf{sched}(\mathcal{H}, \cdot, J_{i,j})$ is a piecewise constant function, $\mathcal{A}(\mathcal{H}, 0, \cdot, J_{i,j})$ is continuous (by Assumption SH3, the value of $\mathsf{sched}(\mathcal{H}, \cdot, J_{i,j})$ changes its values a finite number of times within $[0, t]$). To aid in expressing other needed formulas, we also define

$$\mathcal{A}(\mathcal{H}, t_1, t_2, \tau_i) = \sum_j \mathcal{A}(\mathcal{H}, t_1, t_2, J_{i,j}), \text{and}$$

$$\mathcal{A}(\mathcal{H}, t_1, t_2, \tau) = \sum_{\tau_i \in \tau} \mathcal{A}(\mathcal{H}, t_1, t_2, \tau_i).$$

Definition 5. The lag for job $J_{i,j}$ is defined as $\mathsf{lag}(J_{i,j}, t) = \mathcal{A}(\mathcal{I}, 0, t, J_{i,j}) - \mathcal{A}(\mathcal{S}, 0, t, J_{i,j})$.

Example 3 (functions $\mathsf{sched}, \mathcal{A}, \mathsf{lag}$)**.** *Consider a job $J_{1,1}$ in a schedule $\mathcal{S}$ with other jobs, where $C_{1,1} = 3, T_1 = D_1 = 4$, and $u_1 = 0.75$. Inset (a) of Fig. 2 shows how this job is scheduled in some actual schedule. Inset (b) shows the corresponding values of $\mathsf{sched}(\mathcal{I}, t, J_{1,1})$ and $\mathsf{sched}(\mathcal{S}, t, J_{1,1})$. Inset (c) depicts $\mathsf{lag}(t, J_{1,1})$ as a function of time.*

Definition 6. The lag for task τ_i is defined as $\mathsf{Lag}(\tau_i, t) = \sum_j \mathsf{lag}(J_{i,j}, t)$, which is equivalent to $\mathcal{A}(\mathcal{I}, 0, t, \tau_i) - \mathcal{A}(\mathcal{S}, 0, t, \tau_i)$.

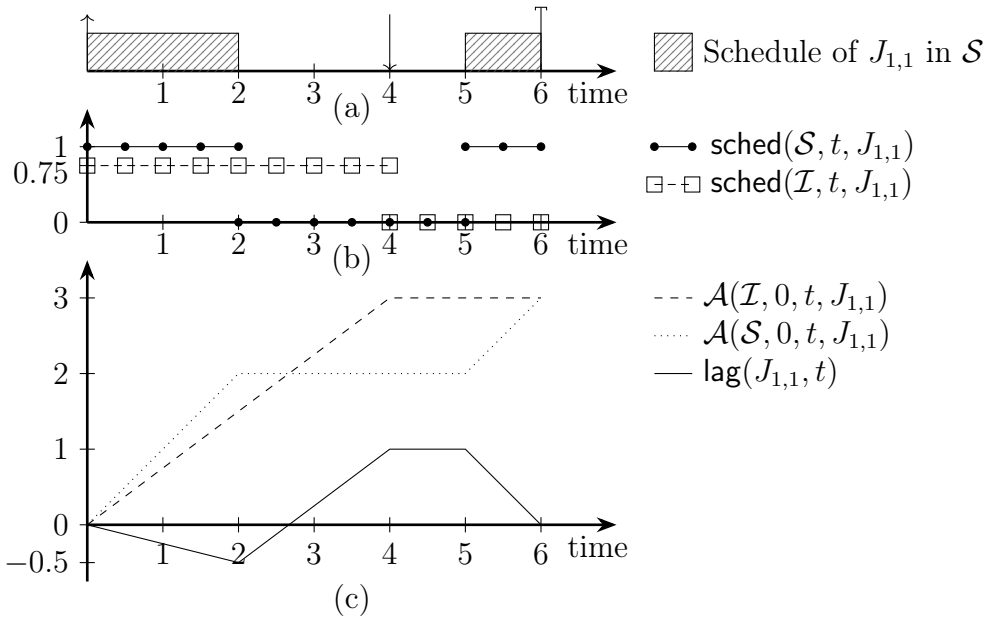


Figure 2: Ex. 3 reference picture.

As in Def. 4, the Lag of τ_i can be equivalently defined as

$$\mathsf{Lag}(\tau_i, t) = \int_0^t \left(\sum_j \mathsf{sched}(\mathcal{I}, t, J_{i,j}) - \sum_j \mathsf{sched}(\mathcal{S}, t, J_{i,j}) \right) dt.$$

Definition 7. We define the total lag of the entire task system as $\mathsf{LAG}(t) = \sum_i \mathsf{Lag}(\tau_i, t) = \sum_i \sum_j \mathsf{lag}(J_{i,j}, t)$, which is equivalent to $\mathcal{A}(\mathcal{I}, 0, t, \tau) - \mathcal{A}(\mathcal{S}, 0, t, \tau)$. We also define $\mathsf{LAG}(\alpha, t) = \sum_{\tau_i \in \alpha} \mathsf{Lag}(\tau_i, t)$ (the total lag of the tasks in α). Thus, $\mathsf{LAG}(t)$ is an abbreviation of $\mathsf{LAG}(\tau, t)$.

Note that we use function names lag for the lag of a job, Lag for the lag of a task, and LAG for the total lag of a set of tasks.

3 Preliminary Bounds on lag, Lag, and LAG Functions

In this section, we present a number of lemmas pertaining to the lag-based functions at the job, task, and system level. In Sec. 3.1, we derive a few properties of the job lag function and a lower bound on the task Lag function. In Sec. 3.2, we derive an upper bound on the total system LAG function. Note that these lemmas are proven assuming only Assumptions SH1-SH3. Therefore, we are free to use them later in the context of any scheduler that satisfies Assumptions SH1-SH3.

Proof overview. To emphasize the importance of the lemmas proved in this section, we present here an overview of the response-time bound computation for preemptive G-FP. Fortunately, the same approach can be used for all considered G-FP variants with minor modifications.

The main idea behind the npc-sporadic task model is that a task can occupy multiple available processors if it has multiple ready jobs. If $J_{k,d}$'s response time is high enough, then there are at least m non-completed jobs (the job of interest and the following jobs of τ_k). Thus, until the completion of $J_{k,d}$, no processors are idle. Although this approach works for

preemptive G-FP (Sec. 4), we modify it later to handle more complicated schedulers (Sec. 7 and 8).

We exploit this proof idea in three major steps. Firstly, we estimate the total system LAG at times r and $r + R$ in Lemmas 7 and 8 assuming $\alpha = \tau$. Because the job of interest is uncompleted within the interval $[r, r + R)$, either R is small or m cores are busy within most of the interval. In both cases, the LAG increase over the interval is relatively small, which implies an upper bound for $\mathsf{LAG}(r + R)$. Secondly, we compute a lower bound on $J_{k,d}$'s Lag at time $r + R$ in Lemma 9; this Lag value depends on R. Finally, we obtain an upper bound on R in Theorem 1 with $\mathsf{LAG}(r + R) = \sum_i \mathsf{Lag}(\tau_i, r + R)$ using Lemma 4 to bound the Lag of all tasks other than τ_k.

3.1 Task Lag Lower Bound

The general property of Lag for a single task that we establish in this subsection is formulated in Lemma 4. To prove this lemma, we first prove several lemmas concerning job lag.

Lemma 1. *For any time t before the release of job $J_{i,j}$ or after its completion in both schedules $\mathcal{I}$ and $\mathcal{S}$, $\mathsf{lag}(J_{i,j}, t) = 0$.*

Proof. If $J_{i,j}$ is unreleased, then by Def. 4, $\mathcal{A}(\mathcal{I}, 0, t, J_{i,j})$ and $\mathcal{A}(\mathcal{S}, 0, t, J_{i,j})$ are both 0. If $J_{i,j}$ has completed in both schedules $\mathcal{I}$ and $\mathcal{S}$, then by Def. 4, $\mathcal{A}(\mathcal{I}, 0, t, J_{i,j}) = \mathcal{A}(\mathcal{S}, 0, t, J_{i,j}) = C_{i,j}$. In both cases, by Def. 5, $\mathsf{lag}(J_{i,j}, t) = 0$. □

Lemma 2. *If $t \geq r_{i,j} + T_i$, then $\mathsf{lag}(J_{i,j}, t) \geq 0$.*

Proof. If $t \geq r_{i,j} + T_i$, then $\mathcal{A}(\mathcal{I}, 0, t, J_{i,j}) = C_{i,j}$, because $J_{i,j}$ completes in $\mathcal{I}$ by the end of its period. Also, $\mathcal{A}(\mathcal{S}, 0, t, J_{i,j}) \leq C_{i,j}$ for every time instant. By Def. 5, the lemma follows. □

The next lemma provides bounds on a single job's lag.

Lemma 3. $\min(0, (u_i - 1)C_i) \leq \mathsf{lag}(J_{i,j}, t) \leq C_i$.

Proof. According to Def. 5,

$$\begin{aligned}
\mathsf{lag}(J_{i,j}, t) &= \mathcal{A}(\mathcal{I}, 0, t, J_{i,j}) - \mathcal{A}(\mathcal{S}, 0, t, J_{i,j}) \\
&\leq \mathcal{A}(\mathcal{I}, 0, t, J_{i,j}) \\
&\leq \{\text{by Def. 4}\} \\
&\quad C_{i,j} \\
&\leq C_i,
\end{aligned}$$

proving the stated upper bound. In the rest of the proof, we focus on proving the stated lower bound. Let $f = C_{i,j}/C_i$. Note that $f \cdot T_i = (C_{i,j}T_i)/C_i = C_{i,j}/u_i$, which is the exact amount of time that is needed for $J_{i,j}$'s completion in the ideal schedule $\mathcal{I}$. To simplify the proof, we split the time line into three intervals: "before $J_{i,j}$'s release" : $[0, r_{i,j})$; "$J_{i,j}$ is scheduled in $\mathcal{I}$": $[r_{i,j}, r_{i,j} + f \cdot T_i)$; and "$J_{i,j}$ has completed in $\mathcal{I}$" : $[r_{i,j} + f \cdot T_i, \infty)$. We consider each interval separately.

Case 1: $t \in [0, r_{i,j})$. By Lemma 1, $\mathsf{lag}(J_{i,j}, t) = 0$.

Case 2: $t \in [r_{i,j}, r_{i,j} + f \cdot T_i)$. For any such t, we define $t' = t - r_{i,j}$. By definition, $t' \in [0, f \cdot T_i)$. Since $J_{i,j}$ is released at time $r_{i,j}$ and is continuously scheduled in $\mathcal{I}$ (by Def. 3) during $[r_{i,j}, r_{i,j} + f \cdot T_i)$ on a processor with speed u_i, $\mathcal{A}(\mathcal{I}, 0, t, J_{i,j}) = u_i t'$.

In completing the reasoning for this case, we first dispense with the possibility that $u_i \geq 1$ holds. Because job $J_{i,j}$ is not scheduled in $\mathcal{S}$ before $r_{i,j}$,

$$\mathcal{A}(\mathcal{S}, 0, t, J_{i,j}) = \mathcal{A}(\mathcal{S}, r_{i,j}, t, J_{i,j}) \leq t - r_{i,j} = t'.$$

Thus, if $u_i \geq 1$ holds, we have

$$\mathcal{A}(\mathcal{I}, 0, t, J_{i,j}) = u_i t' \geq t' \geq \mathcal{A}(\mathcal{S}, 0, t, J_{i,j}),$$

which implies that $\mathsf{lag}(J_{i,j}) \geq 0$ holds. In the rest of the proof for Case 2, we consider the remaining possibility: $u_i < 1$.

Let ρ denote the allocation time for $J_{i,j}$ in $\mathcal{S}$ during the subinterval $[r_{i,j}, t) = [r_{i,j}, r_{i,j} + t')$. Then $\rho \leq t'$ and $\rho \leq C_i$ (the maximum execution time for any job of τ_i). Thus, we have

$$\begin{aligned}
\mathsf{lag}(J_{i,j}, t) &= \mathsf{lag}(J_{i,j}, r_{i,j} + t') \\
&= \mathcal{A}(\mathcal{I}, 0, r_{i,j} + t', J_{i,j}) - \mathcal{A}(\mathcal{S}, 0, r_{i,j} + t', J_{i,j}) \\
&= u_i t' - \rho \\
&= (u_i - 1)\rho + (t' - \rho)u_i \\
&\geq \{t' \geq \rho\} \\
&\quad (u_i - 1)\rho \\
&\geq \{u_i - 1 < 0 \text{ and } \rho \leq C_i\} \\
&\quad (u_i - 1)C_i.
\end{aligned}$$

Case 3: $t \in [r_i + f \cdot T_i, \infty)$. By Def. 3 and the definition of f, $J_{i,j}$ is completed at time instant $r_i + f \cdot T_i$ in $\mathcal{I}$, and $\mathcal{A}(\mathcal{I}, 0, t, J_{i,j}) = C_{i,j}$. With $\mathcal{A}(\mathcal{S}, 0, t, J_{i,j}) \leq C_{i,j}$ (the maximal allocation for $J_{i,j}$ in $\mathcal{S}$), we have $\mathsf{lag}(J_{i,j}, t) = \mathcal{A}(\mathcal{I}, 0, t, J_{i,j}) - \mathcal{A}(\mathcal{S}, 0, t, J_{i,j}) \geq C_{i,j} - C_{i,j} = 0$.

By Cases 1-3, $\mathsf{lag}(J_{i,j}, t) \geq \min(0, (u_i - 1)C_i)$. □

Corollary 1. $\mathsf{lag}(J_{i,j}, t) < 0$ *implies* $t \in [r_{i,j}, r_{i,j} + T_i)$.

Proof. In Cases 1 and 3 in the proof of Lemma 3 above, we proved that $\mathsf{lag}(J_{i,j}, t) \geq 0$ holds. Thus, if $\mathsf{lag}(J_{i,j}, t) < 0$ holds, then $t \in [r_{i,j}, r_{i,j} + f \cdot T_i)$ (the interval considered in Case 2) with $f = C_{i,j}/C_i$. Because $f \leq 1$, $[r_{i,j}, r_{i,j} + f \cdot T_i) \subseteq [r_{i,j}, r_{i,j} + T_i)$. □

Our lower bound on job lag can be extended to task Lag.

Lemma 4. $\min(0, (u_i - 1)C_i) \leq \mathsf{Lag}(\tau_i, t)$.

Proof. By Corollary 1, $\mathsf{lag}(J_{i,j}, t) < 0$ may hold only if $J_{i,j}$ is scheduled for execution in $\mathcal{I}$ at time instant $t \in [r_{i,j}, r_{i,j} + T_i)$. By Def. 3, at most one job per task may be scheduled in $\mathcal{I}$ at any time instant. Therefore, in $\sum_j \mathsf{lag}(J_{i,j}, t)$, at most one summand might be less than 0, because the intervals $[r_{i,j}, r_{i,j} + T_i)$ do not overlap for different choices of j due to the definition of an npc-sporadic task. That is, if $\mathsf{lag}(\tau_{i,h}, t) < 0$ holds, then for any $j \neq h$, $\mathsf{lag}(J_{i,j}, t) \geq 0$. Thus, by Lemma 3

$$\mathsf{Lag}(\tau_i, t) = \sum_j \mathsf{lag}(J_{i,j}, t) \geq \mathsf{lag}(\tau_{i,h}, t) \geq \min(0, (u_i - 1)C_i).$$ □

3.2 System LAG Upper Bounds

In Sec. 3.1, we established results about job and task lags. We are now ready to bound the overall system LAG. To do so, we need to analyze precisely how processors schedule different tasks.

We begin by proving that all functions considered in ours proofs are continuous. This property is used in establishing LAG upper bounds.

Lemma 5. $\mathsf{lag}(J_{i,j}, t)$ *and* $\mathsf{Lag}(\tau_i, t)$ *are continuous functions of* t. *Also,* $\forall \alpha \subseteq \tau$, $LAG(\alpha, t)$ *is a continuous function of* t.

Proof. Both $\mathcal{A}(\mathcal{I}, 0, t, J_{i,j})$ and $\mathcal{A}(\mathcal{S}, 0, t, J_{i,j})$ are continuous by definition. Thus, by Def. 5, $\mathsf{lag}(J_{i,j}, t)$ is continuous because

$$\mathsf{lag}(J_{i,j}, t) = \mathcal{A}(\mathcal{I}, 0, t, J_{i,j}) - \mathcal{A}(\mathcal{S}, 0, t, J_{i,j}).$$

Let h be the number of jobs, released by τ_i at or before t. Then $r_{i,h} \leq t < r_{i,h+1}$. By Lemma 1, $\mathsf{lag}(J_{i,j}, t) = 0$ for all $j > h$. Thus,

$$\mathsf{Lag}(\tau_i, t) = \sum_j \mathsf{lag}(J_{i,j}, t) = \sum_{j \leq h} \mathsf{lag}(J_{i,j}, t).$$

$\mathsf{Lag}(\tau_i, t)$ is continuous because it is a sum of a finite number of continuous functions. Similarly, $\mathsf{LAG}(\alpha, t)$ is continuous because

$$\mathsf{LAG}(\alpha, t) = \sum_{\tau_i \in \alpha} \mathsf{Lag}(\tau_i, t).$$

□

The following lemma is used to bound the execution rate of a given task set in the ideal schedule $\mathcal{I}$.

