GEDF Tardiness: Open Problems Involving Uniform Multiprocessors and Affinity Masks Resolved

Stephen Tang
Department of Computer Science, University of North Carolina at Chapel Hill, NC, USA
sytang@cs.unc.edu

Sergey Voronov
Department of Computer Science, University of North Carolina at Chapel Hill, NC, USA
rdkl@cs.unc.edu

James H. Anderson
Department of Computer Science, University of North Carolina at Chapel Hill, NC, USA
anderson@cs.unc.edu

Abstract

Prior work has shown that the global earliest-deadline-first (GEDF) scheduler is *soft real-time (SRT)-optimal* for sporadic task systems in a variety of contexts, meaning that bounded deadline tardiness can be guaranteed under it for any task system that does not cause platform overutilization. However, one particularly compelling context has remained elusive: multiprocessor platforms in which tasks have *affinity masks* that determine the processors where they may execute. Actual GEDF implementations, such as the `SCHED_DEADLINE` class in Linux, have dealt with this unresolved question by foregoing SRT guarantees once affinity masks are set. This unresolved question, as it pertains to `SCHED_DEADLINE`, was included by Peter Zijlstra in a list of important open problems affecting Linux in his keynote talk at ECRTS 2017. In this paper, this question is resolved along with another open problem that at first blush seems unrelated but actually is. Specifically, both problems are closed by establishing two results. First, a proof strategy used previously to establish GEDF tardiness bounds that are exponential in size on heterogeneous uniform multiprocessors is generalized to show that polynomial bounds exist on a wider class of platforms. Second, both uniform multiprocessors and identical multiprocessors with affinities are shown to be within this class. These results yield the first polynomial GEDF tardiness bounds for the uniform case and the first such bounds of any kind for the identical-with-affinities case.

2012 ACM Subject Classification Software and its engineering → Real-time schedulability

Keywords and phrases scheduling theory, multicore, processor affinity masks, GEDF, uniform multiprocessors

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2019.6

Related Version A full version of the paper is available at `http://www.cs.unc.edu/~anderson/papers.html`.

Funding Work supported by NSF grants CNS 1409175, CNS 1563845, CNS 1717589, and CPS 1837337, ARO grant W911NF-17-1-0294, and funding from General Motors.

1 Introduction

The global earliest-deadline-first (GEDF) scheduler has received considerable prior attention. One attractive property of GEDF is that its use ensures guaranteed bounded deadline tardiness on certain multiprocessor platform types for any sporadic task system that does not cause platform overutilization [5, 9]. In this sense, GEDF is considered an *optimal soft real-time (SRT)* scheduler. This SRT-optimality is significant enough to warrant mention in

the documentation of `SCHED_DEADLINE` [2], Linux's implementation of GEDF. A practical use by `SCHED_DEADLINE` of this optimality is the ability to perform online admission control.

A major caveat of GEDF's SRT-optimality is that it was originally proven only for processors with identical speeds [5]. Processor models that break this assumption, called *heterogeneous multiprocessors*, tend to fundamentally break the proof techniques applied to the identical model. Only one prior work [9] has succeeded in extending the SRT-optimality of GEDF to a multiprocessor model with heterogeneity, namely the *uniform model*, which allows speed differences among processors. This work required years of thought and several new proof techniques but only yielded tardiness bounds that are exponential in size.

Affinity masks. Heterogeneity is also introduced by the usage of *affinity masks*. A task's affinity mask (typically a bit vector) indicates the processors upon which it may execute. Affinity masks are useful for preventing excessive task migrations and can be used to take better advantage of cache hierarchies. Also, global, clustered, and partitioned scheduling can all be expressed using affinity masks.

Affinity masks introduce heterogeneity by removing the symmetry among processors. An important property used to prove the SRT-optimality of GEDF without affinities is that the existence of an idle processor is a sufficient condition for a pending task to begin executing. With this property, showing that a task of interest makes progress merely requires showing that *some* processor becomes idle in a bounded amount of time. With affinity masks, this proof strategy does not work. In particular, this strategy's use would require showing that a processor *allowed by the considered task's affinity mask* becomes idle, and doing so over the entire space of arbitrary affinity masks leads to a case explosion.

This difficulty has left unresolved the question of whether GEDF retains its SRT-optimality with the introduction of affinity masks. This unresolved question has rendered systems that support both GEDF and affinity masks incomplete. For example, in the case of `SCHED_DEADLINE`, either admission control is disabled or affinity masks are forbidden altogether. This specific gap between theory and implementation was mentioned by Peter Zijlstra [10] in a keynote talk at ECRTS 2017 and by Luca Abeni [1] at RTSS 2017.

