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

Sergey Voronov
Univ. of North Carolina at Chapel Hill
rdkl@cs.unc.edu

James H. Anderson
Univ. of North Carolina at Chapel Hill
anderson@cs.unc.edu

Kecheng Yang
Texas State University
yangk@txstate.edu

ABSTRACT

Fixed-priority multicore 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 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 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.

CCS CONCEPTS

• **Computer systems organization → Real-time systems**;

KEYWORDS

global fixed-priority scheduling, intra-task parallelism, tardiness, optimality, multiprocessors

ACM Reference Format:
Sergey Voronov, James H. Anderson, and Kecheng Yang. 2018. Tardiness Bounds for Fixed-Priority Global Scheduling without Intra-Task Precedence Constraints. In *26th International Conference on Real-Time Networks and Systems (RTNS '18), October 10–12, 2018, Chasseneuil-du-Poitou, France.* RTNS'18, Poitiers/Futuroscope, France, 11 pages. https://doi.org/10.1145/3273905.3273913

ACKNOWLEDGMENTS

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.

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., [1, 3–6, 11]). 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* [8]. 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 [7].

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 [7] 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.

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 [2] and SRT systems [10, 12]. The latter work showed that allowing intra-task parallelism enables much lower tardiness bounds to be derived. Following [12], 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

RTNS '18, October 10–12, 2018, Chasseneuil-du-Poitou, France

ACM ISBN 978-1-4503-6463-8/18/10…$15.00
https://doi.org/10.1145/3273905.3273913

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 [10, 12]. A particularly compelling use case is computer-vision (or radar) object detection [12]. 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 [14], 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. In our experiments, we also compared relative tardiness bounds obtained from our analysis vs. observed average relative tardiness. (A task relative tardiness is given by its tardiness divided by its period.) We found that bounds for task sets with high total utilization tended to be four to ten times larger than observed relative tardiness.

Due to space constraints, we mostly limit attention to preemptive G-FP in this paper. However, as discussed later, our response-time bound proof can be adjusted to apply to non-preemptive G-FP (and, in fact, to any work-conserving global scheduler). Thus, *non-preemptive G-FP is SRT-optimal as well.*

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.

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), present our experimental results (Sec. 6), and conclude (Sec. 7).

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 cores, $\pi_1, \ldots, \pi_m$. 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 denotes the worst-case execution time of task τ_i, T_i denoted 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 assume that tasks are indexed by priority, with higher-priority tasks having lower indexes. We also assume that time is continuous. We denote the overall system utilization by $U = \sum_{i=1}^{n} u_i$ and the total utilization of tasks $\tau_1, \ldots, \tau_\ell$ by $U_\ell = \sum_{i=1}^{\ell} u_i$.

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. In this paper, we focus on establishing the SRT-optimality of the preemptive G-FP scheduler. Hereafter, all references to G-FP without qualification should be taken to mean *preemptive* G-FP. For simplicity, we assume unique task priorities. Moreover, we assume that jobs of the same task are prioritized against each other on a first-in-first-out (FIFO) basis. This assumption is implicit in the conventional sporadic task model (which precludes a job from starting until the previous job of the same task completes).

Although the potential for conventional sporadic tasks to have unbounded response times under preemptive G-FP has been shown previously [7], 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 2.1. G-FP with preemption. 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 4.6 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 2.2. G-FP without preemption. 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.

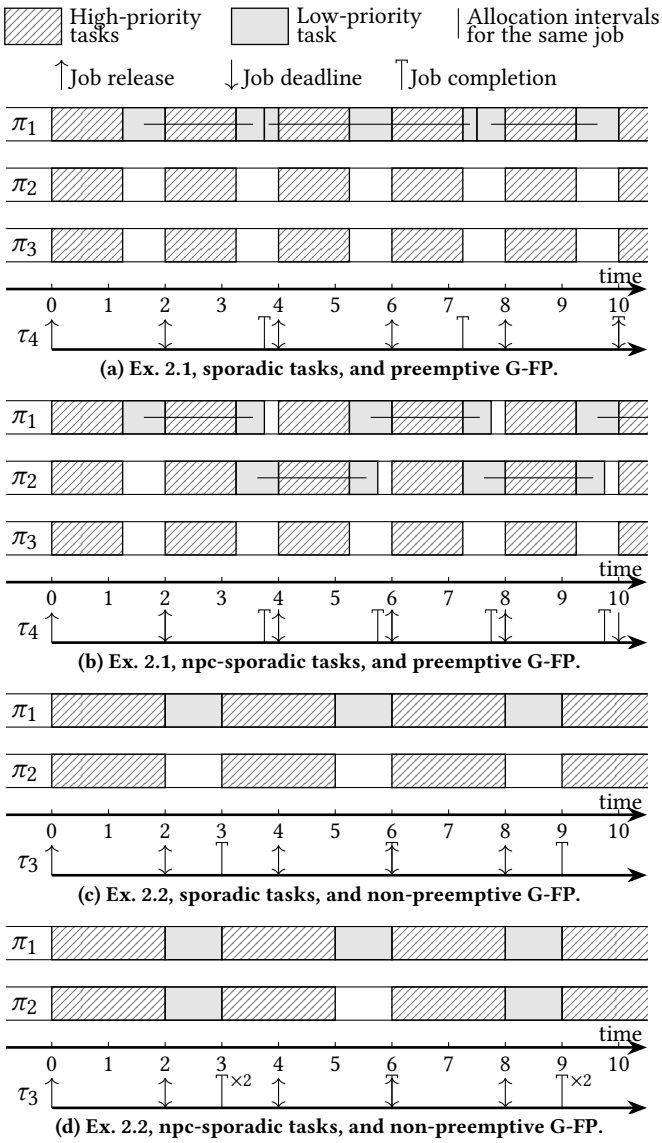

Figure 1: Schedules for the task systems in Exs. 2.1 and 2.2.