Lemma 6. *For any time interval* $[t_1, t_2)$ *and any task set* α,

$$\mathcal{A}(\mathcal{I}, t_1, t_2, \tau_i) \leq u_i(t_2 - t_1), \text{ and}$$
$$\mathcal{A}(\mathcal{I}, t_1, t_2, \alpha) \leq U_\alpha(t_2 - t_1).$$

Proof. At every time instant in the ideal schedule $\mathcal{I}$, there is at most one job from task τ_i scheduled (see Def. 3). The speed of the processor that schedules τ_i is u_i, while the length of the considered interval is $(t_2 - t_1)$. Thus,

$$\mathcal{A}(\mathcal{I}, t_1, t_2, \tau_i) = \sum_j \mathcal{A}(\mathcal{I}, t_1, t_2, J_{i,j}) \leq u_i(t_2 - t_1), \text{and} \tag{1}$$

$$\begin{aligned}
\mathcal{A}(\mathcal{I}, t_1, t_2, \alpha) &= \sum_{\tau_i \in \alpha} (\mathcal{A}(\mathcal{I}, t_1, t_2, \tau_i)) \\
&\leq \{\text{by (1)}\} \\
&\quad \sum_{\tau_i \in \alpha} u_i(t_2 - t_1) \\
&= \left(\sum_{\tau_i \in \alpha} u_i \right) (t_2 - t_1) \\
&= U_\alpha(t_2 - t_1).
\end{aligned}$$

□

Definition 8. We call a processor *busy* at time instant t if there exists a job scheduled for execution on this processor at time t. Otherwise, the processor is *idle*.

Our upper bound on the LAG of the set of tasks is given by the following lemma.

Lemma 7. *Consider a set of tasks α such that at any time instant in $[0, t')$, for some time instant t', only jobs from tasks in α are scheduled. Then for any $t \in [0, t']$, $\mathsf{LAG}(\alpha, t) \leq (\lceil U_\alpha \rceil - 1)C_{\max}$, where $C_{\max} = \max_{\tau_i \in \alpha} C_i$.*

Proof. To prove this lemma, we split the interval $[0, t')$ into a set of maximal continuous intervals such that the number of busy processors in $\mathcal{S}$ during each interval does not change. More formally, these intervals satisfy the following assumptions (for an interval I):

I1: The number of busy processors in $\mathcal{S}$ during I does not change.

I2: I cannot be extended without violating I1.

Since any finite time interval I contains a finite number of scheduling events by Assumption SH3, I contains a finite number of the intervals from the set just defined.

We now prove the lemma by contradiction: let I_b be the first time interval from the set defined above such that

$$\exists t_2 \in I_b : \mathsf{LAG}(\alpha, t_2) > (\lceil U_\alpha \rceil - 1)C_{\max}. \tag{2}$$

Let t_1 be the beginning of I_b, which implies $t_1 \leq t_2$. To set up deriving a contradiction later, we next show that (3) below holds.

$$\mathsf{LAG}(\alpha, t_1) \leq (\lceil U_\alpha \rceil - 1)C_{\max}. \tag{3}$$

If $t_1 = 0$, then $\mathsf{LAG}(\alpha, 0) = 0$, so (3) holds since $U_\alpha > 0$. On the other hand, if $t_1 \neq 0$, then there exists a preceding interval I_a such that the end of I_a is t_1, and, by the definition of I_b,

$$\forall t \in I_a : \mathsf{LAG}(\alpha, t) \leq (\lceil U_\alpha \rceil - 1)C_{\max}.$$

By Lemma 5, $\mathsf{LAG}(\alpha, t)$ is a continuous function, so

$$\mathsf{LAG}(\alpha, t_1) = \lim_{t \to t_1^-} \mathsf{LAG}(\alpha, t) = \lim_{t \in I_a, t \to t_1^-} \mathsf{LAG}(\alpha, t) \leq (\lceil U_\alpha \rceil - 1)C_{\max},$$

i.e., (3) holds. Thus, we have

$$\mathsf{LAG}(\alpha, t_1) \leq (\lceil U_\alpha \rceil - 1)C_{\max} < \mathsf{LAG}(\alpha, t_2), \tag{4}$$

and the number of busy processors during $[t_1, t_2)$ in the actual schedule $\mathcal{S}$ does not change. Let bp denote this number. The overall allocation given to τ in $\mathcal{S}$ with exactly bp busy processors during the interval $[t_1, t_2)$ is

$$\mathcal{A}(\mathcal{S}, t_1, t_2, \tau) = \mathsf{bp} \cdot (t_2 - t_1). \tag{5}$$

Combining Lemma 6 and (5), we can bound the change of LAG over $[t_1, t_2)$:

$$\begin{aligned}
&\mathsf{LAG}(\alpha, t_2) - \mathsf{LAG}(\alpha, t_1) \\
&= \mathcal{A}(\mathcal{I}, 0, t_2, \tau) - \mathcal{A}(\mathcal{S}, 0, t_2, \tau) - (\mathcal{A}(\mathcal{I}, 0, t_1, \tau) - \mathcal{A}(\mathcal{S}, 0, t_1, \tau)) \\
&= \mathcal{A}(\mathcal{I}, t_1, t_2, \tau) - \mathcal{A}(\mathcal{S}, t_1, t_2, \tau) \\
&\leq \{\text{by Lemma 6 and (5)}\} \\
&\quad U_\alpha(t_2 - t_1) - \mathsf{bp} \cdot (t_2 - t_1) \\
&= (U_\alpha - \mathsf{bp})(t_2 - t_1).
\end{aligned}$$

Thus, by (4), $(U_\alpha - \mathsf{bp})(t_2 - t_1) > 0$. By the definition of t_1 and t_2, $t_1 \leq t_2$, so $U_\alpha - \mathsf{bp} > 0$. Thus,

$$\mathsf{bp} \leq \lceil U_\alpha \rceil - 1. \tag{6}$$

Since $U_\alpha \leq m$, we therefore have $\mathsf{bp} < m$. Thus, there is at least one non-busy processor. By Assumption SH1, the scheduler is work-conserving, so all ready jobs are scheduled, and bp is the number of uncompleted jobs of tasks in α at time t_2^-. We can now derive a bound for $\mathsf{LAG}(\alpha, t_2^-)$:

$$
\begin{aligned}
\mathsf{LAG}(\alpha, t_2^-) &= \sum_i \mathsf{Lag}(\tau_i, t_2^-) \\
&\leq \{\text{by Def. 7 and Lemma 1}\} \\
&\quad \sum_{J_{i,j} \text{ is uncompleted in } \mathcal{I} \text{ or } \mathcal{S} \text{ at } t_2^-} \mathsf{lag}(J_{i,j}, t_2^-) \\
&\leq \sum_{J_{i,j} \text{ is uncompleted in } \mathcal{S} \text{ at } t_2^-} \mathsf{lag}(J_{i,j}, t_2^-) + \sum_{J_{i,j} \text{ is uncompleted in } \mathcal{I} \text{ but not in } \mathcal{S} \text{ at } t_2^-} \mathsf{lag}(J_{i,j}, t_2^-) \\
&\leq \{\text{if } J_{i,j} \text{ is uncompleted in } \mathcal{I} \text{ but not in } \mathcal{S} \text{ then } \mathsf{lag}(J_{i,j}, t_2^-) \leq 0\} \\
&\quad \sum_{J_{i,j} \text{ is uncompleted in } \mathcal{S} \text{ at } t_2^-} \mathsf{lag}(J_{i,j}, t_2^-) \\
&\leq \{\text{by Lemma 3}\} \\
&\quad \sum_{J_{i,j} \text{ is uncompleted in } \mathcal{S} \text{ at } t_2^-} C_i \\
&\leq \{\mathsf{bp} \text{ is the number of uncompleted jobs in } \mathcal{S} \text{ and } C_{\max} \geq C_i\} \\
&\quad \mathsf{bp} \cdot C_{\max} \\
&\leq \{\text{by (6)}\} \\
&\quad (\lceil U_\alpha \rceil - 1) \cdot C_{\max}. \qquad (7)
\end{aligned}
$$

By Lemma 5, $\mathsf{LAG}(\alpha, t_2) = \mathsf{LAG}(\alpha, t_2^-)$, so (7) contradicts (2). This contradiction finishes the proof. □

Note that the previous lemma bounds the total system LAG at any time instant. However, if $J_{k,d}$ has a large response time, we know that other jobs exclusively occupied all processors. Thus, we can use this information to provide a tighter bound for LAG at the specific time instant. We use the following definitions to encapsulate information relevant to the scheduling of $J_{k,d}$.

Definition 9. Let $\mathcal{J} = \{J_{k,d}, J_{k,d+1}, J_{k,d+2}, ...\}$. $\mathcal{J}$ contains the job of interest $J_{k,d}$ and all jobs from τ_k following the job of interest.

Definition 10. Let W denote the overall processor allocation to jobs from $\mathcal{J}$ in $\mathcal{S}$ in the interval $[r, r + R)$. More formally,

$$W = \mathcal{A}(\mathcal{S}, r, r + R, \mathcal{J}).$$

To establish our new bound on LAG, we exploit the following property of a work-conserving scheduler (Assumption SH1): if $J_{k,d}$ is ready and not scheduled at time t, then all processors are busy from t until $J_{k,d}$ is scheduled in $\mathcal{S}$ after t.

Lemma 8. *Consider a set of tasks α such that at any time instant in $[r, r + R)$ either $J_{k,d}$ is scheduled or m jobs from tasks in α are scheduled, and $\tau_k \in \alpha$. Then,*

$$\mathsf{LAG}(\alpha, r + R) \leq \mathsf{LAG}(\alpha, r) + mC_k + (U_\alpha - m)R - W.$$

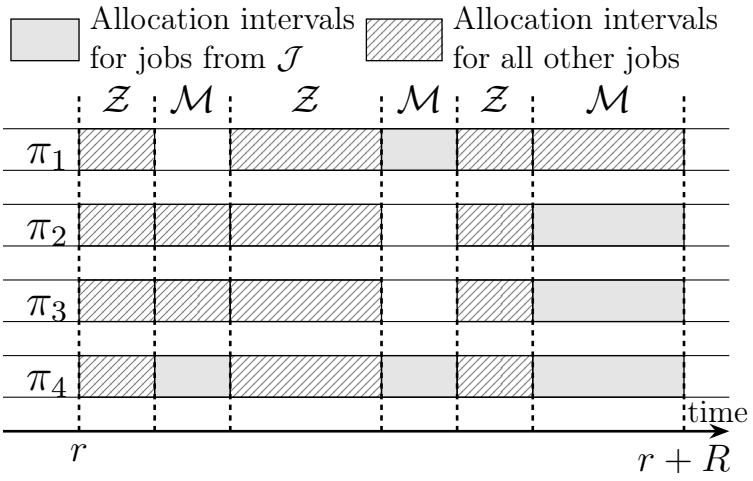


Figure 3: Example partitioning of $[r, r + R)$ into intervals (Lemma 8 reference).

Proof. To prove this lemma, we split the interval $[r, r + R)$ into a set of maximal continuous intervals such that the number of jobs from $\mathcal{J}$ scheduled in $\mathcal{S}$ during each interval does not change. More formally, these intervals satisfy the following assumptions (for an interval I):

I1: The number of scheduled jobs from $\mathcal{J}$ in $\mathcal{S}$ during I does not change.

I2: I cannot be extended without violating I1.

By Assumption SH3, we have a finite number of such intervals (since they are defined by a finite number of scheduling events within $[r, r + R)$). Let us define the set of interval starting points as $\{t_0 = r, t_1, ..., t_{h-1}\}$, with an additional $t_h = r + R$. Also, let I_e denote the interval $[t_e, t_{e+1})$ for $0 \leq e \leq h - 1$.

Let $\mathcal{Z}$ be the set of all intervals for which no jobs from $\mathcal{J}$ are scheduled in $\mathcal{S}$. Let $\mathcal{M}$ be the set of all intervals for which at least one job from $\mathcal{J}$ is scheduled in $\mathcal{S}$. It is clear that any interval from $\mathcal{Z}$ is disjoint from all intervals in $\mathcal{M}$, while the union of all intervals from $\mathcal{Z} \cup \mathcal{M}$ is $[r, r + R)$ by the definition of α. An example of such intervals is given in Fig. 3.

By the definition of $\mathcal{Z}$, no jobs from $\mathcal{J}$ are scheduled for any $I_e \in \mathcal{Z}$. Because there is a ready job $J_{k,d}$ that is not scheduled, there are m scheduled jobs for any time instant of I_e. By the definition of α, all m jobs are from tasks in α. Thus,

$$\forall I_e \in \mathcal{Z} : \ \mathcal{A}(\mathcal{S}, t_e, t_{e+1}, \alpha) = m(t_{e+1} - t_e). \tag{8}$$

Let $||\mathcal{Z}||$ (resp., $||\mathcal{M}||$) denote the overall length of all intervals from set $\mathcal{Z}$ (resp., $\mathcal{M}$).

For any $I_e \in \mathcal{M}$ and any $t \in I_e$, at least one job from $\mathcal{J}$ is scheduled. Since, by Assumption SH2, jobs from the same task τ_k are prioritized against each other on a FIFO basis, and $J_{k,d}$ is the first one in $\mathcal{J}$, $J_{k,d}$ should be scheduled at t (and, possibly, some other jobs from $\mathcal{J}$; note that earlier jobs of τ_k are not included in $\mathcal{J}$, which is used to define $\mathcal{M}$). Thus,

$$\begin{aligned} &||\mathcal{M}|| \leq C_k, \text{and} \\ &||\mathcal{Z}|| = |[r, r + R)| - ||\mathcal{M}|| \geq R - C_k. \end{aligned} \tag{9}$$

Moreover, if any job from $\mathcal{J}$ is scheduled at time instant t, then $t \in I_e$ for some $I_e \in \mathcal{M}$. Thus, all processor allocations accounted for in W may happen only during intervals from $\mathcal{M}$. Therefore,

$$\sum_{I_e \in \mathcal{M}} \mathcal{A}(\mathcal{S}, t_e, t_{e+1}, \mathcal{J}) = W. \tag{10}$$

We now can estimate the change in LAG over $[t_0, t_h)$:

$$
\begin{aligned}
&\mathsf{LAG}(t_h) - \mathsf{LAG}(t_0) \\
&= \sum_{e=0}^{h-1} (\mathsf{LAG}(t_{e+1}) - \mathsf{LAG}(t_e)) \\
&= \{\text{by the definitions of } \mathcal{Z} \text{ and } \mathcal{M}\} \\
&\quad \sum_{I_e \in \mathcal{Z}} (\mathsf{LAG}(t_{e+1}) - \mathsf{LAG}(t_e)) + \sum_{I_e \in \mathcal{M}} (\mathsf{LAG}(t_{e+1}) - \mathsf{LAG}(t_e)) \\
&= \{\text{by Def. 5}\} \\
&\quad \sum_{I_e \in \mathcal{Z}} (\mathcal{A}(\mathcal{I}, t_e, t_{e+1}, \alpha) - \mathcal{A}(\mathcal{S}, t_e, t_{e+1}, \alpha)) + \sum_{I_e \in \mathcal{M}} (\mathcal{A}(\mathcal{I}, t_e, t_{e+1}, \alpha) - \mathcal{A}(\mathcal{S}, t_e, t_{e+1}, \alpha)) \\
&= \{\text{by rearranging and (8)}\} \\
&\quad \sum_{I_e \in \mathcal{Z} \cup \mathcal{M}} \mathcal{A}(\mathcal{I}, t_e, t_{e+1}, \alpha) - \sum_{I_e \in \mathcal{Z}} m(t_{e+1} - t_e) - \sum_{I_e \in \mathcal{M}} \mathcal{A}(\mathcal{S}, t_e, t_{e+1}, \alpha) \\
&\leq \{\text{by (10) and the definitions of } \mathcal{Z} \text{ and } \mathcal{M}\} \\
&\quad \sum_{I_e \in \mathcal{Z} \cup \mathcal{M}} \mathcal{A}(\mathcal{I}, t_e, t_{e+1}, \alpha) - m||\mathcal{Z}|| - W \\
&\leq \{\text{by Lemma 6}\} \\
&\quad ||\mathcal{Z} \cup \mathcal{M}||U_\alpha - m||\mathcal{Z}|| - W \\
&\leq \{\text{by (9) and } ||\mathcal{Z} \cup \mathcal{M}|| = |[r, r+R)|\} \\
&\quad RU_\alpha - m(R - C_k) - W \\
&= mC_k + (U_\alpha - m)R - W. \qquad (11)
\end{aligned}
$$

By rearranging (11) with $t_h = r + R$, and $t_0 = r$, we obtain a bound for $\mathsf{LAG}(\alpha, r + R)$:

$$\mathsf{LAG}(\alpha, r + R) \leq \mathsf{LAG}(\alpha, r) + mC_k + (U_\alpha - m)R - W. \qquad \square$$

Using Def. 10, we can compute the exact value of $\mathsf{Lag}(\tau_k, r + R)$ by an alternative approach compared to that used to prove Lemma 4. Note that no job from $\mathcal{J}$ is ready during $[0, r)$. Thus, $\mathcal{A}(\mathcal{S}, 0, r, \mathcal{J}) = 0$, and

$$
\begin{aligned}
W &= \mathcal{A}(\mathcal{S}, r, r + R, \mathcal{J}) \\
&= \mathcal{A}(\mathcal{S}, 0, r, \mathcal{J}) + \mathcal{A}(\mathcal{S}, r, r + R, \mathcal{J}) \\
&= \mathcal{A}(\mathcal{S}, 0, r + R, \mathcal{J}). \qquad (12)
\end{aligned}
$$