Contributions. In this paper, we close the open problem of whether GEDF retains its SRT-optimality on identical multiprocessors with affinities by showing that it does, provided the GEDF implementation maintains a certain task-migration property. We also establish the first ever polynomial tardiness bounds for GEDF on uniform multiprocessors. While these two platform models (uniform and identical with affinities) may seem unrelated to each other, we shall see that they are.

These results hinge on three proof innovations. First, we introduce a layer of abstraction between the processor models we consider and the tardiness analysis. This abstraction layer is a property we call "HP-LAG," which we prove is satisfied by GEDF on the models we consider for all sporadic task systems that do not cause platform overutilization.[1] This strategy allows us to reason about tardiness without regard for specifics concerning the underlying platform, and as a result, we are able to avoid the aforementioned case explosion.

Second, for the statement made in the prior paragraph concerning GEDF to be valid, the definition of GEDF itself must be tailored for the specific platform type under consideration. Prior work [9] has shown how to do so for the case of a uniform platform, and we show here

[1] The concept of "overutilization" is more nuanced on uniform platforms and platforms with affinities than on identical platforms with no affinities.

how to do so for an identical platform with affinities by specifying rules that GEDF must adhere to in this case. These rules specify when and how tasks must be migrated as scheduling decisions are made. Now that these rules are known, actual implementations of GEDF on systems that support affinity masks should be designed to uphold them. Unfortunately, as we discuss in App. C, available online [7], the current `SCHED_DEADLINE` implementation does not do so, so a refinement of it is needed if affinity masks are to be used alongside admission control. In App. A, also available online [7], we present an algorithm that implements our GEDF scheduling rules for affinity masks with lower time complexity than `SCHED_DEADLINE`, given some preprocessing.

The last innovation is our improved tardiness analysis with less pessimism than prior work, particularly with respect to uniform multiprocessor platforms. This required maintaining an exponential number of invariants relative to prior work.

Organization. In the rest of this paper, we cover needed background and define the important concept of Lag, which is widely used in tardiness analysis (Sec. 2), list some general Lag properties that are predicated upon tasks being periodic and executing for their worst-case requirements (Sec. 3), define HP-LAG and present our tardiness analysis for HP-LAG-compliant schedulers using said properties (Sec. 4), show that GEDF (if implemented appropriately) is HP-LAG-compliant on both uniform multiprocessors (Sec. 5) and identical multiprocessors with affinities (Sec. 6), show that our results extend to sporadic tasks that may execute for less than their worst-case requirements (Sec. 7), explain why these results cannot be easily extended to more general models (Sec. 8), and conclude (Sec. 9).

2 Background

We consider the problem of scheduling n implicit-deadline sporadic tasks $\tau = \{\tau_1, \ldots, \tau_n\}$ on a multiprocessor $\pi = \{\pi_1, \ldots, \pi_m\}$ with m processors. We consider time to be continuous. Each task τ_i releases a sequence of jobs with a minimum separation of T_i time units between releases; T_i is called τ_i's *period*. The j^{th} released job of τ_i is denoted $J_{i,j}$, and its release time is denoted $r_{i,j}$. When this separation between the jobs of each task τ_i is exactly T_i, the system is called *periodic*. With the assumption of implicit deadlines, the *deadline* of a job of τ_i is exactly T_i time units after its release; the deadline of job $J_{i,j}$ is denoted $d_{i,j} = r_{i,j} + T_i$. The amount of work that is needed to complete a job of τ_i is bounded by τ_i's *worst-case execution requirement (WCER)* C_i, the largest of which among the tasks in τ we denote as C_{max}. Note that C_i is typically called the worst-case execution *time* in the literature. This is because much of the literature assumes that processors complete one unit of execution in one unit of time. This assumption does not hold for some of the hardware platform models we consider in this work (see Sec. 2.1). τ_i's *utilization* is defined as $u_i = C_i/T_i$. We let u_{min} be the smallest utilization among the tasks in τ. For any $\tau' \subseteq \tau$, we let $U_{\tau'}$ denote $\sum_{\tau_i \in \tau'} u_i$. A job is *pending* at time t if it has been released but has not completed. Likewise, a task is *pending* at t if any of its jobs are pending at t. A job is *ready* at t if it is the earliest released pending job from its task.