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. □

Proof setup. In reasoning about the job of interest $J_{k,d}$, we make the following simplifying assumptions.

- **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 $J_{k,d}$).

Assumption A1 is tantamount to discarding all tasks with a priority lower than τ_k; these tasks do not affect the scheduling of tasks with higher priority under preemptive G-FP scheduling. Enforcing Assumptions A2 and A3 cannot reduce $J_{k,d}$'s response time (A3 affects only jobs of τ_k released after $J_{k,d}$, while A2 may only increase the execution times of $J_{k,d}$, $J_{k,d+1}$, ...); thus, any response-time bound derived with these assumptions in place is still valid if they are weakened.

We now can formalize our problem: *find a bound for $J_{k,d}$'s response time under A1-A3.* In considering this problem, we make use of some additional terms and notation, which we introduce next.

Definition 2.3. We let $r_{i,j}$ and $R_{i,j}$ denote the release and response times, respectively, of job $J_{i,j}$. For conciseness, we use r and R in reference to the job of interest instead of $r_{k,d}$ and $R_{k,d}$.

Definition 2.4. 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 [9], we use the concept of lag, which we define by considering an "ideal" platform π' consisting of k cores, $\pi'_1, \ldots, \pi'_k$, with speeds $u_1, \ldots, u_k$, respectively (by A1, we have only k tasks). A core'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 core 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 cores). 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 2.5. We denote as $\mathcal{I}$ the ideal schedule of the task set $\{\tau_1, \ldots, \tau_k\}$ on π' as described above. Also, we denote as $\mathcal{S}$ the schedule of $\{\tau_1, \ldots, \tau_k\}$ produced by our G-FP scheduler on the actual platform π with m unit-speed cores.

Definition 2.6. For an arbitrary schedule $\mathcal{H}$, we denote as $\mathcal{A}(\mathcal{H}, t_1, t_2, J_{i,j})$ the overall processor capacity allocated to job $J_{i,j}$ in $\mathcal{H}$ within the interval $[t_1, t_2)$. Note that if $J_{i,j}$ is released at or after t_2, or $t_1 > t_2$, then by definition, $\mathcal{A}(\mathcal{H}, t_1, t_2, J_{i,j}) = 0$.

Definition 2.7. The lag for job $J_{i,j}$ is defined as $\text{lag}(J_{i,j}, t) = \mathcal{A}(\mathcal{I}, 0, t, J_{i,j}) - \mathcal{A}(\mathcal{S}, 0, t, J_{i,j})$.

Definition 2.8. The lag for task τ_i is defined as $\text{Lag}(\tau_i, t) = \sum_j \text{lag}(J_{i,j}, t)$.

Definition 2.9. The total lag of the entire considered task system is defined as $\text{LAG}(t) = \sum_i \text{Lag}(\tau_i, t) = \sum_i \sum_j \text{lag}(J_{i,j}, t)$.

3 LAG BOUNDS

In Sec. 3.1 below, we derive a number of lag-related properties, as well as a lower bound on task Lag. In Sec. 3.2, we derive an upper bound on total system LAG.

3.1 Task Lag Lower Bound

The general property of Lag for a single task we establish in this subsection is formulated in Lemma 3.5. To prove this lemma, we first prove several lemmas concerning job lag.

Lemma 3.1. *For any time t before the release of job $J_{i,j}$ or after its completion in both schedules $\mathcal{I}$ and $\mathcal{S}$,* $\text{lag}(J_{i,j}, t) = 0$.

Proof. If $J_{i,j}$ is unreleased, then by Def. 2.6, $\mathcal{A}(\mathcal{I}, 0, t, J_{i,j})$ 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. 2.6, $\mathcal{A}(\mathcal{I}, 0, t, J_{i,j}) = \mathcal{A}(\mathcal{S}, 0, t, J_{i,j}) = C_{i,j}$. In both cases, by Def. 2.7, $\text{lag}(J_{i,j}, t) = 0$. □

Lemma 3.2. *If $t \geq r_{i,j} + T_i$, then $\text{lag}(J_{i,j}, t) \geq 0$.*

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. 2.7, the lemma follows. □

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

Lemma 3.3. $\min(0, (u_i - 1)C_i) \leq \text{lag}(J_{i,j}, t) \leq C_i$.

Proof. According to Def. 2.7,

$$\begin{aligned}\text{lag}(J_{i,j}, t) &= \mathcal{A}(\mathcal{I}, 0, t, J_{i,j}) - \mathcal{A}(\mathcal{S}, 0, t, J_{i,j}) \\ &\leq \mathcal{A}(\mathcal{I}, 0, t, J_{i,j}) \\ &\leq \{\text{by Def. 2.6}\} \\ &\quad C_{i,j} \\ &\leq C_i,\end{aligned}$$

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 3.1, $\text{lag}(J_{i,j}, t) = 0$.

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. 2.5) during $[r_{i,j}, r_{i,j} + f \cdot T_i)$ on a core 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 $\text{lag}(J_{i,j}) \geq 0$ 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

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

Case 3. $t \in [r_i + f \cdot T_i, \infty)$. By Def. 2.5 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 $\text{lag}(J_{i,j}, t) = \mathcal{A}(\mathcal{I}, 0, t, J_{i,j}) - \mathcal{A}(\mathcal{S}, 0, t, J_{i,j}) \geq C_{i,j} - C_{i,j} = 0$.