Lemma 9. *If $J_{k,d}$ has an execution time of C_k, then*

$$\mathsf{Lag}(\tau_k, r + R) \geq \min(C_k, u_k R) - W. \qquad (13)$$

Moreover, if all jobs in $\mathcal{J}$ have execution time equal to C_k and are released periodically with period T_k, then

$$\mathsf{Lag}(\tau_k, r + R) = u_k R - W. \qquad (14)$$

Proof. The task τ_k has a worst-case execution time of C_k, so all jobs $J_{k,d-1}, J_{k,d-2}, ..., J_{k,1}$ have execution times not higher than C_k. By Assumption SH2, these jobs are prioritized over $J_{k,d}$

and completed in $\mathcal{S}$ at or before $r + R$. By Def. 3, these jobs are completed in $\mathcal{I}$ at or before r. Thus, by Lemma 1, $\forall j < d : \mathsf{lag}(J_{k,j}, r + R) = 0$.

$$\begin{aligned}
\mathsf{Lag}(\tau_k, r + R) &= \sum_{j=1}^{\infty} \mathsf{lag}(J_{k,j}, r + R) \\
&= \sum_{j=1}^{d-1} \mathsf{lag}(J_{k,j}, r + R) + \sum_{j=d}^{\infty} \mathsf{lag}(J_{k,j}, r + R) \\
&= \{\forall j < d : \mathsf{lag}(J_{k,j}, r + R) = 0\} \\
&\quad \sum_{j=d}^{\infty} \mathsf{lag}(J_{k,j}, r + R) \\
&= \sum_{j=d}^{\infty} (\mathcal{A}(\mathcal{I}, 0, r + R, J_{k,j}) - \mathcal{A}(\mathcal{S}, 0, r + R, J_{k,j})) \\
&= \left(\sum_{j=d}^{\infty} \mathcal{A}(\mathcal{I}, 0, r + R, J_{k,j}) \right) - \mathcal{A}(\mathcal{S}, 0, r + R, \mathcal{J}) \\
&= \{\text{by (12)}\} \\
&\quad \sum_{j=d}^{\infty} \mathcal{A}(\mathcal{I}, 0, r + R, J_{k,j}) - W \qquad (15)
\end{aligned}$$

Note that $\mathcal{A}(\mathcal{I}, 0, r + R, J_{k,j}) \geq 0$ for any $J_{k,j}$ because we cannot allocate a negative amount of execution time to a job. Thus, by (15),

$$\mathsf{Lag}(\tau_k, r + R) \geq \mathcal{A}(\mathcal{I}, 0, r + R, J_{k,d}). \qquad (16)$$

We can compute $\mathcal{A}(\mathcal{I}, 0, r + R, J_{k,d})$ by considering two cases: $R < T_k$ and $R \geq T_k$. If $R < T_k$, then $J_{k,d}$ is executed for R time units in $\mathcal{I}$ within $[r, r + R)$. Otherwise $J_{k,d}$ is executed in $\mathcal{I}$ for T_k time units. Thus,

$$\begin{aligned}
A(\mathcal{I}, r, r + R, J_{k,d}) &= A(\mathcal{I}, 0, r + R, J_{k,d}) - A(\mathcal{I}, 0, r, J_{k,d}) \\
&= \{J_{k,d} \text{ is released at } r\} \\
&\quad A(\mathcal{I}, 0, r + R, J_{k,d}) \\
&= \min(u_k T_k, u_k R) \\
&= \min(C_k, u_k R).
\end{aligned}$$

Using (16), we get $\mathsf{Lag}(\tau_k, r + R) \geq \min(C_k, u_k R)$, which finishes the proof of the first part of the lemma.

We now move to the second part of the lemma. Fig. 4 shows the release pattern of jobs from τ_k (which covers jobs from $\mathcal{J}$). By the lemma statement, task τ_k becomes periodic with execution time C_k starting with $J_{k,d}$. Thus, exactly one job from τ_k is scheduled for execution in $\mathcal{I}$ at any time instant in $[r, r + R)$. Thus,

$$\sum_{j=d}^{\infty} \mathcal{A}(\mathcal{I}, r, r + R, J_{k,j}) = u_k R. \qquad (17)$$

By (15) and (17), we have

$$\mathsf{Lag}(\tau_k, r + R) = u_k R - W. \qquad \square$$

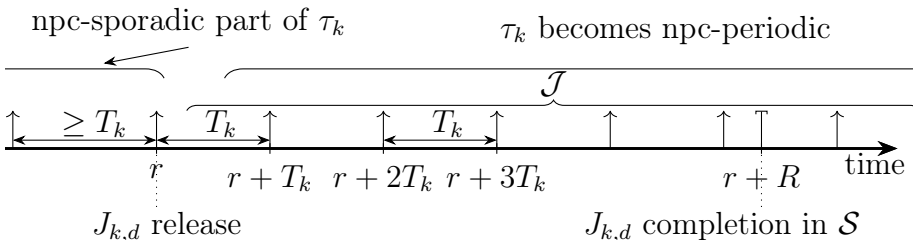


Figure 4: The release pattern of jobs of τ_k needed for the second part of Lemma 9 (eq. (14)).

4 Preemptive G-FP

In this section, we focus on establishing the SRT-optimality of the preemptive G-FP scheduler. Thus, all references to G-FP without qualification within this section should be taken to mean *preemptive* G-FP. Note that the G-FP scheduler satisfies Assumptions SH1-SH3. Thus, all lemmas from Sec. 3 hold for this scheduler.

In this section, we assume that tasks are indexed by priority, with higher-priority tasks having lower indices. For simplicity, we assume unique task priorities. Recall that we focus our attention on the job of interest $J_{k,d}$ with release time r and response time R. Note that the obtained bound (Theorem 1) does not depend on the job's number d, so the same bound applies for any job of τ_k, and therefore the bound can be used as a response-time bound of the task τ_k.

Proof setup. Our proof focuses on schedules that have certain properties (properties A1-A3), which are defied by leveraging the following definition.

Definition 11. For a task system τ, an *instantiation* ρ_τ defines an actual release and execution time for every job. A single sporadic task system has infinitely many instantiations. Note that for any instantiation ρ_τ, G-FP produces a single schedule.

Consider any instantiation ρ_τ of a task system τ. Consider a task system $\Gamma(\tau, k) = \{\tau_1, ..., \tau_k\}$ such that all tasks in $\Gamma(\tau, k)$ have the same parameters as in τ. Consider a instantiation $\rho_{\Gamma(\tau,k)}$ such that

- All jobs from tasks $\tau_1, ..., \tau_{k-1}$ in $\rho_{\Gamma(\tau,k)}$ have the same release and execution times as in ρ_τ.
- Jobs $J_{k,1}, J_{k,1}, ..., J_{k,d-1}$ in $\rho_{\Gamma(\tau,k)}$ have the same release and execution times as in ρ_τ.
- Jobs in $\mathcal{J} = \{J_{k,d}, J_{k,d+1}, ...\}$ are released periodically in $\rho_{\Gamma(\tau,k)}$, starting from time r, with period T_k, and the execution time of each of these jobs in $\rho_{\Gamma(\tau,k)}$ is C_k.

Lemma 10. *The response time of $J_{k,d}$ in a schedule produced by preemptive G-FP from $\rho_{\Gamma(\tau,k)}$ for the task system $\Gamma(\tau, k)$ is not less than the response time of $J_{k,d}$ in a schedule produced by the same scheduler from ρ_τ for the task system τ.*

Proof. Note that under preemptive G-FP, jobs from tasks with priorities lower than τ_k's priority do not affect $J_{k,d}$'s schedule. Thus, discarding all tasks with a priority lower than τ_k does not affect $J_{k,d}$. Jobs from $\mathcal{J}\backslash\{J_{k,d}\}$ have a priority lower than $J_{k,d}$ due to Assumption SH2. Thus, any change in their release pattern and execution times does not affect the schedule of $J_{k,d}$ (in any instantiation). Therefore, if $J_{k,d}$ is scheduled in the schedule produced from ρ_τ by preemptive G-FP at time instant t, then $J_{k,d}$ is also scheduled in the schedule produced from $\rho_{\Gamma(\tau,k)}$ by preemptive G-FP at time instant t (because $J_{k,d}$'s execution time in $\rho_{\Gamma(\tau,k)}$ is not less than its execution time in ρ_τ). □

Define a schedule produced from $\rho_{\Gamma(\tau,k)}$ under preemptive G-FP as the *canonical* schedule $\mathcal{S}$. Let $\mathcal{I}$ denote the ideal schedule for task system $\Gamma(\tau, k)$ and its instantiation ρ_τ. Then, $\mathcal{S}$ has the following properties.

A1: Task τ_k has the lowest priority among all tasks.

A2: Following $J_{k,d-1}$, every new job (including $J_{k,d}$) from task τ_k has execution time equal to C_k.

A3: Following $J_{k,d}$, every new job from task τ_k is released exactly T_k time units later than the previous job of τ_k (i.e., τ_k "becomes periodic" after $J_{k,d}$).

Fig. 4 shows the release pattern of jobs of τ_k. Note that none of our reasoning requires modifying the initial schedule, and we work only with schedules obtained from $\rho_{\Gamma(\tau,k)}$. Lemma 10 shows that any response-time bound derived with respect to the canonical schedule $\mathcal{S}$ for the task system $\Gamma(\tau, k)$ is valid for ρ_τ. We now can formalize our problem: *find a bound for $J_{k,d}$'s response time in $\mathcal{S}$ under A1-A3.*

We start the first part of the proof by computing $\mathsf{Lag}(\tau_k, r + R)$. Note that Property A3 ensures that task τ_k becomes npc-periodic starting with $J_{k,d}$ as in Fig. 4, while Property A2 ensures that all jobs in $\mathcal{J}$ have an execution time of C_k. Thus, by (14),

$$\mathsf{Lag}(\tau_k, r + R) = u_k R - W. \tag{18}$$

Now we move to the second part of the proof: the estimation of $\mathsf{LAG}(r + R)$.

Lemma 11. $\mathsf{LAG}(r + R) \leq (\lceil U_k \rceil - 1)C_{\max} + mC_k + (U_k - m)R - W$.

Proof. By Property A1, there are only k tasks in $\mathcal{S}$. Their total utilization is U_k, so, by Lemma 7, $\mathsf{LAG}(r) \leq (\lceil U_k \rceil - 1)C_{\max}$. Also note that if $J_{k,d}$ is not scheduled at some $t \in [r, r+R)$, then there are m other ready jobs in the system. Thus, applying Lemma 8 with $\alpha = \{\tau_1, ..., \tau_k\}$ we get $\mathsf{LAG}(r + R) \leq (\lceil U_k \rceil - 1)C_{\max} + mC_k + (U_k - m)R - W$. □

Finally, we use (18) and Lemma 11 to bound the response time of $J_{k,d}$.

Theorem 1. *For any job of the npc-sporadic task τ_k, its response time under preemptive G-FP is bounded by*

$$R \leq \frac{1}{m - U_{k-1}} \left((\lceil U_k \rceil - 1)C_{\max} + mC_k + \sum_{i=1}^{k-1} \max(0, (1 - u_i)C_i) \right),$$

where $C_{\max} = \max_{i \leq k} C_i$.

Proof. Consider the following:

$$
\begin{aligned}
& u_k R - W \\
= & \{\text{by (18)}\} \\
& \mathsf{Lag}(\tau_k, r + R) \\
= & \mathsf{LAG}(r + R) - \sum_{i=1}^{k-1} \mathsf{Lag}(\tau_i, r + R) \\
\leq & \{\text{by Lemma 4}\} \\
& \mathsf{LAG}(r + R) + \sum_{i=1}^{k-1} \max(0, (1 - u_i)C_i) \\
\leq & \{\text{by Lemma 11}\} \\
& (\lceil U_k \rceil - 1)C_{\max} + mC_k + (U_k - m)R - W + \sum_{i=1}^{k-1} \max(0, (1 - u_i)C_i). \qquad (19)
\end{aligned}
$$

Canceling W from the both sides of (19), we get

$$
R(u_k + m - U_k) \leq (\lceil U_k \rceil - 1)C_{\max} + mC_k + \sum_{i=1}^{k-1} \max(0, (1 - u_i)C_i).
$$

Rearranging the last expression and rewriting $u_k - U_k$ as $-U_{k-1}$ completes the proof. □

Corollary 2. *For the npc-sporadic task τ_k, its tardiness under preemptive G-FP is bounded by*

$$
\max \left(\frac{(\lceil U_k \rceil - 1)C_{\max} + mC_k + \sum_{i=1}^{k-1} \max(0, (1 - u_i)C_i)}{m - U_{k-1}} - T_k, 0 \right).
$$

Proof. If $J_{k,j}$ has the response time $R_{k,j}$, then its tardiness is $\max(0, R_{k,j} - T_k)$. Notice that the response-time bound from Theorem 1 does not depend on j, i.e., it applies to any job of τ_k. □

5 Asymptotic Tightness

In this section we show that bound from Theorem 1 is asymptotically tight.

Theorem 2. *For every $m \geq 2$ there exists an npc-sporadic task system such that the response time of the first job of the lowest-priority task is arbitrarily close to the bound from Theorem 1.*

Proof. For fixed T and m, consider the npc-sporadic task system consisting of m high-priority tasks with execution time mT and period $4emT$, plus one low-priority task with execution time $(1 - m/2e)T$ and period T, where $e > m$ is an arbitrary number. Let $k = m + 1$, assume all jobs from every task τ_i have the same execution time C_i, and assume that all tasks release jobs periodically starting at time 0.

The response time for $J_{m+1,1}$ (the first job of the lowest priority task) is $mT + T - mT/2e$, because for any $t \in [0, mT)$ all processors are occupied by higher-priority tasks. Subsequent jobs of τ_{m+1} have the same response time because the schedule repeats every $4emT$ time units. This is illustrated for the first two jobs per task in Fig. 5.

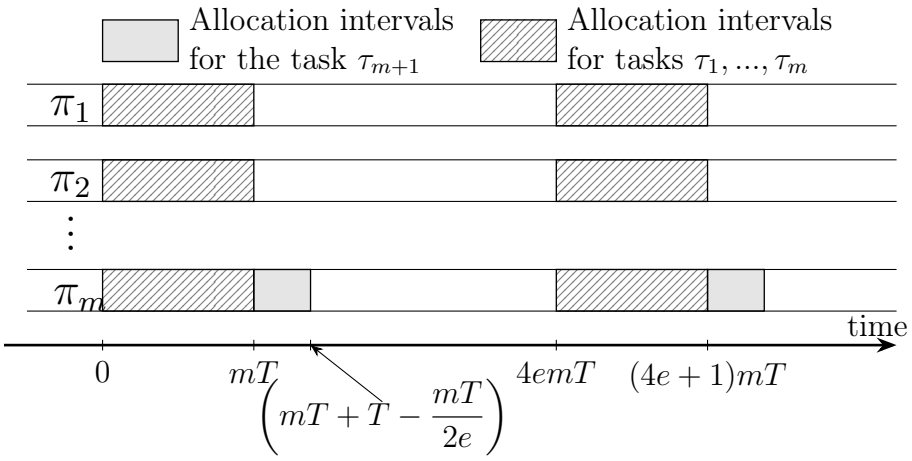


Figure 5: Schedule of the first two jobs for each task.

The overall utilization of this task set is

$$U = U_{m+1} = m \cdot \frac{1}{4e} + 1 - \frac{m}{2e} = 1 - \frac{m}{4e} < 1.$$

By construction, we have

$$\lceil U_{m+1} \rceil = 1, \tag{20}$$

$$m - U_m = m\left(1 - \frac{1}{4e}\right), \tag{21}$$

$$C_{max} = mT, \tag{22}$$

$$\sum_{i=1}^{m} \max(0, (1-u_i)C_i) = \left(1 - \frac{1}{4e}\right) m^2 T. \tag{23}$$

With all these computed values, the bound from Theorem 1 is:

$$\begin{aligned}
&\frac{1}{m - U_m}\left((\lceil U_{m+1} \rceil - 1)C_{max} + mC_{m+1} + \sum_{i=1}^{m} \max(0, (1-u_i)C_i)\right) \\
&\quad \{\text{by (20), (21), (22), and (23)}\} \\
&= \frac{1}{m\left(1 - \frac{1}{4e}\right)}\left(0 + m\left(1 - \frac{m}{2e}\right)T + \left(1 - \frac{1}{4e}\right)m^2 T\right) \\
&= \frac{T}{\left(1 - \frac{1}{4e}\right)}\left(1 - \frac{m}{2e} + m - \frac{m}{4e}\right) \\
&= \frac{4e(m+1) - 3m}{4e-1} T \\
&= \frac{(4e-1)(m+1) - 2m + 1}{4e-1} T \\
&= (m+1)T + \frac{1-2m}{4e-1} T.
\end{aligned}$$

The difference between this bound and the real response time for τ_{m+1} is

$$\left(\frac{1-2m}{4e-1}T + \frac{m}{2e}T\right) \to 0 \text{ as } e \to \infty, \text{ with fixed } m \text{ and } T.$$

Thus, the bound from Theorem 1 is asymptotically tight. □

6 Prioritizing Tasks

In the above sections, we considered npc-sporadic task sets with predefined priorities. In this section, we consider the problem of choosing task priorities for such task sets. Note that, by Theorem 1, the prioritization of tasks does not affect SRT schedulability because any task set that does not over-utilize the system is schedulable; in contrast, schedulability is the main evaluation metric in the HRT case. However, the prioritization of tasks can affect the guaranteed tardiness bounds. In this section, we propose a polynomial priority-assignment algorithm.