▶ **Definition 1.** *If task τ_i is pending at time t, then we define its release time at t, denoted $r_i(t)$, and its deadline at t, denoted $d_i(t)$, as the release time and deadline, respectively, of its ready job at time t. If τ_i is not pending at t, then we define $d_i(t) = \infty$.*

If a job has a deadline at time t_d and completes at time t_c, then its *tardiness* is defined as $\max(0, t_c - t_d)$. The tardiness of a task is the supremum of the tardiness of any of its jobs. If this value is finite, then we say that the task has *bounded tardiness*.

In the real-time literature, both hard real-time (HRT) and soft real-time (SRT) systems are considered. While SRT can be defined in different ways, we adopt the following definitions. A task set τ is HRT-schedulable (resp., SRT-schedulable) under a given scheduling algorithm if each task in τ has 0 (resp., bounded) tardiness in any schedule for τ generated by that algorithm. A task set τ is HRT-feasible (resp., SRT-feasible) if it is HRT-schedulable (resp., SRT-schedulable) under some scheduling algorithm. A scheduler is HRT-optimal (resp., SRT-optimal) if it can schedule any HRT-feasible (resp., SRT-feasible) system.

2.1 Multiprocessor Platform Models

There exist several multiprocessor platform models in the literature that differ by how execution speeds are allowed to vary. We formalize execution speeds as follows. Hereafter, we let $A(\mathcal{S}, \tau_i, t, t')$ denote the cumulative execution allocated to jobs of task τ_i in the schedule $\mathcal{S}$ in the time interval (t, t').

▶ **Definition 2.** *A time interval (t, t') is a continuous scheduling interval if the assignment of tasks to processors at t is maintained throughout (t, t').*

▶ **Definition 3.** *Suppose task τ_i executes on processor π_j over the continuous scheduling interval (t, t') in schedule $\mathcal{S}$. If the speed of τ_i on π_j is s, then $A(\mathcal{S}, \tau_i, t, t') = s(t' - t)$.*

In order of increasing generality, the platform models of relevance to us are as follows.

1. **Identical.** All tasks execute with speed 1.0 on all processors.
2. **Uniform.** Speeds may vary by processor, but not by task. Any task on processor π_i executes with speed s_i, which may differ from 1.0.
3. **Unrelated.** Speeds may vary by processor and by task. Task τ_i executes on processor π_j with speed $s_{i,j}$.

In addition to these models, the migration scheme can be one of two types.

1. **Global.** A task can be scheduled on any processor.
2. **Affinity.** A task can only be scheduled on a specific set of processors as defined by its *affinity mask*. We let $\alpha_i \subseteq \pi$ denote the set of processors allowed by τ_i's affinity mask.

Note that global, clustered, and partitioned scheduling can all be defined using affinity masks. We separate global scheduling as a separate case because some later proofs focus on it exclusively. From this point on, global scheduling is assumed unless the -Aff suffix is appended. For example, Identical and Identical-Aff refer to an identical multiprocessor under global and affinity scheduling, respectively. Note that Unrelated generalizes not only Identical and Uniform, but also affinity scheduling, as $\pi_j \notin \alpha_i$ can be represented by letting $s_{i,j} = 0$. Hence, all combinations of platform and migration scheme are generalized by Unrelated.

GEDF has been proven to be SRT-optimal under Identical, but no prior work has generalized this to Identical-Aff. GEDF's SRT-optimality has been generalized to Uniform [9], but with exponential tardiness bounds. In this work, we establish the SRT-optimality of GEDF with polynomial tardiness under both Uniform and Identical-Aff.

The SRT-optimality of GEDF under Uniform-Aff and Unrelated are difficult or impossible to prove using the techniques of this work. We describe why in Sec. 8.

We will later show that the typical GEDF scheduling rules under Identical may be insufficient to achieve SRT-optimality under the more general models. As such, we will later define extended GEDF scheduling rules for Identical-Aff and Uniform, respectively (our rules for Uniform are actually from [9]).

The standard GEDF scheduler under Identical is:

IG-GEDF:	At every time instant, if more than m tasks are pending, then the m pending tasks with the earliest deadlines are scheduled; otherwise, all pending tasks are scheduled.

The prefix "IG" denotes that this rule applies under Identical with global scheduling. We will keep this notation when extending GEDF, denoting the extended GEDF rules for Identical-Aff and Uniform as IA-GEDF and UG-GEDF, respectively. Under all GEDF variants we consider, we assume deadline ties are broken in some arbitrary but consistent way (e.g., by task index). For any time instant t, for tasks τ_i and τ_j, we let $d_i(t) \prec d_j(t)$ denote that either $d_i(t) < d_j(t)$ holds or $d_i(t) = d_j(t)$ holds with the tie broken in τ_i's favor. As we shall see, IA-GEDF and UG-GEDF reduce to IG-GEDF when the underlying processor model is Identical (note that both Uniform and Identical-Aff generalize Identical).