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

Corollary 3.4. $\text{lag}(J_{i,j}, t) < 0$ *implies* $t \in [r_{i,j}, r_{i,j} + T_i)$.

Proof. In Cases 1 and 3 in the proof of Lemma 3.3 above, we proved that $\text{lag}(J_{i,j}, t) \geq 0$ holds. Thus, if $\text{lag}(J_{i,j}, t) < 0$ 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 3.5. $\min(0, (u_i - 1)C_i) \leq \text{Lag}(\tau_i, t)$.

Proof. By Corollary 3.4, $\text{lag}(J_{i,j}, t) < 0$ 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. 2.5, at most one job per task may be scheduled in $\mathcal{I}$ at any time instant. Therefore, in $\sum_j \text{lag}(J_{i,j}, t)$, 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 $\text{lag}(\tau_{i,h}, t) < 0$ holds, then for any $j \neq h$, $\text{lag}(J_{i,j}, t) \geq 0$. Thus, by Lemma 3.3

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

3.2 System LAG Upper Bound

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 cores schedule different tasks.

Lemma 3.6. $\text{lag}(J_{i,j}, t)$, $\text{Lag}(\tau_i, t)$, *and* $\text{LAG}(t)$ *are continuous functions 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. 2.7, $\text{lag}(J_{i,j}, t)$ is continuous because

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

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 3.1, $\text{lag}(J_{i,j}, t) = 0$ for all $j > h$. Thus,

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

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

$$\text{LAG}(t) = \sum_{i \leq k} \text{Lag}(\tau_i, t). \square$$

Lemma 3.7. *For any time interval* $[t_1, t_2)$

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

$$\sum_{i,j} \mathcal{A}(\mathcal{I}, t_1, t_2, J_{i,j}) \leq U_k(t_2 - t_1).$$

Proof. At every time instant in the ideal schedule $\mathcal{I}$ there is at most one job from task τ_i scheduled (see Def. 2.5). The speed of the core that schedules τ_i is u_i, while the length of the considered interval is $(t_2 - t_1)$. Thus,

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

$$\sum_{i,j} \mathcal{A}(\mathcal{I}, t_1, t_2, J_{i,j}) = \sum_i \left(\sum_j \mathcal{A}(\mathcal{I}, t_1, t_2, J_{i,j}) \right)$$
$$\leq \{\text{by (1)}\}$$
$$\sum_i u_i (t_2 - t_1)$$
$$= (t_2 - t_1) \sum_i u_i$$
$$= (t_2 - t_1) U_k. \ \square$$

Definition 3.8. We call a core *busy* at time instant t if there exists a job scheduled for execution on this core at time t. Otherwise, the core is *idle*.

Our upper bound on system LAG is given by the following lemma.

Lemma 3.9. $\mathsf{LAG}(t) \leq (\lceil U_k \rceil - 1) C_{max}$, *where* $C_{max} = \max_{i \leq k} \{C_i\}$.

Proof. To prove this lemma, we split the timeline $[0, +\infty)$ into a set of maximal continuous intervals such that the number of busy cores 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 cores in $\mathcal{S}$ during I does not change.

I2: I cannot be extended without violating I1.

On an identical multiprocessor G-FP, may alter the set of scheduled jobs at time instant t only if some job completes or a new job is released at t. Since in the npc-sporadic task system τ any finite time interval I contains a finite number of such events, 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

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

Let t_1 be the beginning of I_b. Then $t_1 \leq t_2$. We next show that (3) below holds.

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

If $t_1 = 0$, then $\mathsf{LAG}(0) = 0$, so (3) holds, since $U_k \geq u_k > 0$. 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,

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

By Lemma 3.6, $\mathsf{LAG}(t)$ is a continuous function, so

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

i.e., (3) holds.

Thus, we have

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

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

$$\sum_{i,j} \mathcal{A}(\mathcal{S}, t_1, t_2, J_{i,j}) = \mathrm{bc} \cdot (t_2 - t_1). \tag{5}$$

Combining Lemma 3.7 and (5), we can bound the change of LAG over $[t_1, t_2)$:

$$\mathsf{LAG}(t_2) - \mathsf{LAG}(t_1)$$
$$= \sum_{i,j} (\mathcal{A}(\mathcal{I}, t_1, t_2, J_{i,j}) - \mathcal{A}(\mathcal{S}, t_1, t_2, J_{i,j}))$$
$$\{\text{rearranging}\}$$
$$= \sum_{i,j} \mathcal{A}(\mathcal{I}, t_1, t_2, J_{i,j}) - \sum_{i,j} \mathcal{A}(\mathcal{S}, t_1, t_2, J_{i,j})$$
$$\leq \{\text{by Lemma 3.7 and (5)}\}$$
$$U_k (t_2 - t_1) - \mathrm{bc} \cdot (t_2 - t_1)$$
$$= (U_k - \mathrm{bc})(t_2 - t_1).$$

Thus, by (4), $(U_k - \mathrm{bc})(t_2 - t_1) > 0$. By the definition of t_1 and t_2, $t_1 \leq t_2$, so $U_k - \mathrm{bc} > 0$. Thus,

$$\mathrm{bc} \leq \lceil U_k \rceil - 1. \tag{6}$$

Since $U_k \leq m$, we have $\mathrm{bc} < m$. Thus, there is at least one non-busy core, and bc is the number of uncompleted jobs t_2^-. We can now derive a bound for $\mathsf{LAG}(t_2^-)$:

$$\mathsf{LAG}(t_2^-) = \sum_i \mathsf{Lag}(\tau_i, t_2^-)$$
$$\leq \{\text{by Def. 2.9 and Lemma 3.1}\}$$
$$\sum_{J_{i,j} \text{ is uncompleted}} \mathsf{lag}(J_{i,j}, t_2^-)$$
$$\leq \{\text{by Lemma 3.3}\}$$
$$\sum_{J_{i,j} \text{ is uncompleted}} C_i$$
$$\leq \{\text{bc is the number of uncompleted jobs and } C_{max} \geq C_i\}$$
$$\mathrm{bc} \cdot C_{max}$$
$$\leq \{\text{by (6)}\}$$
$$(\lceil U_k \rceil - 1) \cdot C_{max}. \tag{7}$$

By Lemma 3.6, $\mathsf{LAG}(t_2) = \mathsf{LAG}(t_2^-)$, so (7) contradicts (2). This contradiction finishes the proof. $\square$

4 RESPONSE-TIME BOUND

In this section, we focus our attention on the job of interest $J_{k,d}$ with release time r and response time R.