The existing literature pertaining to task prioritizations under preemptive G-FP scheduling considers only ordinary sporadic task sets. Audsley [2] proposed an optimal prioritization algorithm for a uniprocessor (in an HRT sense: every deadline must be met). Audsley's algorithm relies on a given schedulability test and can be adapted for the multiprocessor case. Note that Audsley's algorithm calls the schedulability test, which has non-polynomial time complexity. Moreover, the problem of obtaining an optimal priority assignment for an HRT multiprocessor system is known to be NP-hard. Davis and Burns [10] developed schedulability-test properties that ensure that Audsley's algorithm is optimal with respect to a given test (i.e., it requires an optimal schedulability test to produce an optimal prioritization; there are no known optimal polynomial tests). Given these results pertaining to HRT tasks, we expect that any optimal prioritization algorithm in the SRT casee would likely be non-polynomial. Thus, we provide a non-optimal polynomial priority-assignment algorithm.

We no longer keep the assumption made in Sec. 4 that tasks are indexed with decreasing priority. Instead, we define priorities via a function. We assume that task priorities are represented by unique integers in $[1, n]$, where 1 and n represent the highest and lowest priorities, respectively.

Definition 12. For a given task τ_i, we denote its priority as $p(i)$ and the set of tasks with higher priority as $hp(i)$. Note that $hp(i)$ contains $p(i) - 1$ tasks, so $p(i) = |hp(i)| + 1$. Function $p(\cdot)$ is called the *prioritization function*.

In addition to the previous definition, we also need to alter the definition of U_k, because it assumes tasks are indexed by priority.

Definition 13. We let U_k denote the total utilization of the k tasks with highest priorities.

$$U_k = \sum_{p(i) \leq k} u_i.$$

By construction, $U_0 = 0$.

With Defs. 12 and 13 we can rewrite the tardiness bound of Corollary 2 (assuming the upper bound to be positive) using the prioritization function $p(\cdot)$:

$$\frac{1}{m - U_{p(k)-1}} \left(\lceil U_{p(k)} \rceil - 1 \right) \left(\max_{\tau_i \in hp(k)} C_i \right) + mC_k + \sum_{\tau_i \in hp(k)} \max(0, (1 - u_i)C_i) \Bigg) - T_k. \tag{24}$$

Lemma 12. *Under preemptive G-FP scheduling, if a task τ_l swaps its priority with the task τ_h that has the next higher priority than τ_l (i.e., $p(h) = p(l) - 1$), then the tardiness of τ_l is still bounded by* (24).

Proof. Under preemptive G-FP scheduling, only higher-priority tasks may have impacts on τ_l's execution. Therefore, the scheduling of τ_l after the priority swap is identical to a special case of the scheduling of τ_l before the priority swap where every job of τ_h happens to have zero actual execution time. Since the tardiness of τ_l is bounded by (24) in any case before the priority swap, the lemma follows. □

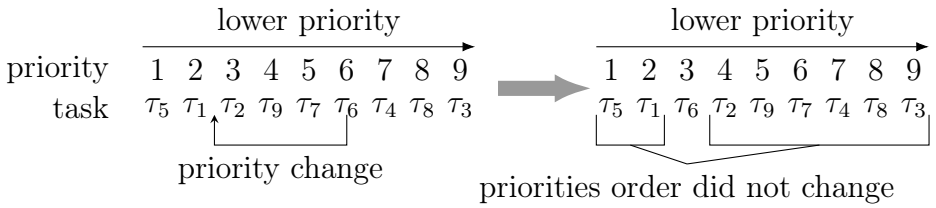


Figure 6: An example of single task priority change.

```
/* computes τ_i's tardiness assuming the lowest priority in the given τ */
Function Tardiness(m, τ_i, τ', U_τ', G_τ')
    G_{τ'\τ_i} ← G_τ' − max(0, (1 − u_i)C_i);
    U_{τ'\τ_i} ← U_τ' − u_i;
    C_max ← max_{τ_s ∈ τ'\τ_i} C_s;
    return ((⌈U_τ'⌉ − 1)C_max + mC_i + G_{τ'\τ_i}) / (m − U_{τ'\τ_i});
end
Function ComputePrioritization(τ)
    // for complexity reasons rem_tasks stored as an array sorted by WCET
    rem_tasks ← τ;
    U ← Σ_i u_i;
    G ← Σ_i max(0, (1 − u_i)C_i);
    for j = n...1 do
        τ_k ← argmin^{τ_i} Tardiness(m, τ_i, rem_tasks, U, G);
        p(k) = j;
        U ← U − u_k;
        G ← G − max(0, (1 − u_k)C_k;
        rem_tasks ← rem_tasks \ τ_k;
    end
end
```

Algorithm 1: minimization of maximal tardiness.

Lemma 13. *If the priority of a single task τ_k increases while the relative order of the priorities of all other tasks remains unchanged, then the tardiness bound for τ_k after the priority change is at most the bound for τ_k before the change.*

Proof. Fig. 6 illustrates the priority change. Increasing the priority of τ_k without changing the relative order of priorities of all other tasks can be done by repeatedly swapping the priority of τ_k and the task that has the next higher priority than τ_k. By applying Lemma 12, the lemma follows. □

Consider the well-known priority-assignment algorithm by Audsley [2]. Informally speaking, Audsley's algorithm establishes priorities from lowest to highest by assigning the lowest unassigned priority to the task with the lowest tardiness bound (which is always zero for a schedulable HRT task set). We propose using to use the same assignment scheme for the SRT case, as shown in Alg. 1. Lemma 13 gives us the intuition that non-assigned tasks' bounds do not increase after several other tasks are assigned the lowest priorities. Lemma 14 shows that Alg. 1's time complexity is relatively small.

Lemma 14. *Alg. 1 can be implemented with $O(n^2)$ time complexity.*

Proof. Firstly, we consider the function *Tardiness*. Lines 2, 3, and 5 require only constant time. Assume that tasks of τ' are stored in a sorted array by WCET. Hence, $\max_{\tau_s \in \tau' \setminus \tau_i} C_s$ is stored in the last element of τ' if it is not τ_i and in the last but one otherwise. The last statement ensures that line 4 is completed in constant time. Thus, the *Tardiness* function has $O(1)$ time complexity.

Secondly, consider the function *ComputePrioritization*. Consider a single iteration of the **for** loop. Line 12 calls the $O(1)$ function *Tardiness* $j \leq n$ times, ensuring $O(n)$ time complexity. Lines 13-15 are completed in $O(1)$ time. At line 16 Alg. 1 removes a single task from the sorted array. Because we want to keep *rem_tasks* sorted (to ensure $O(1)$ complexity for the *Tardiness* function), this takes $O(n)$ time in the worst case. Thus, a single iteration of the **for** loop requires $O(n)$ time. Because the total number of iterations is $O(n)$, the total time complexity of Alg. 1 is $O(n^2)$. □

In Sec. 10.2, we evaluate the efficacy of Alg. 1 in comparison with various prioritization heuristics.

7 Non-Preemptive G-FP

In this section, we adapt the proof strategy we described in the beginning of Sec. 3 and used to prove Theorem 1 for preemptive G-FP. This strategy has three major steps: provide a lower bound for a single task's Lag (Lemma 4), establish an upper bound on $\mathsf{LAG_k}(r+R)$ (Lemmas 17 and 18), and compute $\mathsf{Lag}(\tau_k, r+R)$ (Lemma 9). To complete the proof, we use the definition of $\mathsf{LAG_k}$ to provide a bound for the response time R of the job of interest $J_{k,d}$. Recall that $\mathsf{LAG_k}(t) = \mathsf{LAG}(\{\tau_1, ..., \tau_k\}, t) = \sum_{i=1}^{k} \mathsf{Lag}(\tau_k, t)$.

Note that the non-preemptive G-FP scheduler satisfies Assumptions SH1-SH3. Thus, all lemmas from Sec. 3 hold for this scheduler.

Consider any instantiation ρ_τ of a task system τ. Consider an instantiation ρ'_τ such that

- All jobs from tasks $\tau \setminus \{\tau_k\}$ in ρ'_τ have the same release and execution times as in ρ_τ.
- Jobs $J_{k,1}, J_{k,1}, ..., J_{k,d-1}$ in ρ'_τ have the same release and execution times as in ρ_τ.
- Jobs in $\mathcal{J} = \{J_{k,d}, J_{k,d+1}, ...\}$ are released periodically in ρ'_τ, starting from time r, with period T_k, and the execution time of each of these jobs in ρ'_τ is C_k.

Lemma 15. *The response time of $J_{k,d}$ in a schedule produced by non-preemptive G-FP from ρ'_τ is not less than the response time of $J_{k,d}$ in a schedule produced by the same scheduler from ρ_τ.*

Proof. Jobs from $\mathcal{J} \setminus \{J_{k,d}\}$ have a priority lower that $J_{k,d}$ due to Assumption SH2. Thus, any change in their release pattern and execution times does not affect the schedule of $J_{k,d}$ (in any instantiation) because once $J_{k,d}$ is scheduled, it cannot be preempted. Therefore, if $J_{k,d}$ is scheduled in the schedule produced from ρ_τ by non-preemptive G-FP at time instant t, $J_{k,d}$ is also scheduled in the schedule produced from ρ'_τ by non-preemptive G-FP at time instant t (because $J_{k,d}$'s execution time in ρ'_τ is not less than its execution time in ρ_τ). □

Define a schedule produced from ρ'_τ under non-preemptive G-FP as the *canonical* schedule $\mathcal{S}$. Define the ideal schedule $\mathcal{I}$ for the task system τ and its instantiation ρ_τ. Then, $\mathcal{S}$ has the following properties (Properties A2 and A3 are defined identically as in Sec. 4)

A2: Following $J_{k,d-1}$, every new job (including $J_{k,d}$) from task τ_k has execution time equal to C_k.

A3: Following $J_{k,d}$, every new job from task τ_k is released exactly T_k time units later than the previous job of τ_k (i.e., τ_k "becomes periodic" after $J_{k,d}$).

Fig. 4 shows the release pattern of jobs of τ_k. Note that none of our reasoning requires modifying the initial schedule, and we work only with schedules obtained from ρ'_τ. Lemma 15 shows that any response-time bound derived in the canonical schedule $\mathcal{S}$ from ρ'_τ is valid for the schedule obtained from ρ_τ. We now can formalize our problem: *find a bound for $J_{k,d}$'s response time in $\mathcal{S}$ under A2-A3.*

Note that if a job starts executing on some processor, then no higher-priority job can occupy this processor. Thus, the schedule of lower-priority jobs directly affects higher-priority ones. This implies that we are not able to discard the tasks with priority lower than the task of interest τ_k in the general case (and obtain Property A1 for $\mathcal{S}$ in addition to A2 and A3).

To simplify the following reasoning, we denote $HP = \{\tau_1, ..., \tau_k\}$ and $LP = \{\tau_{k+1}, \tau_{k+2}, ..., \tau_n\}$. We say that a job is LP (resp., HP) if it was generated by a task in LP (resp., HP).

Definition 14. Let us denote $C_{\max} = \max\limits_{i \le k} C_i$ and $B = \max\limits_{i > k} C_i$. $C_{\max}$ provides an execution time bound for any job of a task in HP, while B is a bound on the maximal blocking time for such a job (an execution time bound for any job of a task in LP).

Note that the first step of the proof overview given in the beginning of this section is already completed with Lemma 4. Thus, we move to the second step, bounding $\mathsf{LAG_k}(r+R)$. We begin by establishing two upper bounds in $\mathsf{LAG_k}(t)$ in Lemmas 16 and 17.

Lemma 16. $\mathsf{LAG_k}(t) \le qC_{\max}$, *where q is the number of non-completed HP jobs in $\mathcal{S}$ at time t.*

Proof.

$$
\begin{aligned}
\mathsf{LAG_k}(t) &= \sum_{\tau_i \in HP} \mathsf{Lag}(J_{i,j}, t) \\
&\le \{\text{jobs completed in } \mathcal{S} \text{ have non-positive } \mathsf{Lag}\} \\
&\quad \sum_{J_{i,j} \text{ is a ready } HP \text{ job in } \mathcal{S}} \mathsf{Lag}(J_{i,j}, t) \\
&\le \{\text{by Lemma 3}\} \\
&\quad \sum_{J_{i,j} \text{ is a ready } HP \text{ job in } \mathcal{S}} C_i \\
&\le \{\text{by the defintion of } C_{\max} \text{ and } q\} \\
&\quad qC_{\max}
\end{aligned}
$$

□

Lemma 17. [modified Lemma 7] $\mathsf{LAG_k}(t) \le (U_k + 1)\max(B, C_{\max})$.

Proof. Let $F = (U_k + 1)\max(B, C_{\max})$. For any time instant t_1, define t_1^- (resp., t_1^+) to be $t_1 - \varepsilon$ (resp., $t_1 + \varepsilon$) for some arbitrarily small $\varepsilon > 0$ such that the set of scheduled jobs does not change within $[t_1^-, t_1)$ (resp., $[t_1, t_1^+)$). t_1^- and t_1^+ exist for any time instant for any scheduler satisfying Assumption H3. Fig. 7 illustrates these and other important time instants referenced in this proof.

We prove the lemma by contradiction by assuming that $\mathsf{LAG_k}(t) > F$ holds for some t. Let t_1 be the first time instant such that $\mathsf{LAG_k}(t_1) = F$ and $\mathsf{LAG_k}(t_1^+) > F$. Such an instant t_1 exists because $\mathsf{LAG_k}(0) = 0$, $F > 0$, and $\mathsf{LAG_k}(t)$ is a continuous function by Lemma 5.

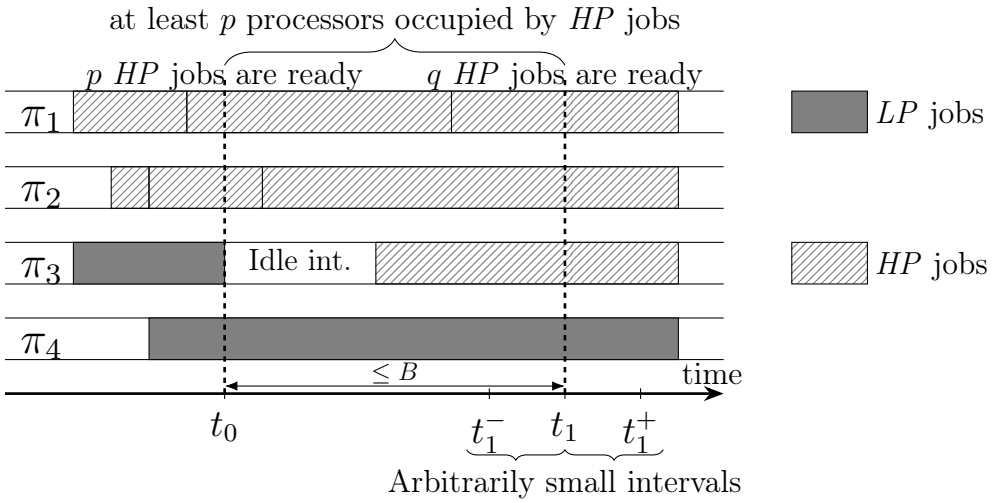


Figure 7: Lemma 17 proof reference.

Claim 1: there is at least one LP job scheduled at t_1^-.

Proof. Denote the number of scheduled HP jobs at t_1 as q (by the definition of t_1^+ these jobs are scheduled within $[t_1, t_1^+)$). If there are no scheduled LP jobs at t_1, then either all ready HP jobs are scheduled at t_1 or $q = m$. In the first case, $F = \mathsf{LAG_k}(t_1) \leq qC_{\max}$ by Lemma 16. Then, $F = (U_k + 1)\max(B, C_{\max}) \leq qC_{\max}$, so $U_k + 1 \leq q$. In the second case, $q = m$, so $U_k \leq q$ because $U_k \leq U \leq m$.

Thus, $\mathsf{LAG_k}(t_1^+) = \mathsf{LAG_k}(t_1) + (U_k - q)(t_1^+ - t_1)$, and, because $U_k \leq q$, $\mathsf{LAG_k}(t_1^+) \leq \mathsf{LAG_k}(t_1) = F$ (which contradicts the definition of t_1).

So at least one LP job $J_{i,j}$ is scheduled at t_1. If the first time instant $J_{i,j}$ is scheduled is t_1, then all ready HP jobs are scheduled at t_1, and, by the same reasoning as above, $U_k \leq q$ and $\mathsf{LAG_k}(t_1^+) \leq F$ (which contradicts the definition of t_1). Thus, $J_{i,j}$ is scheduled within $[t_1^-, t_1]$. □

We say that a job *becomes* scheduled at t if t is the first time instant it is executed. Consider all time instants within $[0, t_1)$ such that either an LP job becomes scheduled or a processor becomes idle. Denote the last such time instant as t_0. If no such t_0 exists then only HP jobs are scheduled within $[0, t_1)$. In this case, by Lemma 7 with task set $\alpha = HP = \{\tau_1, ..., \tau_k\}$, $\mathsf{LAG_k}(t_1) \leq (\lceil U_k \rceil - 1)C_{\max}$. Because $(\lceil U_k \rceil - 1)C_{\max} < (U_k + 1)\max(B, C_{\max})$, this contradicts the definition of t_1. We therefore conclude that t_0 does exist.

Note that all ready HP jobs are scheduled at t_0.

Claim 2: $t_1 - t_0 \leq B$.

Proof. By Claim 1, there is at least one LP job scheduled at t_1^-. It becomes scheduled at or before t_0 by the definition of t_0. Its execution time is bounded by B. □

Let p denote the number of incomplete HP jobs at t_0 (all of them are scheduled at t_0 by its definition). Due to the definition of t_0, we can determine the number of processors that are used by HP jobs within $[t_0, t_1)$.

Claim 3: there are at least p processors that are busy executing HP jobs for any $t \in [t_0, t_1)$.