2.2 Lag

As in [5], our analysis is based on the concept of Lag. Lag compares the execution of a task in a "real" schedule $\mathcal{R}$ to its allocation in an "ideal" schedule $\mathcal{I}$.

▶ **Definition 4.** *A non-fluid schedule is a schedule such that at any time instant t, there exists some $\delta > 0$ such that $(t, t + \delta)$ is a continuous scheduling interval.*

Implementable schedulers are non-fluid.

▶ **Definition 5.** *We let $\mathcal{R}$ denote a non-fluid schedule produced under a considered scheduling algorithm and multiprocessor platform.*

▶ **Definition 6.** *We let $\mathcal{I}$ denote a schedule that executes task τ_i on processor π_i of speed u_i.*

Notice that $\mathcal{I}$ is defined with respect to a Uniform multiprocessor consisting of n processors $\pi_1, ..., \pi_n$ with speeds $u_1, ..., u_n$, respectively, and not the actual platform π.

Under the implicit-deadline sporadic task model, every job executes in $\mathcal{I}$ from its release until its completion without interference from other jobs or tasks (different tasks run on different processors). If a job's execution requirement is smaller than the WCER of its task, then the job completes in the ideal schedule before its deadline; otherwise, it completes exactly at its deadline. Thus, in $\mathcal{I}$, at most one job from every task is ever scheduled.

We are now ready to formally define Lag.

▶ **Definition 7.** *For a single task τ_i, $\mathsf{Lag}_i(t) = A(\mathcal{I}, \tau_i, 0, t) - A(\mathcal{R}, \tau_i, 0, t)$. For the subset $\tau' \subseteq \tau$, $\mathsf{LAG}(\tau',\ t) = \sum_{\tau_i \in \tau'} \mathsf{Lag}_i(t)$.*

3 General Lag Properties

In [9], Yang and Anderson showed how to generalize IG-GEDF to obtain a variant, which we denote as UG-GEDF, that is SRT-optimal under Uniform. Yang and Anderson's proof relied on several properties of Lag that we make use of in this work. We repeat these properties and proofs verbatim from [9], with minor wording changes. However, unlike [9], where these properties were considered in the context of Uniform, we consider them in the context of Unrelated. Because Unrelated generalizes all the models in Sec. 2.1, these Lag properties apply to all the models considered in this work.

Lemmas 10–12 rely on the following assumptions, which we henceforth assume until stated otherwise.

Every task is periodic. (P)

Every job $J_{i,j}$ has an execution requirement equal to C_i (the worst case for τ_i). (W)

So as to leave no doubt that the properties considered in this section hold under Unrelated, we begin by listing all of the model-related concepts the proofs below will use:

- the task-system parameters C_i, T_i, u_i, $r_i(t)$, and $d_i(t)$;
- Lag and LAG, as defined in Def. 7;
- the fact that $\mathcal{I}$ continuously executes task τ_i with speed u_i, which follows from (P) and (W), hence the need for these assumptions

[9] showed that removing (P) and (W) cannot cause greater tardiness in UG-GEDF under Uniform. We will show the same for IA-GEDF under Identical-Aff later in Sec. 7.

▶ **Lemma 8** (Property 1 of [9]). $\forall \tau' \subseteq \tau :$ $\mathsf{LAG}(\tau',\ t)$ *is a continuous function of* t.

Proof. By definition, $A(\mathcal{S}, \tau_i, t, t')$ is a continuous function in t and t'. By Def. 7, $\mathsf{LAG}(\tau',\ t) = \sum_{\tau_i \in \tau'} \mathsf{Lag}_{\tau_i}(t) = \sum_{\tau_i \in \tau'} (A(\mathcal{I}, \tau_i, 0, t) - A(\mathcal{R}, \tau_i, 0, t))$. Because $\mathsf{LAG}(\tau',\ t)$ is a finite sum of continuous functions in t, it is continuous in t. ◀

▶ **Lemma 9** (Lemma 1 of [9]). $\mathsf{Lag}_i(t) > 0 \Rightarrow \tau_i$ *has a pending job at* t.