Lemma 4.1. *None of the jobs from* τ_k *that are released after* $J_{k,(d+m-1)}$ *can be scheduled in* $\mathcal{S}$ *before the completion of* $J_{k,d}$.

Proof. Recall that, by A1-A3 (stated in Sec. 2), τ_k becomes periodic with a fixed execution time C_k and period T_k, starting from $r = r_{k,d}$. This is illustrated in Fig. 2.

Job $J_{k,d+m}$ cannot be scheduled at any time $t < r + mT_k = r_{k,d+m}$ because it is not ready; the same is true of jobs from τ_k released after $J_{k,d+m}$. Thus, if $J_{k,d}$ completes earlier than $r + mT_k$, then the lemma statement is true. Otherwise $R > r + mT_k$ and the interval $[r + mT_k, R)$ is non-empty. Let us consider any t from this interval. Note the following:

- $J_{k,d}$ is not completed (by Def. 2.3 of R) at t.
- Jobs $J_{k,d+1}, \ldots, J_{k,d+m-1}$ are ready at t (their release times are $r + T_k, \ldots, r + (m-1)T_k$) and not completed (by A2, they have the same execution time as $J_{k,d}$, C_k, and they have lower priority than $J_{k,d}$).

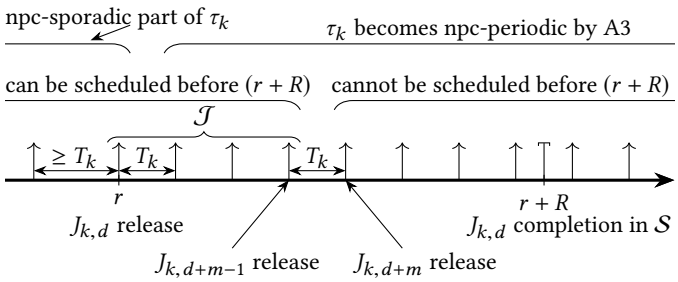

Figure 2: Lemma 4.1 illustration with $m = 4$.

- All jobs from τ_k following $J_{k,d+m}$ have lower priorities.

Thus, jobs $J_{k,d+m}, J_{k,d+m+1}, \ldots$ are not scheduled in $\mathcal{S}$ at t (there are m cores and at least m jobs with higher priority are ready). □

Lemma 4.1 allows us to discard from consideration all jobs following $J_{k,d+m-1}$ because their scheduling intervals do not intersect with that of $J_{k,d}$.

Definition 4.2. Let us define $\mathcal{J} = \{J_{k,d}, J_{k,d+1}, \ldots, J_{k,d+m-1}\}$. By Lemma 4.1, $\mathcal{J}$ contains all jobs from τ_k following the job of interest $J_{k,d}$ that can be scheduled in parallel with $J_{k,d}$.

Definition 4.3. Let us denote as W the overall processor allocation to jobs from $\mathcal{J}$ in $\mathcal{S}$ in the interval $[r, r + R)$. More formally,

$$W = \sum_{j=d}^{d+m-1} \mathcal{A}(\mathcal{S}, r, r + R, J_{k,j}).$$

No job from $\mathcal{J}$ is ready during $[0, r)$, so

$$W = \sum_{j=d}^{d+m-1} \mathcal{A}(\mathcal{S}, 0, r + R, J_{k,j}). \tag{8}$$

Moreover, by Lemma 4.1, no job of τ_k beginning with $J_{k,d+m}$ can be scheduled in $\mathcal{S}$ during $[r, r + R)$. Thus, W is the total allocation of all jobs from task τ_k following the job of interest $J_{k,d}$ from its release until its completion in $\mathcal{S}$.

Using Def. 4.3, we can compute a lower bound on $\mathrm{Lag}(\tau_k, r+R)$ by an alternative approach compared to that used to prove Lemma 3.5.

Lemma 4.4. $\mathrm{Lag}(\tau_k, r + R) = u_k R - W$.

Proof. By Def. 2.8, $\mathrm{Lag}(\tau_k, t) = \sum_j \mathrm{lag}(J_{k,j}, t)$. We split all jobs from τ_k into groups by their job index: $J_{k,j}$ with $j \in [1, d)$, $j \in [d, d + m)$, and $j \in [d + m, \infty)$.