Proof. At t_0 there are p such processors by the definition of p. If there is a $t' > t_0$ that contradicts the claim, then either an *LP* job is scheduled at t', or some processor becomes idle. This contradicts the definition of t_0. □

By Claim 3, $\mathcal{A}(\mathcal{S}, t_0, t_1, HP) \geq p(t_1 - t_0)$. By Lemma 6, $\mathcal{A}(\mathcal{I}, t_0, t_1, HP) \leq U_k(t_1 - t_0)$. Thus,

$$\mathsf{LAG_k}(t_1) - \mathsf{LAG_k}(t_0) = \mathcal{A}(\mathcal{I}, t_0, t_1, HP) - \mathcal{A}(\mathcal{S}, t_0, t_1, HP) \leq (U_k - p)(t_1 - t_0).$$

Rearranging the last expression we get

$$\mathsf{LAG_k}(t_1) \leq \mathsf{LAG_k}(t_0) + (U_k - p)(t_1 - t_0). \tag{25}$$

Claim 4: $p \leq U_k$.

Proof. By the definition of t_1, $\mathsf{LAG_k}(t_0) \leq F = \mathsf{LAG_k}(t_1)$. Then, by (25), $(U_k - p)(t_1 - t_0) \geq 0$. Because $t_1 - t_0 > 0$, $p \leq U_k$. □

By the definition of t_0, all ready *HP* jobs are scheduled at t_0. Thus, by Lemma 16, $\mathsf{LAG_k}(t_0) \leq pC_{\max}$. By (25), this implies

$$F = \mathsf{LAG_k}(t_1) \leq \mathsf{LAG_k}(t_0) + (U_k - p)(t_1 - t_0) \leq pC_{\max} + (U_k - p)(t_1 - t_0),$$

which by the definition of F implies

$$(U_k + 1)\max(B, C_{\max}) \leq pC_{\max} + (U_k - p)(t_1 - t_0). \tag{26}$$

Using Claim 2 from (26) we get

$$\begin{aligned}
(U_k + 1)\max(B, C_{\max}) &\leq pC_{\max} + (U_k - p)B \\
&\leq \{0 \leq p, C_{\max} \leq \max(B, C_{\max})\} \\
&\quad p\max(B, C_{\max}) + (U_k - p)B \\
&\leq \{p \leq U_k \text{ by Claim 4, } B \leq \max(B, C_{\max}))\} \\
&\quad p\max(B, C_{\max}) + (U_k - p)\max(B, C_{\max}) \\
&= U_k \max(B, C_{\max}),
\end{aligned}$$

which leads to contradiction. □

Now we are able to adapt Lemma 8 for non-preemptive G-FP. Recall from Def. 10 that W is the overall processor allocation to jobs from $\mathcal{J} = \{J_{k,d}, J_{k,d+1}, J_{k,d+2}, ...\}$ in $\mathcal{S}$ in the interval $[r, r + R)$.

Lemma 18. [modified Lemma 8] *If* $R \geq C_k + B$ *then*

$$\mathsf{LAG_k}(r + R) \leq \mathsf{LAG_k}(r) + m(C_k + B) + (U_k - m)R - W.$$

Proof. Let t_0 be the last completion time of an *LP* job scheduled at r ($t_0 = r$ if no *LP* jobs are scheduled at r). Then $t_0 \leq r + B$ by the definition of B. Let $t_1 = r + R - C_k$. Then $t_0 \leq r + B \leq t_1$ because $R \geq C_k + B$. Recall that $HP = \{\tau_1, ..., \tau_k\}$, and, by Def. 7, $\mathsf{LAG_k}(t) = \mathcal{A}(\mathcal{I}, 0, t, HP) - \mathcal{A}(\mathcal{S}, 0, t, HP)$. An example schedule over time interval $[r, r + R)$ under non-preemptive G-FP can be found in Fig. 8.

Note that if $J_{k,d}$ is scheduled, then it cannot be preempted. Thus, $J_{k,d}$ is scheduled within $[t_1, r + R)$ because the execution time of $J_{k,d}$ is C_k by Property A2. Also note that no jobs from $\mathcal{J}$ can be scheduled before t_1 by Assumption SH2.

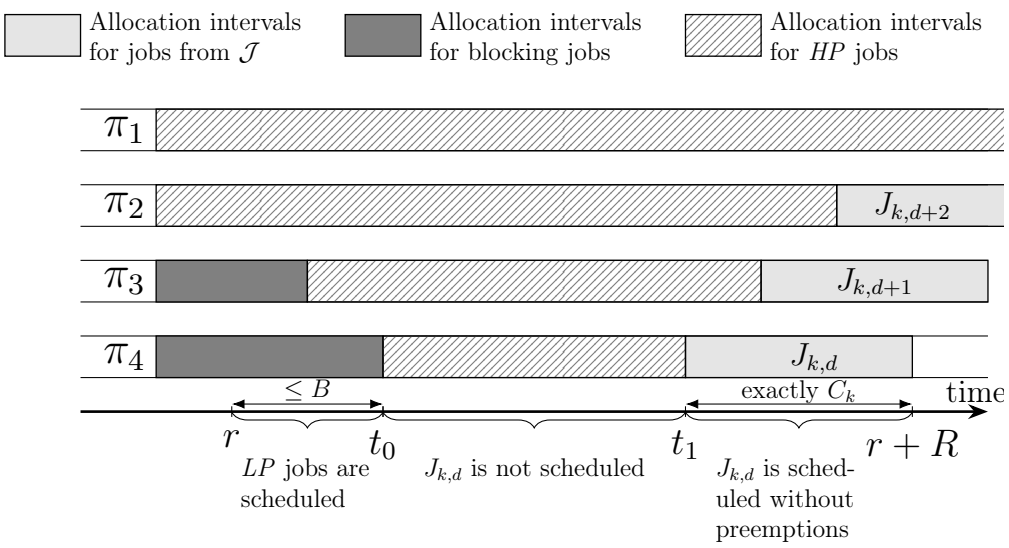


Figure 8: Example partitioning of $[r, r + R)$ into intervals (Lemma 18 reference).

In fact, the proof we provide here is a simplified version of the proof of Lemma 8 that utilizes alternate definitions of $\mathcal{Z}$ and $\mathcal{M}$. Due to the non-preemptivity of the scheduler, some LP jobs are scheduled within $[r, t_0)$. Thus, we define $\mathcal{Z}$ to be the set of all intervals for which only HP jobs are scheduled in $\mathcal{S}$, and $\mathcal{M}$ to be the set of all time intervals for which at least one non-HP job is scheduled in $\mathcal{S}$. Note that these definitions imply that $\mathcal{M} = \{[r, t_0), [t_1, r + R)\}$ and $\mathcal{Z} = \{[t_0, t_1)\}$. Fortunately, since $\mathcal{Z}$ and $\mathcal{M}$ together contain only three time intervals, we can simplify our reasoning compared to the proof of Lemma 8.

$$
\begin{aligned}
&\mathsf{LAG_k}(r + R) - \mathsf{LAG_k}(r) \\
&= \mathcal{A}(\mathcal{I}, r, r + R, HP) - \mathcal{A}(\mathcal{S}, r, r + R, HP) \\
&\leq \{\text{by Lemma 6 with } \alpha = HP = \{\tau_1, ..., \tau_k\}\} \\
&\quad U_k R - \mathcal{A}(\mathcal{S}, r, r + R, HP) \\
&= U_k R - \mathcal{A}(\mathcal{S}, r, t_0, HP) - \mathcal{A}(\mathcal{S}, t_0, t_1, HP) - \mathcal{A}(\mathcal{S}, t_1, r + R, HP) \\
&\leq \{\mathcal{A}(\mathcal{S}, r, t_0, HP) \geq 0\} \\
&\quad U_k R - \mathcal{A}(\mathcal{S}, t_0, t_1, HP) - \mathcal{A}(\mathcal{S}, t_1, r + R, HP) \\
&= \{\text{exactly } m \text{ processors are busy with tasks from } HP \text{ within } [t_0, t_1)\} \\
&\quad U_k R - m(t_1 - t_0) - \mathcal{A}(\mathcal{S}, t_1, r + R, HP)
\end{aligned}
$$

$$
\begin{aligned}
&\leq \{\text{all allocation to tasks in } \mathcal{J} \text{ happens within } [t_1, r + R)\} \\
&\quad U_k R - m(t_1 - t_0) - W \\
&= U_k R - m(R - B - C_k) - W \\
&= m(C_k + B) + (U_k - m)R - W. \qquad (27)
\end{aligned}
$$

By rearranging (27), we obtain a bound for $\mathsf{LAG_k}(r + R)$. □

Thus, if $R \geq C_k + B$, using Lemma 17 and Lemma 18, we can write

$$\mathsf{LAG_k}(r + R) \leq (U_k + 1) \max(B, C_{\max}) + m(C_k + B) + (U_k - m)R - W. \qquad (28)$$

Theorem 3. *Non-preemptive G-FP ensures the following response-time bound for any npc-sporadic task τ_k.*

$$\max\left(C_k + B, \frac{1}{m - U_{k-1}}\left(mB + (U_k + 1)\max(B, C_{\max}) \right.\right.$$
$$\left.\left. +(m-1)C_k + \sum_{i<k} \max(0, (1-u_i)C_i)\right)\right),$$

where $C_{\max} = \max_{i \le k} C_i$ *and* $B = \max_{i>k} C_i$.

Proof. Consider a job $J_{k,d}$ of task τ_k. Assume that $R \ge B + C_k$. Then (28) holds. We apply the same reasoning as in Theorem 1 where Lemma 11 is replaced with Lemma 18.

$$\begin{aligned}
& u_k R - W \\
= \; & \{\text{by A2, A3, Lemma 9 and (14)}\} \\
& \mathsf{Lag}(\tau_k, r + R) \\
= \; & \mathsf{LAG}(r + R) - \sum_{i \ne k} \mathsf{Lag}(\tau_i, r + R) \\
\le \; & \{\text{by Lemma 4}\} \\
& \mathsf{LAG}(r + R) + \sum_{i \ne k} \max(0, (1 - u_i)C_i) \\
\le \; & \{\text{by (28)}\} \\
& (U_k + 1)\max(B, C_{\max}) + m(C_k + B) + (U_k - m)R - W \\
& \quad + \sum_{i \ne k} \max(0, (1 - u_i)C_i). \qquad (29)
\end{aligned}$$

Canceling W from the both sides of (29), we get

$$R(u_k + m - U_k) \le mB + (U_k + 1)\max(B, C_{\max}) + (m-1)C_k + \sum_{i \ne k} \max(0, (1-u_i)C_i).$$

Rearranging the last expression (and using $U_k - u_k = U_{k-1}$) completes the proof. □

8 G-FP with Preemption Thresholds

Fixed-priority scheduling is widely used in real-time operating systems (RTOSs). However, G-FP may induce a high number of preemptions (the preemptive variant) or large blocking times (the non-preemptive variant). One way to mitigate these issues is to use two priorities per task instead of one. The first priority is applied at task release; once a task is selected for execution, the second priority is applied. This approach is known as G-FP with preemption thresholds (G-FP-PT), and was introduced in the commercial RTOS ThreadX [15] and academically studied in [28] (corrected in [24]). Other studies include [8, 16–18, 25].

To specify the scheduler, we define a preemption priority $\mathcal{P}_i$ for every task τ_i. Recall that our tasks are sorted by priority, so τ_i has regular priority i (lower values represent higher priority). We assume that $\mathcal{P}_i \le i$, so the preemption priority of task is not lower than its regular priority. These priorities are assigned offline. Thus, a scheduled task τ_i can be preempted by τ_j only if τ_j's regular priority is higher than $\mathcal{P}_i$.

Note that preemptive G-FP is a special case of G-FP-PT ($\forall i : \mathcal{P}_i = i$), and non-preemptive G-FP is also a special case of G-FP-PT ($\forall i : \mathcal{P}_i = 0$). For both of these schedulers, in Secs. 4 and 7, we constructed canonical schedules that have properties A2 and A3. Unfortunately, these properties are relatively strong, so we may not be able to construct a canonical schedule $\mathcal{S}$ with a non-decreased response time for $J_{k,d}$. Thus, we introduce a less strict Property A4.

A4: $J_{k,d}$ has execution time equal to C_k.

This property ensures that all jobs of τ_k preceding $J_{k,d}$ are completed at or before $J_{k,d}$'s completion by Assumption SH2.

Consider any instantiation ρ_τ of a task system τ. Consider an instantiation ρ'_τ of the same task system such that

- All jobs from tasks in $\tau \backslash \{\tau_k\}$ in ρ'_τ have the same release and execution times as in ρ_τ.
- Jobs $J_{k,1}, J_{k,1}, ..., J_{k,d-1}, J_{k,d+1}, J_{k,d+2}, ...$ in ρ'_τ have the same release and execution times as in ρ_τ.
- The execution time of $J_{k,d}$ is C_k in ρ'_τ, and its release time is r (identical to the one in ρ_τ).

Lemma 19. *The response time of $J_{k,d}$ in a schedule produced by G-FP-PT from ρ'_τ is not less than the response time of $J_{k,d}$ in a schedule produced by the same scheduler from ρ_τ.*

Proof. Note that ρ_τ and ρ'_τ differ only in the execution time of $J_{k,d}$. Denote the completion of $J_{k,d}$ in a schedule $\mathcal{S}_{\text{init}}$ obtained from ρ_τ under G-FP-PT as t_0. Then, the schedule obtained from ρ'_τ under G-FP-PT ($\mathcal{S}_{\text{mod}}$) is identical to $\mathcal{S}_{\text{init}}$ within $[0, t_0)$ because all releases of all jobs are identical. Then, $J_{k,d}$ is completed in $\mathcal{S}_{\text{init}}$, but has $C_k - C_{k,d}$ time units of uncompleted execution in $\mathcal{S}_{\text{mod}}$. □

We call a schedule $\mathcal{S}_{\text{mod}}$ produced from ρ'_τ under G-FP-PT a *canonical* schedule, which we henceforth denote more simply as $\mathcal{S}$. Define the ideal schedule $\mathcal{I}$ for the task system τ and its instantiation ρ'_τ.

In this section, we prove two different response-time bounds. Both of them require specially defined task sets in order to use Lemma 8.

Definition 15. Let β be the set of tasks in τ such that $\mathcal{P}_i \leq k$ (i.e., have preemption priority not lower than the regular priority of τ_k). Note that, by the definition of $\mathcal{P}_i$, $\mathcal{P}_i \leq i$, so all tasks that have higher regular priority than τ_k are in β. Thus, if $J_{k,d}$ is not scheduled then all m processors are busy with tasks in β.

Lemma 20. [relaxed Lemma 7]

$$\mathsf{LAG}(\beta, t) \leq (\lceil U \rceil - 1)C_{\max} + \sum_{\tau_i \in \tau \backslash \beta} \max(0, (1 - u_i)C_i),$$

where $C_{\max} = \max_i C_i$.

Proof.

$$\begin{aligned}
\mathsf{LAG}(\beta, t) &= \mathsf{LAG}(\tau, t) - \mathsf{LAG}(\tau \backslash \beta, t) \\
&\leq \{\text{by Lemma 7 with } \alpha = \tau\} \\
&\quad (\lceil U \rceil - 1) C_{\max} - \mathsf{LAG}(\tau \backslash \beta, t) \\
&= \{\text{by Def. 7}\} \\
&\quad (\lceil U \rceil - 1) C_{\max} - \sum_{\tau_i \in \tau \backslash \beta} \mathsf{Lag}(\tau_i, t) \\
&\leq \{\text{by Lemma 4}\} \\
&\quad (\lceil U \rceil - 1) C_{\max} - \sum_{\tau_i \in \tau \backslash \beta} \min(0, (u_i - 1) C_i) \\
&= (\lceil U \rceil - 1) C_{\max} + \sum_{\tau_i \in \tau \backslash \beta} \max(0, (1 - u_i) C_i)
\end{aligned}$$

□

Using Lemmas 4, 18, and 20 we provide a response-time bound.

Theorem 4. *G-FP-PT ensures the following response-time bound for any npc-sporadic task τ_k if $U_\beta < m$:*

$$\min \left(T_k, \frac{1}{m - U_\beta} \left((\lceil U \rceil - 1) C_{\max} + (m-1) C_k + \sum_{i \neq k} \max(0, (1 - u_i) C_i) \right) \right),$$

where $C_{\max} = \max_i C_i$ and β is as defined in Def. 15.

Proof. We consider the job of interest $J_{k,d}$. Assume that $R \geq T_k$, so $C_k = u_k T_k \leq u_k R$.

$$\begin{aligned}
&C_k - W \\
&= \{\text{by Lemma 9, (13), and } C_k \leq u_k R\} \\
&\quad \mathsf{Lag}(\tau_k, r + R) \\
&= \mathsf{LAG}(\beta, r + R) - \sum_{\tau_i \in \beta \backslash \tau_k} \mathsf{Lag}(\tau_i, r + R) \\
&\leq \{\text{by Lemma 4}\} \\
&\quad \mathsf{LAG}(\beta, r + R) + \sum_{\tau_i \in \beta \backslash \{\tau_k\}} \max(0, (1 - u_i) C_i) \\
&\leq \{\text{by Lemma 8 with } \alpha = \beta\} \\
&\quad \mathsf{LAG}(\beta, r) + m C_k + (U_\beta - m) R - W + \sum_{\tau_i \in \beta \backslash \{\tau_k\}} \max(0, (1 - u_i) C_i) \\
&\leq \{\text{by Lemma 20, and } C_{\max} = \max_i C_i\} \\
&\quad (\lceil U \rceil - 1) C_{\max} + \sum_{\tau_i \in \tau \backslash \beta} \max(0, (1 - u_i) C_i) + m C_k + (U_\beta - m) R - W \\
&\quad + \sum_{\tau_i \in \beta \backslash \{\tau_k\}} \max(0, (1 - u_i) C_i)
\end{aligned}$$

$$= (\lceil U \rceil - 1)C_{\max} + mC_k + (U_\beta - m)R - W + \sum_{\tau_i \in \tau \setminus \{\tau_k\}} \max(0, (1 - u_i)C_i)$$
$$= (\lceil U \rceil - 1)C_{\max} + mC_k + (U_\beta - m)R - W + \sum_{i \neq k} \max(0, (1 - u_i)C_i) \tag{30}$$

Canceling W from the both sides of (30), we get

$$R(m - U_\beta) \leq (\lceil U \rceil - 1)C_{\max} + (m-1)C_k + \sum_{i \neq k} \max(0, (1 - u_i)C_i.$$

Rearranging the last expression completes the proof. □

Now we repeat the same strategy with a slightly different task set γ to provide a response-time bound that can be lower than Theorem 4 in some cases.