Proof. We prove the lemma by contradiction. Suppose that $\mathsf{Lag}_i(t) > 0$ holds but τ_i is not pending at t in $\mathcal{R}$. Then, all jobs of τ_i released before or at t have completed by t. Let W denote the total execution requirement of such jobs such that $W = A(\mathcal{R}, \tau_i, 0, t)$. In $\mathcal{I}$, only released jobs can be scheduled and will not execute for more than their execution requirement. Thus, $A(\mathcal{I}, \tau_i, 0, t) \leq W$ holds as well. Therefore, by Def. 7, we have $\mathsf{Lag}_i(t) = A(\mathcal{I}, \tau_i, 0, t) - A(\mathcal{R}, \tau_i, 0, t) \leq 0$, a contradiction. ◀

▶ **Lemma 10** (Lemma 2 of [9]). *If task* τ_i *is pending at time* t *in* $\mathcal{R}$*, then*

$$t - \frac{\mathsf{Lag}_i(t)}{u_i} < d_i(t) \leq t - \frac{\mathsf{Lag}_i(t)}{u_i} + T_i. \tag{1}$$

Proof. Let $e_i(t)$ denote the remaining execution requirement for the ready job $J_{i,j}$ of τ_i at time t. Because this job is ready, it must not be complete, hence

$$0 < e_i(t) \leq C_i. \tag{2}$$

All jobs of τ_i prior to $J_{i,j}$ must have been completed by time t. Let E denote the total execution requirement of these jobs. Then,

$$A(\mathcal{R}, \tau_i, 0, t) = E + C_i - e_i(t). \tag{3}$$

In $\mathcal{I}$, all prior jobs of τ_i have completed by $r_i(t)$. Within $(r_i(t), t)$, $\mathcal{I}$ continuously executes τ_i on a processor with speed u_i. Thus,

$$A(\mathcal{I}, \tau_i, 0, t) = E + (t - r_i(t))u_i. \tag{4}$$

Given these facts, an expression for $\mathsf{Lag}_i(t)$ can be derived as follows.

$\mathsf{Lag}_i(t) = \{$by Def. 7$\}$

$$
\begin{aligned}
& A(\mathcal{I}, \tau_i, 0, t) - A(\mathcal{R}, \tau_i, 0, t) \\
= & \{\text{by (3) and (4)}\} \\
& (t - r_i(t))u_i - (C_i - e_i(t)) \\
= & \{\text{because } d_i(t) = r_i(t) + T_i\} \\
& (t - d_i(t) + T_i)u_i - (C_i - e_i(t)) \\
= & \{\text{because } u_i T_i = C_i\} \\
& (t - d_i(t))u_i + e_i(t)
\end{aligned}
$$

By (2) and the above expression, we have

$$(t - d_i(t))u_i < \mathsf{Lag}_i(t) \leq (t - d_i(t))u_i + C_i, \tag{5}$$

which (using $T_i = C_i/u_i$) can be rearranged to obtain (1). ◀

▶ **Corollary 11** (Corollary 1 of [9]). *If for some $L > 0$ we have $\forall t, \mathsf{Lag}_i(t) \leq L$, then the tardiness of task τ_i does not exceed L/u_i.*

Proof. We prove the corollary by contradiction. Suppose that

$$\mathsf{Lag}_i(t) \leq L \tag{6}$$

holds but τ_i has tardiness exceeding L/u_i. Then, there exists a job $J_{i,j}$ that is pending at time $t \geq d_{i,j}$ where

$$t - d_{i,j} > L/u_i. \tag{7}$$

Because $J_{i,j}$ is pending at t, τ_i's ready job cannot have been released later than $J_{i,j}$. Thus, $d_i(t) \leq d_{i,j}$. Therefore,

$$
\begin{aligned}
t - d_i(t) & \geq t - d_{i,j} \\
& > \{\text{by (7)}\} \\
& \quad L/u_i \\
& \geq \{\text{by (6)}\} \\
& \quad \mathsf{Lag}_i(t)/u_i.
\end{aligned}
$$

Rearrangement yields $t - \mathsf{Lag}_i(t)/u_i > d_i(t)$, which contradicts Lemma 10. ◀

▶ **Lemma 12** (Lemma 4 of [9]). *If a task τ_i has a pending job at t and for a task τ_j we have*

$$\frac{1}{u_j}\mathsf{Lag}_j(t) + T_{max} \leq \frac{1}{u_i}\mathsf{Lag}_i(t), \tag{8}$$

then $d_i(t) < d_j(t)$.