Case 1. $J_{k,j}$ with $j \in [1, d)$. All jobs from task τ_k are prioritized against each other in FIFO order, and each has an execution time not exceeding C_k. Moreover, by A2 (from Sec. 2), the execution time of $J_{k,d}$ is exactly C_k. Thus, any such $J_{k,j}$ would be completed in $\mathcal{S}$ by time $r + R$. In addition, $J_{k,j}$ would be also completed in $\mathcal{I}$ due to the definition of $\mathcal{I}$ given in Sec. 2. Hence, $\mathrm{lag}(J_{k,j}, r + R) = 0$, and

$$\sum_{j=1}^{d-1} \mathrm{lag}(J_{k,j}, r + R) = 0. \tag{9}$$

Case 2. $J_{k,j}$ with $j \in [d, d + m)$. For that case, we have

$$\begin{aligned}
&\sum_{j=d}^{d+m-1} \mathrm{lag}(J_{k,j}, r + R) \\
&= \{\text{by Def. 2.7}\} \\
&\sum_{j=d}^{d+m-1} \mathcal{A}(\mathcal{I}, 0, r + R, J_{k,j}) - \sum_{j=d}^{d+m-1} \mathcal{A}(\mathcal{S}, 0, r + R, J_{k,j}) \\
&= \{\text{by (8)}\} \\
&\sum_{j=d}^{d+m-1} \mathcal{A}(\mathcal{I}, 0, r + R, J_{k,j}) - W.
\end{aligned} \tag{10}$$

Case 3. $J_{k,j}$ with $j \in [d + m, \infty)$. By Lemma 4.1, job $J_{k,j}$, where $j \geq m + d$, may not be scheduled in $\mathcal{S}$ until the completion of $J_{k,d}$ (which happens at time $r + R$), so $\mathcal{A}(\mathcal{S}, 0, r + R, J_{k,j}) = 0$. Using this information, we have

$$\begin{aligned}
&\sum_{j=d+m}^{\infty} \mathrm{lag}(J_{k,j}, r + R) \\
&= \{\text{by Def. 2.7}\} \\
&\sum_{j=d+m}^{\infty} \mathcal{A}(\mathcal{I}, 0, r + R, J_{k,j}) - \sum_{j=d+m}^{\infty} \mathcal{A}(\mathcal{S}, 0, r + R, J_{k,j}) \\
&= \{\text{because } \mathcal{A}(\mathcal{S}, 0, r + R, J_{k,j}) = 0\} \\
&\sum_{j=d+m}^{\infty} \mathcal{A}(\mathcal{I}, 0, r + R, J_{k,j}).
\end{aligned} \tag{11}$$

By A2 and A3, 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,

$$\sum_{j=d}^{\infty} \mathcal{A}(\mathcal{I}, r, r + R, J_{k,j}) = u_k R. \tag{12}$$

From the three cases above,

$$\begin{aligned}
\mathrm{Lag}(\tau_k, r + R) &= \sum_{j=1}^{\infty} \mathrm{lag}(J_{k,j}, r + R) \\
&= \sum_{j=1}^{d-1} \mathrm{lag}(J_{k,j}, r + R) + \sum_{j=d}^{d+m-1} \mathrm{lag}(J_{k,j}, r + R) \\
&\quad + \sum_{j=d+m}^{\infty} \mathrm{lag}(J_{k,j}, r + R) \\
&= \{\text{by (9), (10), and (11)}\} \\
&\quad \sum_{j=d}^{\infty} \mathcal{A}(\mathcal{I}, 0, r + R, J_{k,j}) - W \\
&= \{\text{by (12)}\} \\
&\quad u_k R - W. \ \square
\end{aligned}$$

Knowledge of W also allows us to compute a more accurate bound on system LAG. To prove the next lemma, we exploit the following property of G-FP scheduling (which holds for any work-conserving scheduler): if $J_{k,d}$ is ready and not scheduled at time

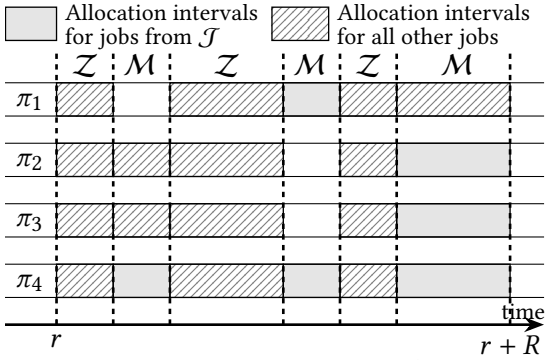

Figure 3: Example partitioning of $[r, r+R)$ into intervals.

t, then there exists some time instant t_1 such that $t_1 \leq t$, and all cores are busy from t_1 until $J_{k,d}$ is scheduled in $\mathcal{S}$ after t.

Lemma 4.5. $\mathsf{LAG}(r+R) \leq (\lceil U_k \rceil - 1)C_{max} + mC_k + (U_k - m)R - W$.

Proof. To prove this lemma, we split the interval $[r, r+R)$ into a set of maximal continuous intervals such that a 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.

Due to the nature of G-FP scheduling, 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 $0 \leq e \leq h-1$.

Definition. 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 two intervals from $\mathcal{Z}$ and $\mathcal{M}$ do not overlap, while the union of all intervals from $\mathcal{Z} \cup \mathcal{M}$ is $[r, r+R)$. An example of such intervals is given in Fig. 3.