Definition 16. We call a set of tasks λ *closed* if and only if $\forall \tau_i \in \lambda, \tau_j \in \tau \setminus \lambda : i < \mathcal{P}_j$ (tasks from the closed set preempt any jobs of tasks not in the set). Let γ denote the smallest closed set that contains τ_k. Note that γ is well-defined because τ is a closed set.

Now we can prove our second response-time bound for τ_k using γ. Notice the key difference with Theorem 4: by the definition of a closed set, we can use Lemma 7 with $\alpha = \gamma$, while for β we have to use its relaxed version, Lemma 20.

Theorem 5. *G-FP-PT ensures the following response-time bound for any npc-sporadic task τ_k if $U_\gamma < m$:*

$$\min \left(T_k, \frac{1}{m - U_\gamma} \left((\lceil U_\gamma \rceil - 1)C_{\max} + (m-1)C_k + \sum_{\tau_i \in \gamma \setminus \{\tau_k\}} \max(0, (1 - u_i)C_i) \right) \right),$$

where $C_{\max} = \max_{\tau_i \in \gamma} C_i$ and γ as is defined in Def. 16.

Proof. We consider the job of interest $J_{k,d}$. Assume that $R \geq T_k$, so $C_k = u_k T_k \leq u_k R$.

$$\begin{aligned}
& C_k - W \\
&= \{\text{by Lemma 9, (13), and } C_k \leq u_k R\} \\
& \mathsf{Lag}(\tau_k, r + R) \\
&= \mathsf{LAG}(\gamma, r + R) - \sum_{\tau_i \in \gamma \setminus \{\tau_k\}} \mathsf{Lag}(\tau_i, r + R) \\
&\leq \{\text{by Lemma 4}\} \\
& \mathsf{LAG}(\gamma, r + R) + \sum_{\tau_i \in \gamma \setminus \{\tau_k\}} \max(0, (1 - u_i)C_i) \\
&\leq \{\text{by Lemma 8 with } \alpha = \gamma\} \\
& \mathsf{LAG}(\gamma, r) + mC_k + (U_\gamma - m)R - W + \sum_{\tau_i \in \gamma \setminus \{\tau_k\}} \max(0, (1 - u_i)C_i) \\
&\leq \{\text{by Lemma 7 with } \alpha = \gamma, \text{ and } C_{\max} = \max_{\tau_i \in \gamma} C_i\} \\
& (\lceil U_\gamma \rceil - 1)C_{\max} + mC_k + (U_\gamma - m)R - W + \sum_{\tau_i \in \gamma \setminus \{\tau_k\}} \max(0, (1 - u_i)C_i)
\end{aligned}$$

Rearranging the last expression completes the proof. □

9 Any Work-Conserving Scheduler

In this section, we extend our attention from the G-FP scheduler and its variants to the broader range of any global work-conserving schedulers that satisfy Assumptions SH1-SH3. A scheduler is global work-conserving (Assumption SH1) if it always generates a schedule such that

- At any time instant where at most m jobs are ready, all ready jobs are scheduled.
- At any time instant where more than m jobs are ready, all processors are busy.

The following theorem shows that, for a feasible npc-sporadic task system, any work-conserving scheduler that prioritizes jobs of the same task in FIFO order will guarantee bounded response times for all tasks.

Theorem 6. *Under any work-conserving scheduler that prioritizes jobs of the same task in FIFO order, the response time of a task τ_k is at most*

$$L_k = \frac{(\lceil U \rceil - 1)C_{\max} + 2C_{sum} + (m-2)C_k}{m - U + u_k}, \tag{31}$$

where $C_{\max} = \max_i C_i$ and $C_{sum} = \sum_i C_i$.

Proof. We prove this theorem by contradiction. Suppose the theorem does not hold and let $J_{k,d}$ denote the first job of task τ_k that has a response time greater than L_k. Recall that the release time of $J_{k,d}$ is r, i.e., $J_{k,d}$ has not completed its execution by time $r + L_k$.

Because jobs of the same task are prioritized in FIFO order, jobs of τ_k released after r cannot prevent $J_{k,d}$ from being executed. We divide all other jobs into the following three disjoint sets.

Ψ_1: the set of jobs that are released *before* r and have a deadline *at or before* r;

Ψ_2: the set of jobs that are released *before* r and have a deadline *after* r;

Ψ_3: the set of jobs of any task *other than* τ_k that are released *at or after* r.

Thus, at any time instant at or after time r, either $J_{k,d}$ is being executed or all m processors are busy executing jobs from $\Psi_1 \cup \Psi_2 \cup \Psi_3$, because the scheduler is work-conserving. As a result, letting ℓ denote the accumulated length of time where all m processors are busy executing jobs from $\Psi_1 \cup \Psi_2 \cup \Psi_3$ at or after time r, the following must hold:

$$L_k < \ell + C_k; \tag{32}$$

otherwise, $J_{k,d}$ must have been executed for at least C_k time units during the time interval $[r, r + L_k)$ but, by the definition of $J_{k,d}$, it has not completed yet. This does not comply with C_k being defined as the worst-case execution time of τ_k.

Additionally, letting W_{tot} denote the total work completed during the time interval $[r, r+L_k)$ for jobs in $\Psi_1 \cup \Psi_2 \cup \Psi_3$ ("*tot*" stands for total), we have

$$W_{tot} \geq m \cdot \ell. \tag{33}$$

We further consider jobs contributing to W_{tot} separately by whether they are (a) released before time r, or (b) released at or after time r. Letting W_{co} denote the unfinished work of jobs in $\Psi_1 \cup \Psi_2$ at time r ("*co*" stands for carry over) and letting W_{nr} denote the work of jobs in Ψ_3 released at or before time $r + L_k$ ("*nr*" stands for new releases), it is clear that

$$W_{tot} = W_{co} + W_{nr}. \tag{34}$$

Recall that $\mathcal{I}$ is the ideal schedule we defined in Sec. 2 where all jobs are completed at or before their deadlines. Therefore, the total work of jobs in Ψ_1 is at most $\mathcal{A}(\mathcal{I}, 0, r, \tau)$. Also, each task can have at most one job in Ψ_2 because its period equals its relative deadline, and task τ_k does not have a job in Ψ_2 because its job $J_{k,d}$ is released precisely at time r. Therefore, the total work of jobs in Ψ_2 is at most $\sum_{i \neq k} C_i$.

Now, let $\mathcal{S}$ denote the schedule generated by the scheduler that, by Assumption SH2, prioritizes jobs of the same task in FIFO order. Because $\mathcal{S}$ cannot schedule any job released at or after r during the time interval $[0, r)$, the total amount of work completed in $\mathcal{S}$ during time interval $[0, r)$, i.e., $\mathcal{A}(\mathcal{S}, 0, r, \tau)$, contributes to jobs in $\Psi_1 \cup \Psi_2$.

Thus, we have

$$\begin{aligned} W_{co} &\leq \mathcal{A}(\mathcal{I}, 0, r, \tau) + \sum_{i \neq k} C_i - \mathcal{A}(\mathcal{S}, 0, r, \tau) \\ &= \{\text{by Def. 7}\} \\ &\quad \mathsf{LAG}(r) + \sum_{i \neq k} C_i \\ &= \{\text{by the definition of } C_{\text{sum}}\} \\ &\quad \mathsf{LAG}(r) + (C_{\text{sum}} - C_k) \\ &\leq \{\text{by Lemma 7}\} \\ &\quad (\lceil U \rceil - 1) C_{\max} + C_{\text{sum}} - C_k. \end{aligned} \tag{35}$$

In terms of W_{nr}, because any task τ_i releases jobs with a minimum separation T_i, it is clear that

$$\begin{aligned} W_{nr} &\leq \sum_{i \neq k} \left(\left\lceil \frac{L_k}{T_i} \right\rceil C_i \right) \\ &< \{\text{because } \lceil x \rceil < (x+1)\} \\ &\quad \sum_{i \neq k} \left(\left(\frac{L_k}{T_i} + 1 \right) C_i \right) \\ &= \{\text{because } u_i = C_i / T_i\} \\ &\quad \left(\sum_{i \neq k} u_i \right) L_k + \sum_{i \neq k} C_i \\ &= \{\text{by the definition of } U \text{ and } C_{\text{sum}}\} \\ &\quad (U - u_k) L_k + C_{\text{sum}} - C_k. \end{aligned} \tag{36}$$

By (32), (33), (34), (35), and (36) and noting that $m > 0$ holds, we have

$$L_k < \frac{(\lceil U \rceil - 1) C_{\max} + C_{\text{sum}} - C_k + (U - u_k) L_k + C_{\text{sum}} - C_k}{m} + C_k.$$

That is,

$$(m - U + u_k) L_k < (\lceil U \rceil - 1) C_{\max} + 2 C_{\text{sum}} + (m - 2) C_k.$$

Because $m \geq U$ must hold for any feasible system and $u_k > 0$ holds (as $C_k > 0$ holds by definition and $u_k = C_k / T_k$), we have $m - U + u_k > 0$. Thus,

$$L_k < \frac{(\lceil U \rceil - 1) C_{\max} + 2 C_{\text{sum}} + (m - 2) C_k}{m - U + u_k},$$

which contradicts the definition of L_k in (31). Thus, the supposition at the beginning of this proof cannot be true, and the theorem follows. □

Corollary 3. *Any work-conserving scheduler is soft real-time optimal under the npc-sporadic task model.*

10 Experiments

In this section, we evaluate the obtained analytical tardiness bound for the preemptive G-FP scheduler, which is probably the most practically relevant scheduler among those considered in the paper. Two evaluation metrics naturally arise with respect to tardiness analysis. The first one is *average tardiness* over all tasks in the system, and the second one is *maximum tardiness* over all tasks. In order to evaluate our analytical results, we use both metrics in several experiments.

Firstly, we consider the difference between analytical and observed tardiness. Because the analytical bound can be significantly higher than the observed one (up to an order of magnitude), we propose a bound reduction method based on clustering. Unfortunately, clustering is an NP-hard problem (it can be reduced to the bin-packing problem). However, we conducted experiments showing that for tasks systems in which no task has high utilization, clustering can effectively reduce the analytical bound for almost all task sets.

Secondly, we consider the task prioritization problem. Our experiments pertaining to this problem focus on heuristics, because, unfortunately, the problem of finding the optimal (with respect to tardiness minimization) prioritization seems to have no polynomial solution unless P=NP. We do not have a proof for this statement but provide intuition that it is so. This intuition comes from examining prior work on two well-studied real-time scheduling models that are related to the npc-sporadic model: ordinary sporadic tasks and a collection of independent jobs.

From the point of view of ordinary tasks, Leung and Merril [21, Theorem 2, for a single processor] and Leung [20, Theorem 4, for multiprocessor] showed that the problem of deciding if a periodic task system is schedulable under any fixed-priority algorithm is co-NP-hard in the strong sense. Note that a periodic task system is a special case of a sporadic tasks system.

From the point of view of independent jobs, the problem of minimizing either of our two evaluation metrics has a similar existing problem known to be NP-hard. Minimizing the average response time of an npc-sporadic task system has the same nature as the problem of minimizing the makespan of a collection of jobs, which was studied by Uzsoy [26, Corollary 1]. The concept of minimizing the maximum response time of an npc-sporadic task system has the same nature as minimizing the number of tardy jobs; this problem was studied was studied by Li and Lee [23, Theorem 1].

Thirdly, we consider an exponential algorithm for the optimal prioritization and use it to evaluate the quality of the considered heuristics for small task sets.

To generate the task systems used in this section, we selected task periods from the range $[10ms, 100ms]$ (uniformly distributed). We also considered various task utilization ranges: *light tasks* with $u_i \in (0.0, 0.3)$, *medium tasks* with $u_i \in [0.3, 0.7)$, and *heavy tasks* with $u_i \in [0.7, 1.0)$.

The source code we developed for this experimental evaluation can be found online [1].

10.1 Analytical Tardiness Bound vs Observed Tardiness

Notice that the numerator in the response-time bound given in Theorem 1 depends on the core count m and priority k, which determines the number of higher-priority tasks that exist. Therefore, a reduced bound can be ensured if these values can be lowered. One way to do this is by partitioning the hardware platform into clusters of cores, assigning each task to one cluster, and applying global scheduling only within clusters. To evaluate the efficacy of such a strategy, we conducted experiments in which clusters of size two, four, eight, and 16 were considered on

PA	period ascending	PD	period descending
UA	utilization ascending	UD	utilization descending
EA	execution time ascending	ED	execution time descending
A1	Algorithm 1		

Table 1: List of tested heuristics.

a 16-core platform. (A cluster size of 16 is simply pure global scheduling.) For these cluster sizes, we randomly generated npc-sporadic task systems and assessed both schedulability (i.e., the fraction of generated systems deemed schedulable) and tardiness bounds for the generated systems as a function of total system utilization. To assign tasks to clusters, we tried four well-known bin-packing heuristics, worst-fit decreasing, best-fit decreasing, next-fit decreasing, and first-fit decreasing, and declared a task system to be schedulable if any of these heuristics could produce an assignment of tasks to clusters that was schedulable. We assigned higher priorities to lower-indexed tasks (i.e., those generated earlier).

In order to account for variations in task periods, we used *average relative tardiness* as our primary evaluation metric; a task's *relative tardiness* is given by its tardiness divided by its period. We computed both bounds on relative tardiness, using Corollary 2, and observed relative tardiness, by examining schedules in which jobs were released periodically for 10000 time units. The results we obtained for light tasks are plotted in Fig. 9, those for medium tasks in Fig. 10, and those for heavy tasks in Fig. 11. Each plot in these figures pertains to one cluster size and shows schedulability, the average observed relative tardiness, and the average relative tardiness bound (each as a function of total utilization) for that cluster size.

As these plots show, clustering had virtually no negative impact on schedulability for light task systems. For medium and heavy task systems, there was some non-negligible impact for clusters of size two and four, as seen by the decline in schedulability as total utilization nears 16.0. As expected, the likelihood of obtaining a schedulable clustering seems to be lower for the task sets with higher average utilization. However, task systems with a large number of tasks, which gain the largest analytical bound decrease from clustering, have to have low average utilization.

For both light and medium task systems, using clusters of size eight decreased the average relative tardiness bound compared to global scheduling (i.e., using one cluster of size 16) by about 30% with almost no impact on schedulability. Using clusters of size four decreased the bound by around 40%, and using clusters of size two reduced it by around 55%, though for these cluster sizes some schedulability impacts existed for medium task systems, as already noted. Heavy task systems have almost zero relative tardiness, so the tasks number in cluster is smaller, though the analytical bound decreased up to six times for the clusters of size two.

Across all of our experiments, average relative tardiness bounds for light and medium task systems with high total utilization were four to ten times larger than average observed relative tardiness. The difference in the observed results follows from two main reasons. First, the provided bound depends on C_{max}, which tends to be larger for tasks with higher utilization. Second, in general, observed tardiness tended to be smaller for task sets with a smaller number of tasks (which arises from the larger average utilization).

10.2 Task Prioritization

In this subsection, we consider seven task prioritization heuristics (see Tbl. 1). We compared them to an initial random ordering.

As discussed above we have two evaluation metrics: relative average tardiness and relative maximum tardiness.

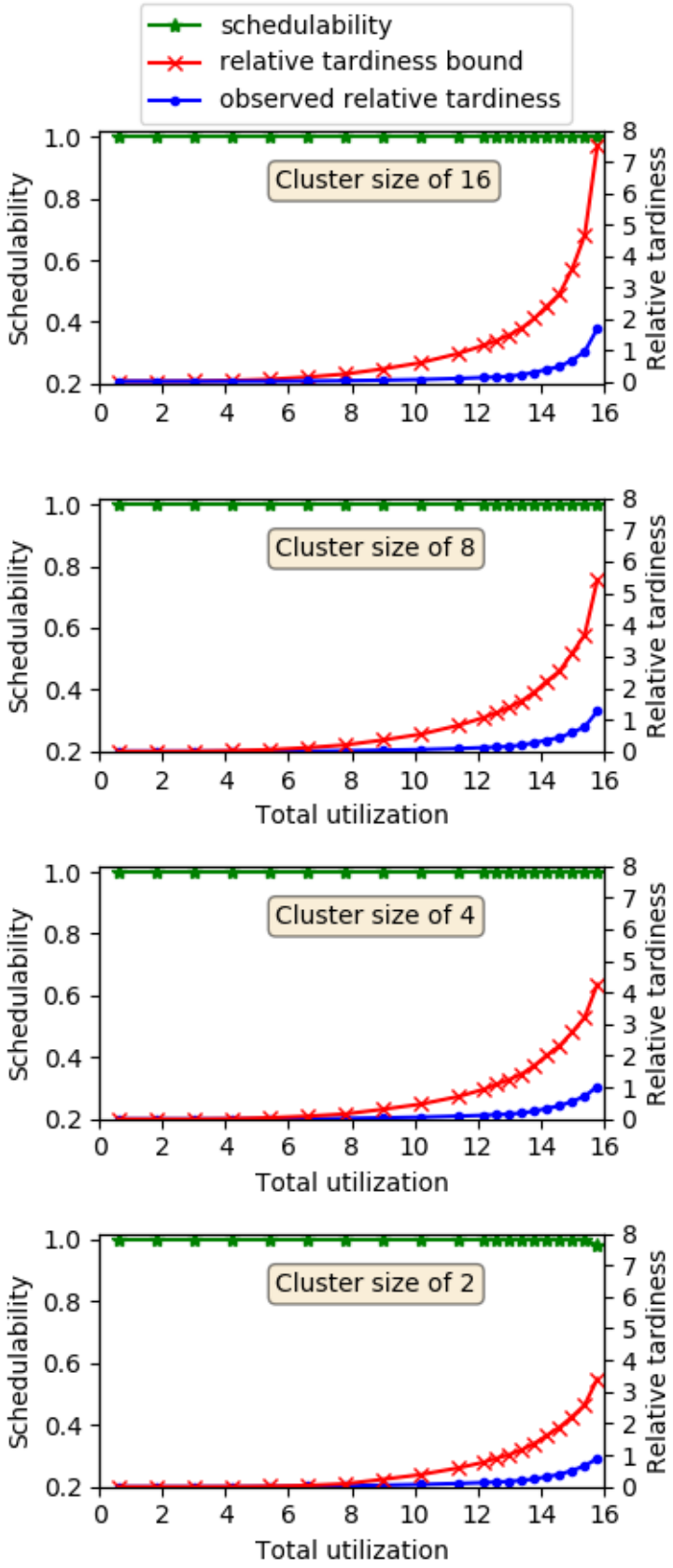


Figure 9: Experimental results for light tasks.