Proof. Assume τ_j is pending at t, as otherwise $d_j(t) = \infty$, and we trivially have $d_i(t) < d_j(t)$.

$$
\begin{aligned}
d_i(t) & \leq \{\text{by Lemma 10}\} \\
& \quad t - \frac{\mathsf{Lag}_i(t)}{u_i} + T_i \\
& \leq \{\text{by (8)}\}
\end{aligned}
$$

$$
\begin{aligned}
& t - \frac{\mathsf{Lag}_j(t)}{u_j} - T_{max} + T_i \\
\leq\ & \{\text{because } -T_{max} + T_i \leq 0\} \\
& t - \frac{\mathsf{Lag}_j(t)}{u_j} \\
<\ & \{\text{by Lemma 10}\} \\
& d_j(t)
\end{aligned}
$$

◀

As a reminder, we are considering the properties in this section in the context of Unrelated. This means these properties apply to all of our considered models.

4 Tardiness Bounds for HP-LAG-Compliant Schedulers

In this paper, we consider two Identical generalizations: Uniform and Identical-Aff. In order to prove GEDF's SRT-optimality under these different processor models, we provide an abstraction layer called HP-LAG. Informally, HP-LAG states that the LAG of any subset of tasks τ' with the earliest deadlines is temporarily non-increasing.

HP-LAG:	For any time instant t, if $\tau' \subseteq \tau$ is a set of pending tasks such that $\forall \tau_h \in \tau'$ and $\forall \tau_\ell \in \tau/\tau'$ we have $d_h(t) < d_\ell(t)$, then $\exists \delta > 0$ such that $\forall t' \in (t, t+\delta) :$ $\mathsf{LAG}(\tau',\ t) \geq \mathsf{LAG}(\tau',\ t')$.

▶ **Definition 13.** *We say that a scheduler is HP-LAG-compliant under a given platform if HP-LAG holds for any feasible task system τ under said platform model.*

In this section, we show that every HP-LAG-compliant scheduler ensures bounded tardiness under its considered platform model. We do this by extending the approach of [9] by maintaining as invariants bounds on the LAG of *every* task subset; in [9], only a *linear* (with respect to m) number of invariants is instead maintained. The LAG bound we define for any subset of tasks $\tau' \subseteq \tau$ is

$$\beta(\tau') = \frac{T_{max}}{2u_{min}} U_{\tau'} \left(2U_\tau - U_{\tau'}\right). \tag{9}$$

We will prove by contradiction that $\forall \tau' \subseteq \tau\ \forall t :\ \mathsf{LAG}(\tau',\ t) \leq \beta(\tau')$ holds for any HP-LAG-compliant scheduler. We continue to consider all properties in the context of Unrelated. Thus, Lemmas 16, 17, 18, and 19 below hold for any scheduler that is HP-LAG-compliant under any processor model that Unrelated generalizes. We begin by defining a set of time instants that must exist if our LAG bounds are violated.

▶ **Definition 14.** *We call a time instant t invalid if $\exists \tau' \subseteq \tau$ such that $\forall \delta > 0\ \exists t' \in (t, t+\delta) :$ $\mathsf{LAG}(\tau',\ t') > \beta(\tau')$. τ' is called an attestant set of the invalid instant t.*

Note that for any invalid instant, $\emptyset$ is never an attestant set because for any time instant t we have $\mathsf{LAG}(\emptyset,\ t) = 0$ and $\beta(\emptyset) = 0$.

▶ **Definition 15.** *If at least one invalid time instant exists, then we call the first[2] such instant a boundary instant, denoted t_b. We let τ^b denote an arbitrary attestant set of t_b. We call any task from τ^b a boundary task.*

[2] It can be shown that the infimum of all invalid instants is itself an invalid instant. Hence, the first invalid instant, t_b, is well-defined.

Because t_b is the first invalid instant, we can prove specific bounds on Lag values at t_b.

▶ **Lemma 16.** *For the boundary instant t_b, the following three expressions hold.*

$$\forall \tau' \subseteq \tau : \mathsf{LAG}(\tau',\ t_b) \leq \beta(\tau') \tag{10}$$

$$\mathsf{LAG}(\tau^b,\ t_b) = \beta(\tau^b) \tag{11}$$

$$\forall \delta > 0\ \exists t' \in (t_b, t_b + \delta) : \mathsf{LAG}(\tau^b,\ t') > \beta(\tau^b) \tag{12}$$

Proof. We prove (10) by contradiction. If for some $\tau' \subseteq \tau$, $\mathsf{LAG}(\tau',\ t_b) > \beta(\tau')$, then, by Lemma 8 (continuity of $\mathsf{LAG}(\tau',\ t)$), $\exists \delta > 0\ \forall t' \in (t_b - \delta, t_b) :\ \mathsf{LAG}(\tau',\ t') > \beta(\tau')$. Thus, time instant $t_b - \delta$ is invalid with attestant set τ', which contradicts Def. 15.