During any $[t_1, t_2) \in \mathcal{Z}$ all cores are busy. Thus,

$$\sum_{i,j} \mathcal{A}(\mathcal{S}, t_1, t_2, J_{i,j}) = m(t_2 - t_1). \tag{13}$$

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

For any $[t_1, t_2) \in \mathcal{M}$ and any $t \in [t_1, t_2)$, at least one job from $\mathcal{J}$ is scheduled. Since 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,

$$\begin{aligned} &||\mathcal{M}|| = C_k, \text{and} \\ &||\mathcal{Z}|| = |[r, r+R)| - ||\mathcal{M}|| = R - C_k. \end{aligned} \tag{14}$$

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

$$\sum_{I_e \in \mathcal{M}} \sum_{j} \mathcal{A}(\mathcal{S}, t_e, t_{e+1}, J_{k,j}) \geq W. \tag{15}$$

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

$$\begin{aligned}
&\mathsf{LAG}(t_h) - \mathsf{LAG}(t_0) \\
&= \sum_{e=0}^{h-1} (\mathsf{LAG}(t_{e+1}) - \mathsf{LAG}(t_e)) \\
&= \{\text{by the definitions of } \mathcal{Z} \text{ and } \mathcal{M}\} \\
&\quad \sum_{I_e \in \mathcal{Z}} (\mathsf{LAG}(t_{e+1}) - \mathsf{LAG}(t_e)) \\
&\qquad + \sum_{I_e \in \mathcal{M}} (\mathsf{LAG}(t_{e+1}) - \mathsf{LAG}(t_e)) \\
&= \{\text{by Def. 2.7}\} \\
&\quad \sum_{i,j,I_e \in \mathcal{Z}} \left(\mathcal{A}(\mathcal{I}, t_e, t_{e+1}, J_{i,j}) - \mathcal{A}(\mathcal{S}, t_e, t_{e+1}, J_{i,j})\right) \\
&\qquad + \sum_{i,j,I_e \in \mathcal{M}} \left(\mathcal{A}(\mathcal{I}, t_e, t_{e+1}, J_{i,j}) - \mathcal{A}(\mathcal{S}, t_e, t_{e+1}, J_{i,j})\right) \\
&= \{\text{by rearranging and (13)}\} \\
&\quad \sum_{i,j,I_e \in \mathcal{Z} \cup \mathcal{M}} \mathcal{A}(\mathcal{I}, t_e, t_{e+1}, J_{i,j}) \\
&\qquad - \sum_{i,j,I_e \in \mathcal{Z}} m(t_{e+1} - t_e) - \sum_{i,j,I_e \in \mathcal{M}} \mathcal{A}(\mathcal{S}, t_e, t_{e+1}, J_{i,j}) \\
&\leq \{\text{by (15) and the definitions of } \mathcal{Z} \text{ and } \mathcal{M}\} \\
&\quad \sum_{i,j,I_e \in \mathcal{Z} \cup \mathcal{M}} \mathcal{A}(\mathcal{I}, t_e, t_{e+1}, J_{i,j}) - m||\mathcal{Z}|| - W \\
&\leq \{\text{by Lemma 3.7}\} \\
&\quad ||\mathcal{Z} \cup \mathcal{M}||U_k - m||\mathcal{Z}|| - W \\
&= \{\text{by (14) and } ||\mathcal{Z} \cup \mathcal{M}|| = |[r, r+R)|\} \\
&\quad RU_k - m(R - C_k) - W \\
&= mC_k + (U_k - m)R - W.
\end{aligned} \tag{16}$$

By rearranging (16) with $t_h = r + R$, and $t_0 = r$, we obtain a bound for $\mathsf{LAG}(r+R)$:

$$\begin{aligned}
&\mathsf{LAG}(r+R) \\
&\leq mC_k + (U_k - m)R - W + \mathsf{LAG}(r) \\
&\leq \{\text{by Lemma 3.9}\} \\
&\quad (\lceil U_k \rceil - 1)C_{max} + mC_k + (U_k - m)R - W. \quad \square
\end{aligned}$$

We now are able to use Lemmas 4.4 and 4.5 to bound the response time of $J_{k,d}$.

Theorem 4.6. *For any job of the npc-sporadic task τ_k, its response time is bounded by*

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

Proof. Recall that all tasks are indexed by priority and, by A1 (see Sec. 2), task τ_k has the lowest priority (and, thus, the highest task index). Consider the following:

$$
\begin{aligned}
u_k R - W &= \{\text{by Lemma 4.4}\} \\
&\quad \mathrm{Lag}(\tau_k, r+R) \\
&= \mathrm{LAG}(r+R) - \sum_{i=1}^{k-1} \mathrm{Lag}(\tau_i, r+R) \\
&\le \{\text{by Lemma 3.5}\} \\
&\quad \mathrm{LAG}(r+R) + \sum_{i=1}^{k-1} \max(0, (1-u_i)C_i) \\
&\le \{\text{by Lemma 4.5}\} \\
&\quad (\lceil U_k \rceil - 1)C_{max} + mC_k + (U_k - m)R - W \\
&\quad + \sum_{i=1}^{k-1} \max(0, (1-u_i)C_i). \qquad (17)
\end{aligned}
$$

Canceling W from the both sides of (17), we get

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

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

Corollary 4.7. *For the npc-sporadic task τ_k, its tardiness 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 4.6 does not depend on j, i.e., it applies to any job of τ_k. □

Discussion. Although the npc-sporadic task model was assumed in verifying the various properties given in Sec. 3, these properties were only applied to task τ_k. Thus, the obtained response-time bound for τ_k actually only requires that τ_k has no precedence constraints. Tasks of higher (or lower) priority could be ordinary sporadic tasks. Therefore, if some tasks from the overall task system are npc-sporadic, then such tasks have bounded response times (and hence tardiness), even if other tasks are sporadic (and hence may have unbounded response times).

Our response-time-bound proof can be adapted to apply to any work-conserving global scheduler (e.g., non-preemptive G-FP, G-EDF, etc.), as long as ready jobs of the same task are prioritized against each other on a FIFO basis. In particular, by viewing the task of interest τ_k to have the lowest priority among all n tasks, we can compute a conservative response-time bound for it that would upper bound actual response times under any work-conserving global scheduler. Such a response-time bound would even be applicable if non-preemptive execution is allowed: a lower-priority task that could cause non-preemptive blocking can induce no less interference on τ_k when τ_k is demoted to have the lowest priority. Of course, such a proof strategy is oblivious to scheduler-specific information that could be important for establishing reasonably tight response-time bounds. Still, this approach allows us to conclude that any (preemptive or non-preemptive) work-conserving global scheduler is SRT-optimal under the npc-sporadic task model.