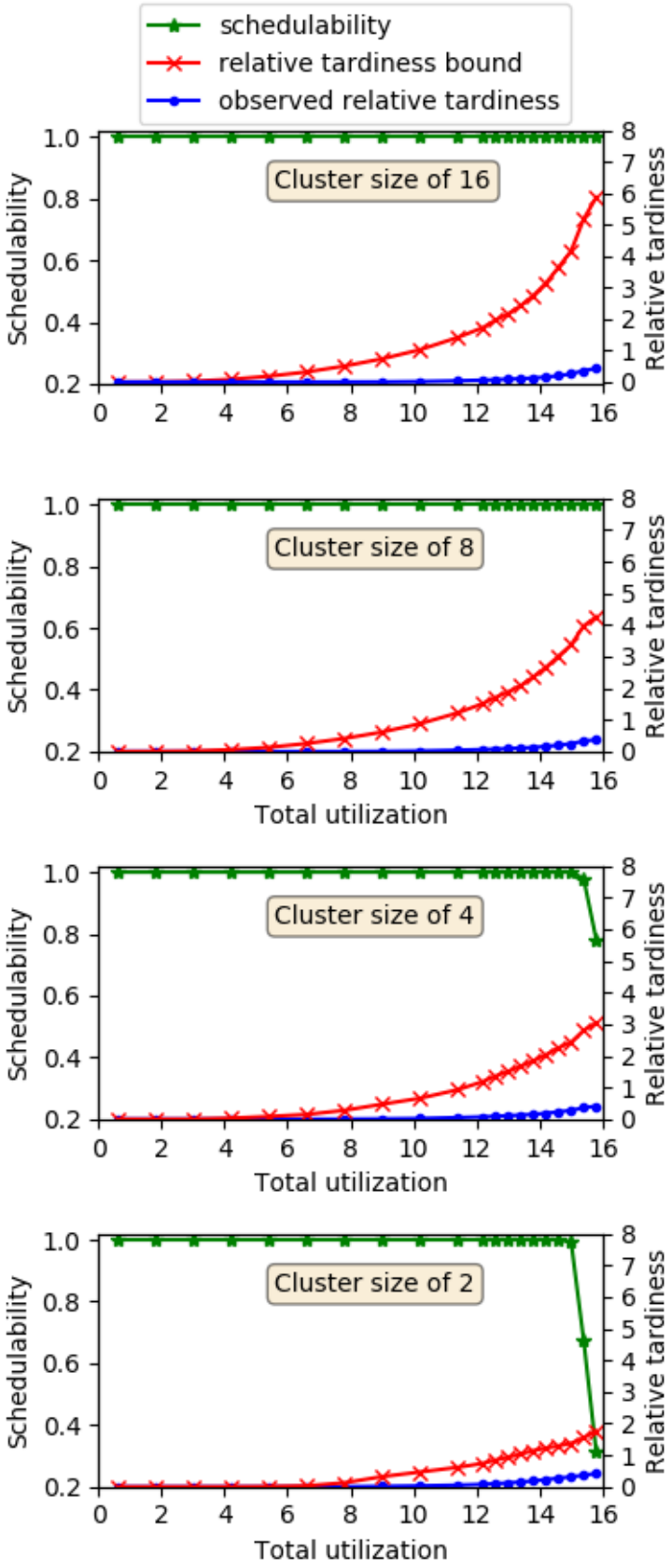


Figure 10: Experimental results for medium tasks.

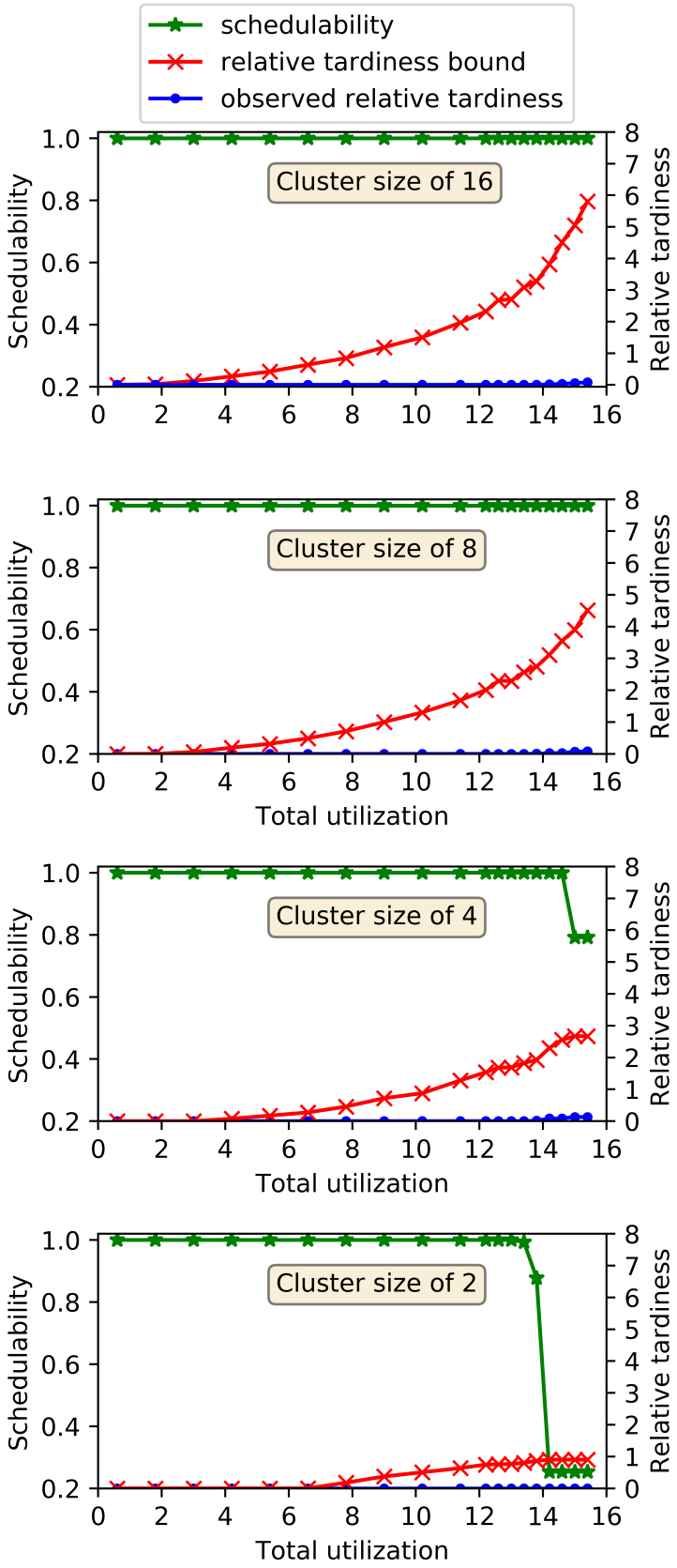


Figure 11: Experimental results for heavy tasks.

function	order	tardiness	max	avg
optimal_max	[5, 3, 1, 2, 4]	0.00 0.38 0.00 1.53 2.01	2.01	0.78
optimal_avg	[3, 2, 1, 5, 4]	0.00 0.00 0.00 0.89 2.01	2.01	0.58
UA	[1, 2, 3, 4, 5]	0.00 0.00 0.07 0.89 2.46	2.46	0.68
EA	[1, 2, 3, 4, 5]	0.00 0.00 0.07 0.89 2.46	2.46	0.68
PA	[2, 1, 3, 4, 5]	0.00 0.00 0.07 0.89 2.46	2.46	0.68
A1	[3, 2, 5, 1, 4]	0.00 0.00 0.48 0.60 2.46	2.46	0.71
UD	[5, 4, 3, 2, 1]	0.00 0.36 1.37 2.58 2.69	2.69	1.40
ED	[5, 4, 3, 1, 2]	0.00 0.36 1.37 1.15 4.57	4.57	1.49
PD	[5, 4, 1, 3, 2]	0.00 0.36 0.00 1.67 4.57	4.57	1.32

Table 2: Relative tardiness from Corollary 2 for the task set {(1, 5); (1, 3); (4, 5); (5, 6); (5, 6)}.

Note that we consider implicit-deadline tasks, so PA in Tbl. 1 is equivalent to the Deadline Monotonic Priority Ordering (DMPO), which is known to be optimal for constrained-deadline sporadic tasks on uniprocessors [22].

Example 4. *Note that all heuristics from the Tbl. 1 may produce sub-optimal prioritizations. For example, consider the following task system (denoted as (execution time, period)):* $\{(1,5);(1,3);(4,5);(5,6);(5,6)\}$. *Tbl. 2 shows relative tardiness for both the average and the maximum case for this task system under each heuristic and also under prioritizations that minimize maximum (optimal_max) and average (optimal_avg) tardiness. As this table shows, none of the heuristics is optimal.*

10.2.1 Heuristics Comparison

In this experiment, we compared the seven heuristics described above with the initial random ordering. We generated task systems of all three types described above: light, medium and heavy for a platform with 16 cores. A comparison of all heuristics can be found in Fig. 12.

As these plots show, three heuristics greatly outperformed the others others: PA, EA, and A1. The difference among their absolute values is not significant, so no one heuristic is best. To provide a more thorough comparison as to which heuristic is best, we limited the scope of the remaining experiments to the three best-performing heuristics. Note that the three best heuristics were able to find prioritizations that ensure almost zero tardiness for nearly all task systems with total utilization in [0, 10]. Thus, we focus on the utilization range [10, 16].

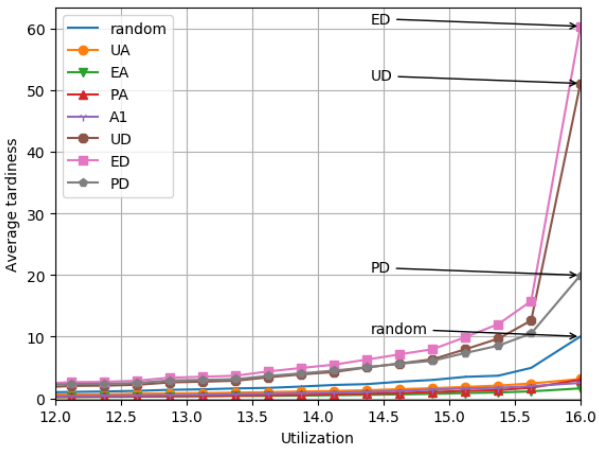
To more precisely compare PA, EA and A1 we computed the share of the generated task systems where each heuristic yields higher schedulability than the other two. A comparison of all three types of task sets on the basis of the two evaluation metrics can be found in Fig. 13.

For all types of task systems with utilization less than the system capacity, the EA heuristic dominates when maximum tardiness is used as the evaluation metric. For task systems with utilization close to the system capacity, the PA heuristic dominates.

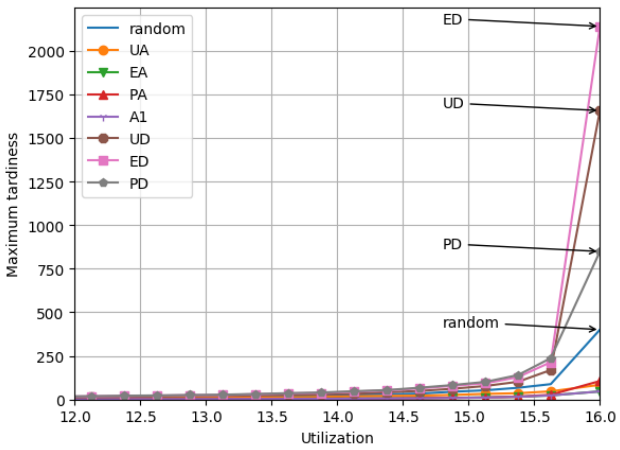
However, when the average relative tardiness is used as the evaluating metric, A1 dominates for almost all task systems with utilizations in the range [10, 15].

10.2.2 Heuristics vs. Optimal

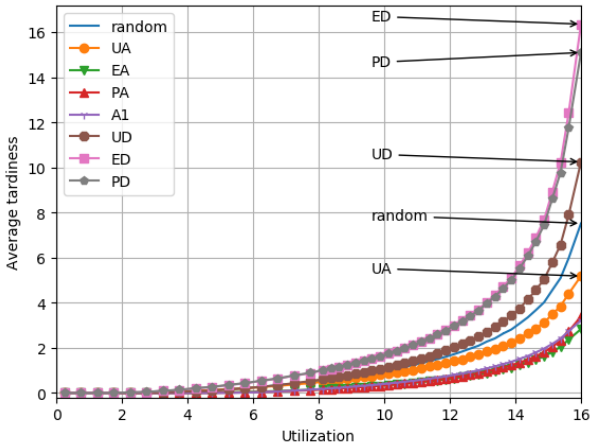
In the subsection above, we considered heuristics for task prioritization. Unfortunately, none of them are optimal. To compute the quality of each heuristic, we need to understand how



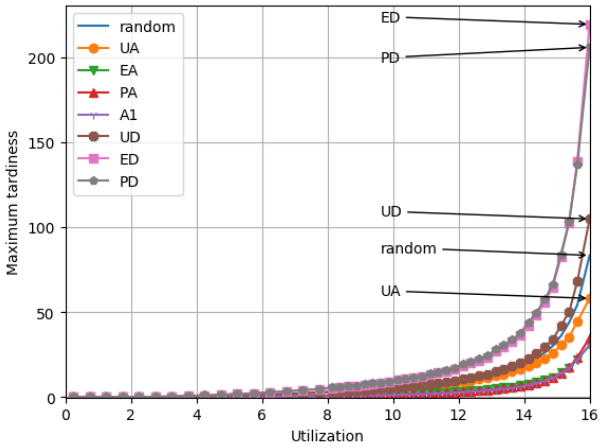
(a) Light tasks and average tardiness



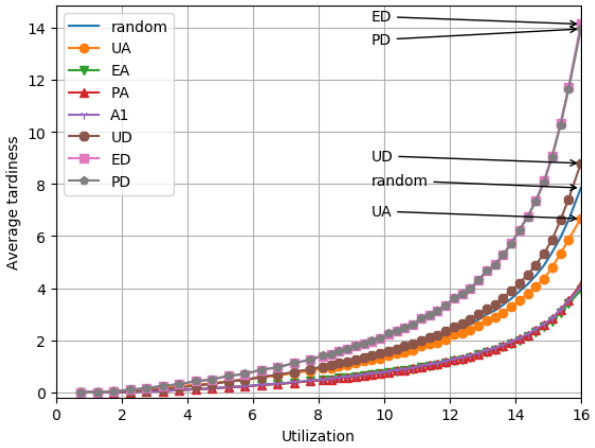
(b) Light tasks and maxumum tardiness



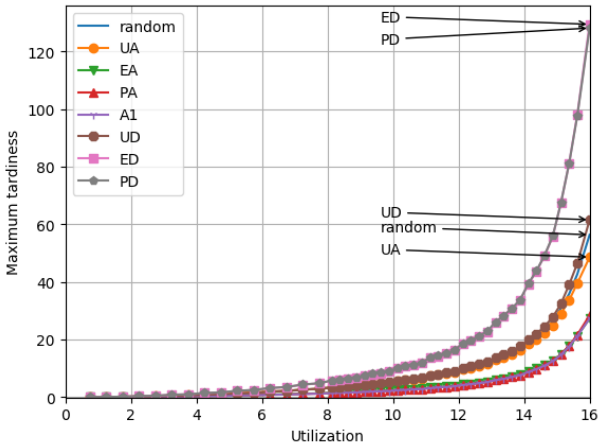
(c) Medium tasks and average tardiness



(d) Medium tasks and maxumum tardiness

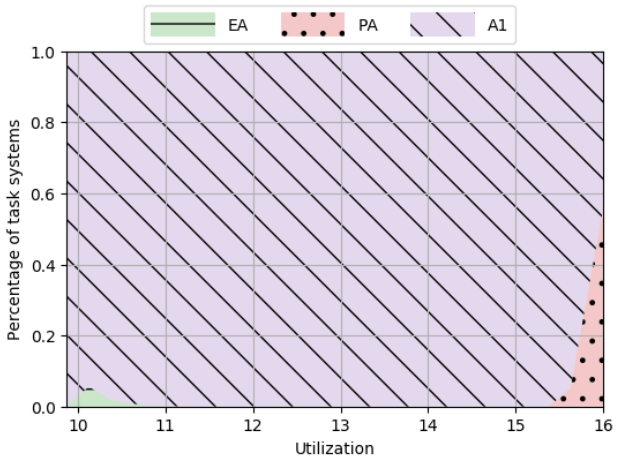


(e) Heavy tasks and average tardiness

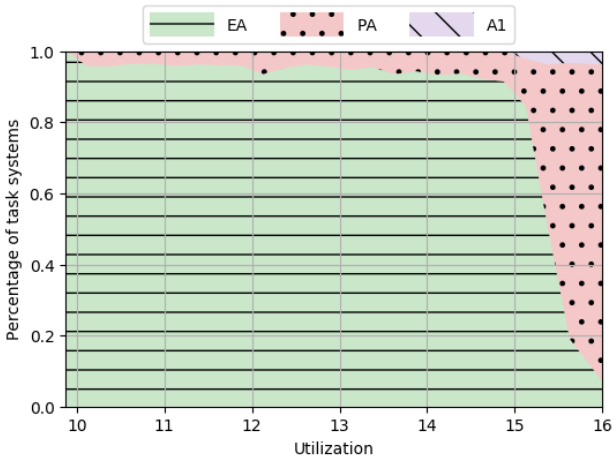


(f) Heavy tasks and maxumum tardiness

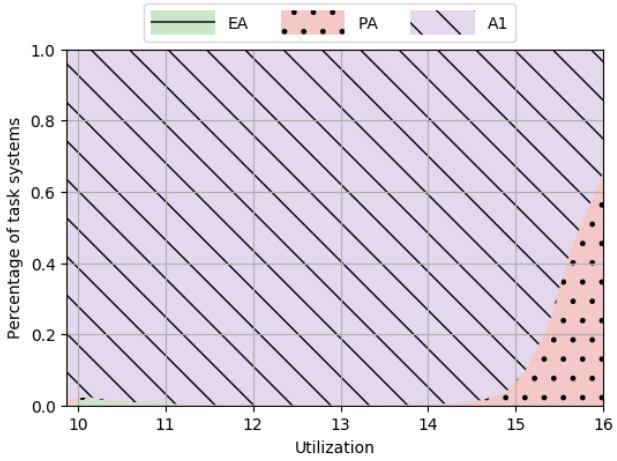
Figure 12: Comparison of all seven heuristics and default random ordering.