By Defs. 14 and 15, $\forall \delta > 0\ \exists t' \in (t_b, t_b + \delta) :\ \mathsf{LAG}(\tau^b,\ t') > \beta(\tau^b)$. By Lemma 8 (continuity of $\mathsf{LAG}(\tau^b,\ t)$), we have $\mathsf{LAG}(\tau^b,\ t_b) \geq \beta(\tau^b)$. By (10), $\mathsf{LAG}(\tau^b,\ t_b) \leq \beta(\tau^b)$, so (11) holds.

(12) follows from Defs. 14 and 15. ◀

▶ **Lemma 17.** *For any boundary task τ_i at t_b, $\mathsf{Lag}_i(t_b) \geq \dfrac{T_{max}}{2u_{min}}(2u_i U_\tau - 2u_i U_{\tau^b} + u_i^2)$.*

Proof.

$$\begin{aligned}
\mathsf{Lag}_i(t_b) &= \{\text{by Def. 7}\} \\
&\quad \mathsf{LAG}(\tau^b,\ t_b) - \mathsf{LAG}(\tau^b/\{\tau_i\},\ t_b) \\
&\geq \{\text{by (11), } \mathsf{LAG}(\tau^b,\ t_b) = \beta(\tau^b), \text{ and by (10), } \mathsf{LAG}(\tau^b/\{\tau_i\},\ t_b) \leq \beta(\tau^b/\{\tau_i\})\} \\
&\quad \beta(\tau^b) - \beta(\tau^b/\{\tau_i\}) \\
&= \{\text{by (9)}\} \\
&\quad \frac{T_{max}}{2u_{min}}\left[U_{\tau^b}\left(2U_\tau - U_{\tau^b}\right)\right] - \frac{T_{max}}{2u_{min}}\left[U_{\tau^b/\{\tau_i\}}\left(2U_\tau - U_{\tau^b/\{\tau_i\}}\right)\right] \\
&= \{\text{by the definition of } U_{\tau^b/\{\tau_i\}}\} \\
&\quad \frac{T_{max}}{2u_{min}}\left[U_{\tau^b}\left(2U_\tau - U_{\tau^b}\right)\right] - \frac{T_{max}}{2u_{min}}\left[\left(U_{\tau^b} - u_i\right)\left(2U_\tau - U_{\tau^b} + u_i\right)\right] \\
&= \{\text{rearranging}\} \\
&\quad \frac{T_{max}}{2u_{min}}(2u_i U_\tau - 2u_i U_{\tau^b} + u_i^2)
\end{aligned}$$

◀

Note that the Lag lower bound from Lemma 17 is strictly positive, because $U_\tau \geq U_{\tau^b}$. Thus, by Lemma 9, any boundary task τ_i is pending at t_b. This proves the following lemma.

▶ **Lemma 18.** *At the boundary time instant t_b, every boundary task has a pending job.*

As shown next, similar reasoning as in Lemma 17 can be used to upper bound the Lag of non-boundary tasks. This allows us to establish a relationship between the deadlines of boundary and non-boundary tasks.

▶ **Lemma 19.** *At time instant t_b, every boundary task has an earlier deadline than any non-boundary task (i.e., from τ/τ^b).*

Proof. For any non-boundary task $\tau_j \in \tau/\tau^b$, we have the following.

$$\begin{aligned}
\mathsf{Lag}_j(t_b) &= \{\text{by Def. 7}\} \\
&\quad \mathsf{LAG}(\tau^b \cup \{\tau_j\},\ t_b) - \mathsf{LAG}(\tau^b,\ t_b)
\end{aligned}$$

$$
\begin{aligned}
&\leq \{\text{by (10), } \mathsf{LAG}(\tau^b \cup \{\tau_j\},\ t_b) \leq \beta(\tau^b \cup \{\tau_j\}), \text{ and by (11), } \mathsf{LAG}(\tau^b,\ t_b) = \beta(\tau^b)\} \\
&\quad \beta(\tau^b \cup \{\tau_j\}) - \beta(\tau^b) \\
&= \{\text{by (9)}\} \\
&\quad \frac{T_{max}}{2u_{min}} \left[U_{\tau^b \cup \{\tau_j\}} \left(2U_\tau - U_{\tau^b \cup \{\tau_j\}}\right)\right] - \frac{T_{max}}{2u_{min}} \left[U_{\tau^b} \left(2U_\tau - U_{\tau^b}\right)\right] \\
&= \{\text{by the definition of } U_{\tau^b \cup \{\tau_j\}}\} \\
&\quad \frac{T_{max}}{2u_{min}} \left[\left(U_{\tau^b} + u_j\right) \left(2U_\tau - U_{\tau^b} - u_j\right)\right] - \frac{T_{max}}{2u_{min}} \left[U_{\tau^b} \left(2U_\tau - U_{\tau^b}\right)\right] \\
&= \{\text{rearranging}\} \\
&\quad \frac{T_{max}}{2u_{min}} (2u_j U_\tau - 2u_j U_{\tau^b} - u_j^2) \qquad (13)
\end{aligned}
$$