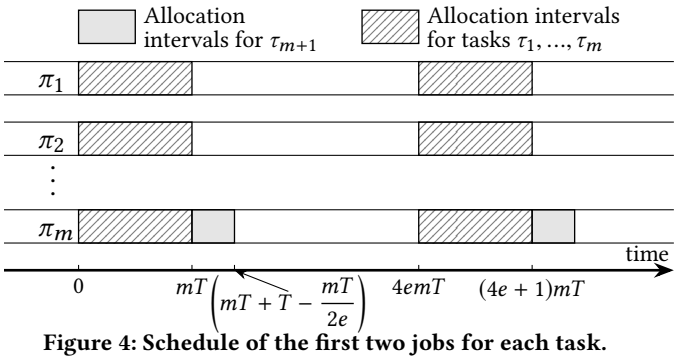

Figure 4: Schedule of the first two jobs for each task.

5 ASYMPTOTIC TIGHTNESS

In this section we show that bound from Theorem 4.6 is asymptotically tight.

Theorem 5.1. *For every $m \ge 2$ 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 4.6.*

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 cores 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. 4.

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

$$
\lceil U_{m+1} \rceil = 1, \qquad (18)
$$

$$
m - U_m = m\left(1 - \frac{1}{4e}\right), \qquad (19)
$$

$$
C_{max} = mT, \qquad (20)
$$

$$
\sum_{i=1}^{m} \max(0, (1-u_i)C_i) = \left(1 - \frac{1}{4e}\right) m^2 T. \qquad (21)
$$

With all these computed values, the bound from Theorem 4.6 is:

$$\frac{1}{m - U_m}\left(([U_{m+1}] - 1)C_{max} + mC_{m+1} + \sum_{i=1}^{m} \max(0, (1 - u_i)C_i)\right)$$

{by (18), (19), (20), and (21)}

$$= \frac{1}{m\left(1 - \frac{1}{4e}\right)}\left(0 + m\left(1 - \frac{m}{2e}\right)T + \left(1 - \frac{1}{4e}\right)m^2T\right)$$

$$= \frac{T}{\left(1 - \frac{1}{4e}\right)}\left(1 - \frac{m}{2e} + m - \frac{m}{4e}\right)$$

$$= \frac{4e(m+1) - 3m}{4e - 1}T$$

$$= \frac{(4e-1)(m+1) - 2m + 1}{4e - 1}T$$

$$= (m+1)T + \frac{1 - 2m}{4e - 1}T.$$

Then 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 4.6 is asymptotically tight. □

6 EXPERIMENTS

Notice that the numerator in the response-time bound given in Theorem 4.6 depends on the core count m and also on k, which determines the number of higher-priority tasks that exist. A reduced bound can therefore 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 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 a valid assignment of tasks to clusters.

To generate task systems, we selected task periods from the range $[10ms, 100ms]$. We also considered various task utilization ranges. Due to space limitations, we consider only two utilization categories here: *light tasks* with $u_i \in (0.0, 0.3)$, and *medium tasks* with $u_i \in [0.3, 0.7)$. 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 4.7, and observed relative tardiness, by examining schedules in which jobs were released periodically. The results we obtained for light tasks are plotted in Fig. 5, and those for medium tasks in Fig. 6. Each plot in both 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 impact on schedulability for light task systems. For medium 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.

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.

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

7 CONCLUSION

In this paper, we considered the scheduling of npc-sporadic task systems under G-FP on a multicore 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.

Our proof strategy can be applied to show that any (preemptive or non-preemptive) work-conserving scheduler is SRT-optimal. 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 scheduled is used [13]. 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.