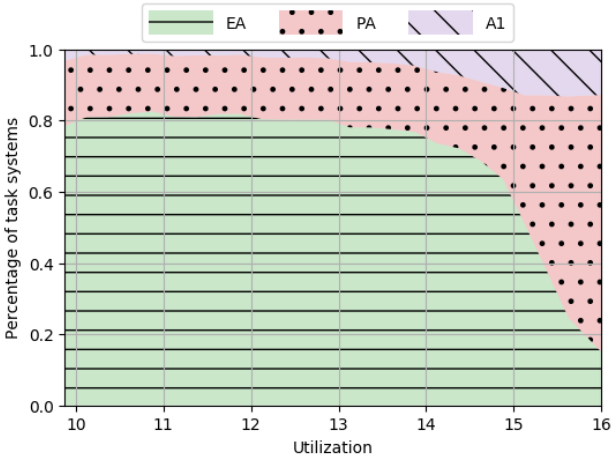
(a) Light tasks and average tardiness



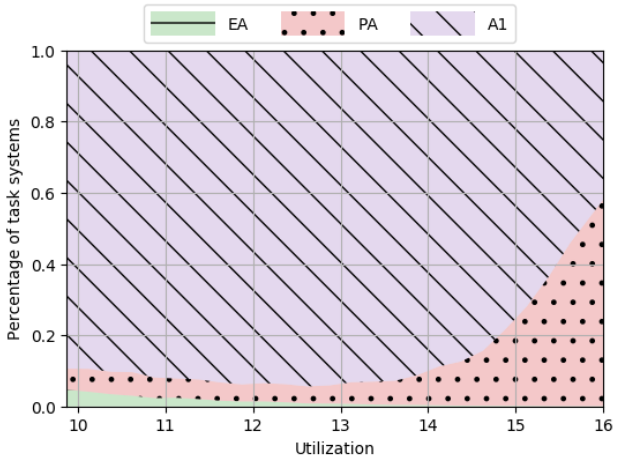
(b) Light tasks and maxumum tardiness



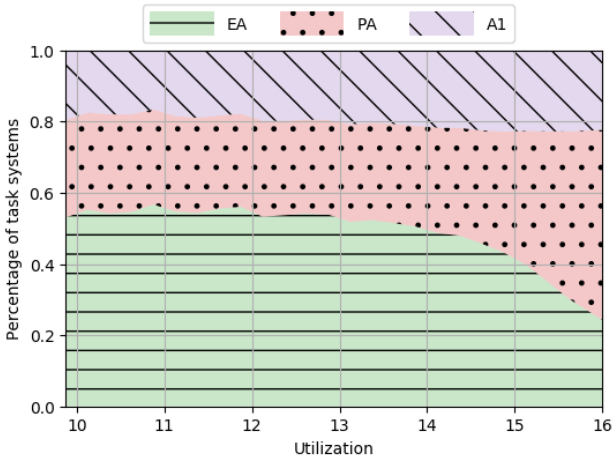
(c) Medium tasks and average tardiness



(d) Medium tasks and maxumum tardiness

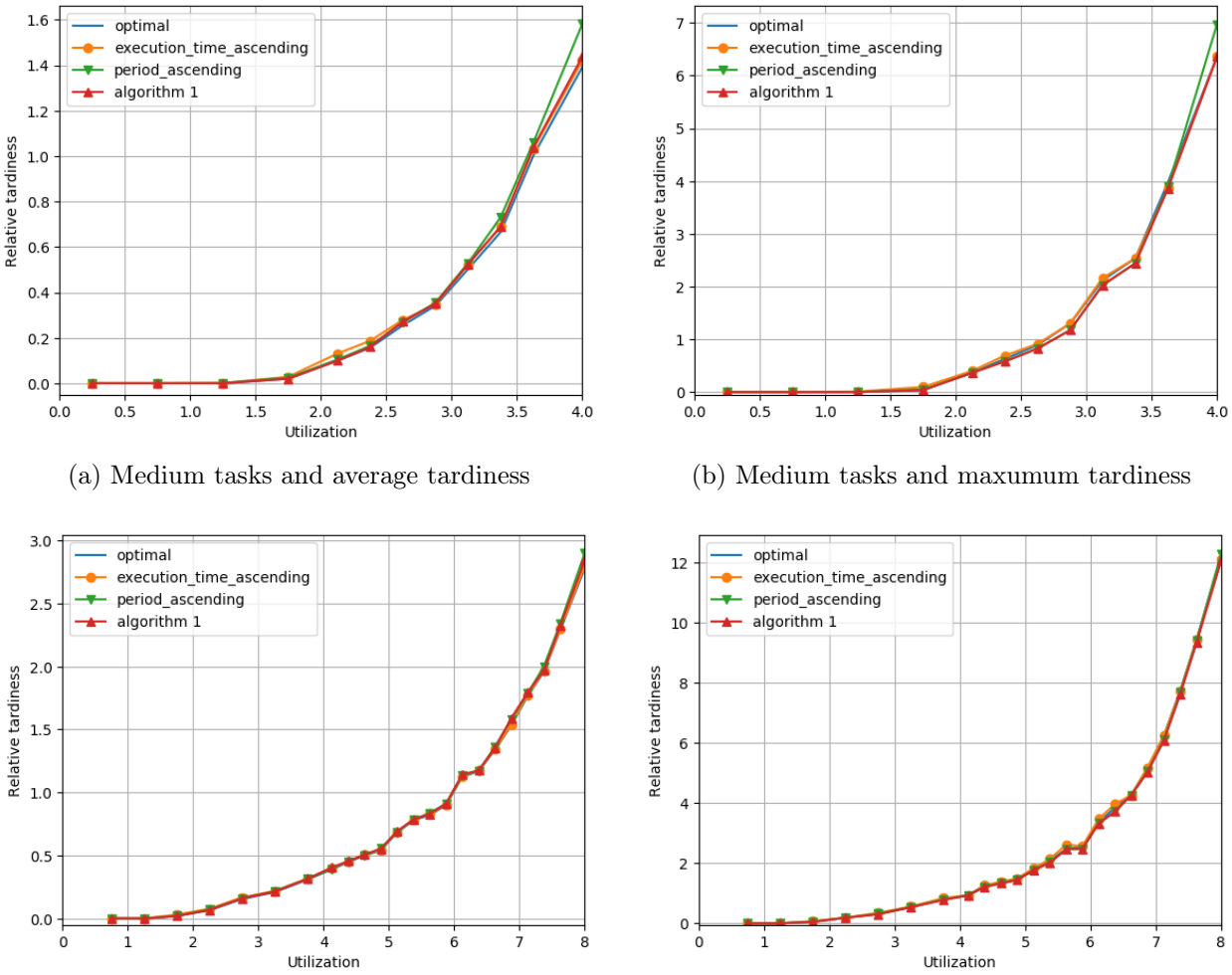


(e) Heavy tasks and average tardiness



(f) Heavy tasks and maxumum tardiness

Figure 13: Comparison of three best heuristics. The y-axis of each plot shows the percentage of tasks sets where each heuristic performed better than two others.



(a) Medium tasks and average tardiness

(b) Medium tasks and maxumum tardiness

(c) Heavy tasks and average tardiness

(d) Heavy tasks and maxumum tardiness

Figure 14: Comparison of three best heuristics.

close to optimal it is. To determine the optimal priority assignment, we computed all priority-assignment permutations and selected the one that yielded the lowest tardiness according to the chosen metric. Such an algorithm is exponential, so we can only test small task systems.

As we see from the previous experiment, three heuristics (EA, PA, A1) performed the best. They outperformed all others for the almost all task systems, so we focused on them only.

Based on the comparison of the relative analytical and observed tardiness, we considered task systems with high per-task utilizations relative to the total system utilization. Such task systems tend to have higher observed and analytical tardiness, so the difference between the optimal algorithm and the other algorithms for the prioritization should be larger. Thus, we focused on medium and heavy tasks (light task systems have lower total utilization). For medium tasks, we considered a 4-core system; for heavy tasks, we considered an 8-core system.

As we can see in Fig. 14, an optimal prioritization does not produce significantly better results for task systems with small numbers of tasks.

11 Conclusion

In this paper, we considered the scheduling of npc-sporadic task systems under G-FP and its variants on a multiprocessor platform. We showed that G-FP (preemptive or non-preemptive) may generate unbounded task response times under the standard sporadic task model, even when the underlying platform is significantly under-utilized. In contrast, under the npc-sporadic

task model, we showed that preemptive G-FP ensures bounded task response times (and hence tardiness) for any task system whose utilization does not exceed the platform's capacity—that is, G-FP is SRT-optimal under the npc-sporadic task model. We further showed that our derived response-time bound is asymptotically tight and that it can be reduced in practice through the use of clustered scheduling. We considered the problem of task system prioritization, and how different heuristics affect the bound. We extended our approach to non-preemptive G-FP and its generalization, G-FP with preemption thresholds.

We showed that any (preemptive or non-preemptive) work-conserving scheduler is SRT-optimal under the npc-sporadic task model. However, this approach may yield conservative bounds for certain schedulers because it does not take into account scheduler-specific information that may be important for obtaining reduced bounds. In the future, we hope to refine this proof strategy so that it can be applied to obtain an asymptotically tight response-time bound for any such scheduler.

This paper was motivated by an industry problem (pertaining to the processing done within 5G cellular base stations) in which tasks exist as nodes within a directed acyclic graph (DAG), the edges of which denote precedence constraints between different tasks, and intra-task parallelism is allowed. In prior work, response-time bounds for such DAG-based systems were presented assuming a dynamic-priority scheduler is used [30]. In this industry problem, a static-priority scheduler would be desirable to use because it would entail lower runtime overheads than a dynamic-priority one. In the future work, we intend to consider in detail the applicability of G-FP scheduling under the npc-sporadic task model in this DAG-based setting.

References

[1] URL https://github.com/rdkl/npc-sporadic-task-model

[2] Audsley, N.: Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Citeseer (1991)

[3] Baker, T.: Comparison of empirical success rates of global vs. partitioned fixed-priority and EDF scheduling for hard real time. Tech. Rep. TR-050601, Department of Computer Science, Florida State University (2005)

[4] Baker, T., Baruah, S.: An analysis of global EDF schedulability for arbitrary-deadline sporadic task systems. Real-Time Systems **43**(1), 3–24 (2009)

[5] Baruah, S., Baker, T.: Schedulability analysis of global EDF. Real-Time Systems **38**(3), 223–235 (2008)

[6] Baruah, S., Fisher, N.: Global fixed-priority scheduling of arbitrary-deadline sporadic task systems. In: Proceedings of the 9th International Conference on Distributed Computing and Networking, pp. 215–226 (2008)

[7] Bertogna, M., Cirinei, M., Lipari, G.: Improved schedulability analysis of EDF on multiprocessor platforms. In: Proceedings of the 17th Euromicro Conference on Real-Time Systems, pp. 209–218 (2005)

[8] Bril, R., Altmeyer, S., van den Heuvel, M., Davis, R., Behnam, M.: Fixed priority scheduling with pre-emption thresholds and cache-related pre-emption delays: integrated analysis and evaluation. Real-Time Systems (2017)

[9] Davis, R., Burns, A.: Priority assignment for global fixed-priority preemptive scheduling in multiprocessor real-time systems. In: Proceedings of the 30th IEEE Real-Time Systems Symposium, pp. 398–409 (2009)

[10] Davis, R., Burns, A.: Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. Real-Time Systems **47**(1), 1–40 (2011)

[11] Devi, U.: Soft real-time scheduling on multiprocessors. Ph.D. thesis, University of North Carolina at Chapel Hill (2006)

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

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

[14] Erickson, J., Anderson, J.: Response time bounds for G-EDF without intra-task precedence constraints. In: Proceedings of the 15th International Conference On Principles Of Distributed Systems, pp. 128–142 (2011)

[15] Express Logic Inc.: Threadx (2019). URL https://rtos.com/solutions/threadx/real-time-operating-system/

[16] Gai, P., Lipari, G., Di Natale, M.: Minimizing memory utilization of real-time task sets in single and multi-processor systems-on-a-chip. In: Proceedings of the 22nd IEEE Real-Time Systems Symposium, pp. 73–83 (2001)

[17] Ghattas, R., Dean, A.: Preemption threshold scheduling: Stack optimality, enhancements and analysis. In: Proceedings of the 13th IEEE Real Time and Embedded Technology and Applications Symposium, pp. 147–157 (2007)

[18] Keskin, U., Bril, R., Lukkien, J.: Exact response-time analysis for fixed-priority preemption-threshold scheduling. In: Proceedings of the 15th IEEE Conference on Emerging Technologies & Factory Automation, pp. 1–4. IEEE (2010)

[19] Leontyev, H., Anderson, J.: Tardiness bounds for EDF scheduling on multi-speed multicore platforms. In: Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, pp. 103–110 (2007)

[20] Leung, J.: A new algorithm for scheduling periodic, real-time tasks. Algorithmica **4**(1-4), 209 (1989)

[21] Leung, J., Merrill, M.: A note on preemptive scheduling of periodic, real-time tasks. Information processing letters **11**(3), 115–118 (1980)

[22] Leung, J., Whitehead, J.: On the complexity of fixed-priority scheduling of periodic, real-time tasks. Performance evaluation **2**(4), 237–250 (1982)

[23] Li, C., Lee, C.: Scheduling with agreeable release times and due dates on a batch processing machine. European Journal of Operational Research **96**(3), 564–569 (1997)

[24] Regehr, J.: Scheduling tasks with mixed preemption relations for robustness to timing faults. In: Proceedings of the 23rd IEEE Real-Time Systems Symposium, pp. 315–326 (2002)

[25] Saksena, M., Wang, Y.: Scalable real-time system design using preemption thresholds. In: Proceedings 21st IEEE Real-Time Systems Symposium, pp. 25–34. IEEE (2000)

[26] Uzsoy, R.: Scheduling a single batch processing machine with non-identical job sizes. The International Journal of Production Research **32**(7), 1615–1635 (1994)

[27] Voronov, S., Anderson, J., Yang, K.: Tardiness bounds for fixed-priority global scheduling without intra-task precedence constraints. In: Proceedings of the 26th International Conference on Real-Time Networks and Systems, pp. 8–18 (2018)

[28] Wang, Y., Saksena, M.: Scheduling fixed-priority tasks with preemption threshold. In: Proceedings Sixth International Conference on Real-Time Computing Systems and Applications. RTCSA'99 (Cat. No. PR00306), pp. 328–335. IEEE (1999)

[29] Yang, K., Anderson, J.: Optimal GEDF-based schedulers that allow intra-task parallelism on heterogeneous multiprocessors. In: Proceedings of the 12th IEEE Symposium on Embedded Systems for Real-Time Multimedia, pp. 30–39 (2014)

[30] Yang, K., Yang, M., Anderson, J.: Reducing response-time bounds for DAG-based task systems on heterogeneous multicore platforms. In: Proceedings of the 24th International Conference on Real-Time Networks and Systems, pp. 349–358 (2016)

[31] Yang, M., Amert, T., Yang, K., Otterness, N., Anderson, J., Smith, F., Wang, S.: Making OpenVX really 'real time'. In: Proceedings of the 39th IEEE Real-Time Systems Symposium, pp. 80–93 (2018)