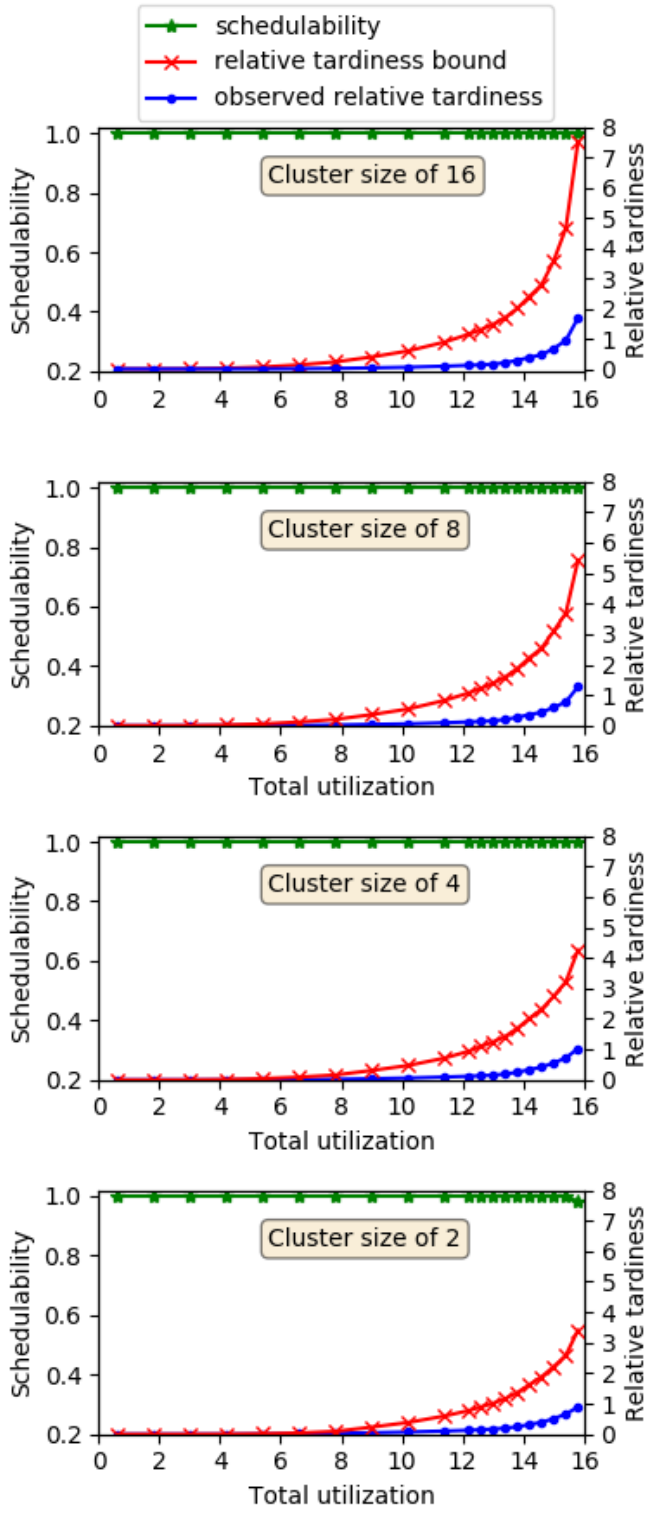

Figure 5: Experimental results for light tasks.

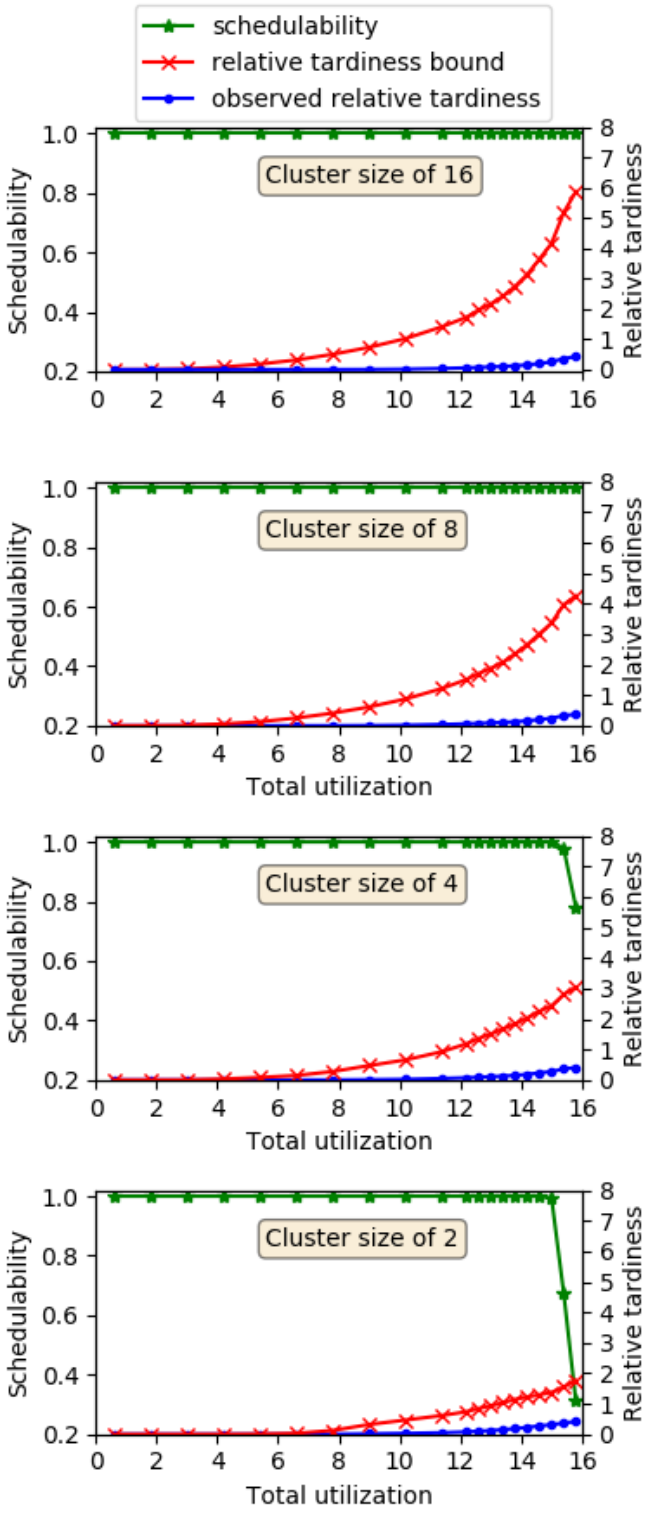

Figure 6: Experimental results for medium tasks.

REFERENCES

[1] T. Baker. Comparison of empirical success rates of global vs. partitioned fixed-priority and EDF scheduling for hard real time. Technical Report TR-050601, Department of Computer Science, Florida State University, 2005.

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

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

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

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

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

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

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

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

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

[11] H. Leontyev and J. Anderson. 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*, pages 103–110, 2007.

[12] K. Yang and J. Anderson. 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*, pages 30–39, 2014.

[13] K. Yang, M. Yang, and J. Anderson. 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*, pages 349–358, 2016.

[14] M. Yang, T. Amert, K. Yang, N. Otterness, J. Anderson, F. D. Smith, and S. Wang. Making OpenVX really 'real time'. In *Proceedings of the 39th IEEE Real-Time Systems Symposium, to appear*, 2018.