To conclude the proof, we show that the Lag of a boundary task $\tau_i \in \tau^b$ and the Lag of a non-boundary task $\tau_j \in \tau/\tau^b$ together satisfy the requirement specified in Lemma 12.

$$
\begin{aligned}
\frac{1}{u_j}\mathsf{Lag}_j(t_b) + T_{max} &\leq \{\text{by (13)}\} \\
&\quad \frac{1}{u_j} \frac{T_{max}}{2u_{min}} (2u_j U_\tau - 2u_j U_{\tau^b} - u_j^2) + T_{max} \\
&= \{\text{factor in } 1/u_j \text{ and out } 1/u_i \text{ from } 2u_j U_\tau - 2u_j U_{\tau^b}\} \\
&\quad \frac{1}{u_i} \frac{T_{max}}{2u_{min}} (2u_i U_\tau - 2u_i U_{\tau^b}) + T_{max} \left(1 - \frac{u_j}{2u_{min}}\right) \\
&\leq \{2u_{min} - u_j = u_{min} + (u_{min} - u_j) \leq u_{min} \leq u_i\} \\
&\quad \frac{1}{u_i} \frac{T_{max}}{2u_{min}} (2u_i U_\tau - 2u_i U_{\tau^b}) + T_{max} \frac{1}{u_i} \left(\frac{u_i^2}{2u_{min}}\right) \\
&\leq \{\text{by Lemma 17}\} \\
&\quad \mathsf{Lag}_i(t_b) \qquad (14)
\end{aligned}
$$

By Lemma 18, a boundary task τ_i is pending, and by Lemma 12, its deadline is earlier than task $\tau_j \in \tau/\tau^b$ (if τ_j has no pending job, then $d_j(t) = \infty$ by Def. 1) at time t_b. ◀

▶ **Theorem 20.** *If a scheduler is HP-LAG-compliant under its considered platform, then for any feasible task system τ, the tardiness of task $\tau_i \in \tau$ is at most*

$$\frac{T_{max}}{2u_{min}} (2U_\tau - u_i). \qquad (15)$$

Proof. If there exists at least one invalid instant, we can define the boundary time instant t_b with an attestant set τ^b. By Lemma 19, tasks in τ^b have earlier deadlines than any task in τ/τ^b. Thus, by HP-LAG with $\tau' = \tau^b$,

$$\exists \delta > 0\ \forall t \in (t_b, t_b + \delta) :\ \mathsf{LAG}(\tau^b,\ t_b) \geq \mathsf{LAG}(\tau^b,\ t). \qquad (16)$$

However, by Lemma 16,

$$\forall \delta > 0\ \exists t \in (t_b, t_b + \delta) :\ \mathsf{LAG}(\tau^b,\ t) > \beta(\tau^b) = \mathsf{LAG}(\tau^b,\ t_b),$$

which contradicts (16). Because the existence of t_b leads to a contradiction, there is no first invalid instant, and hence there are no invalid time instants. Thus, by Def. 14,

$$\forall \tau' \subseteq \tau\ \forall t \geq 0\ \exists \delta > 0\ \forall t' \in (t, t + \delta) :\ \mathsf{LAG}(\tau',\ t') \leq \beta(\tau').$$

By Lemma 8, it follows that

$$\forall \tau' \subseteq \tau \; \forall t \geq 0 : \; \mathsf{LAG}(\tau', \; t) \leq \beta(\tau'). \tag{17}$$

Hence, for any task τ_i and any time instant t,

$$\begin{aligned}
\mathsf{Lag}_i(t) &= \{\text{by Def. 7}\} \\
&\quad \mathsf{LAG}(\{\tau_i\}, \; t) \\
&\leq \{\text{by (17) with } \tau' = \{\tau_i\}\} \\
&\quad \beta(\{\tau_i\}) \\
&= \{\text{by (9)}\} \\
&\quad \frac{T_{max}}{2u_{min}} u_i(2U_\tau - u_i).
\end{aligned}$$

Thus, by Corollary 11, task τ_i has maximum tardiness at most $\frac{T_{max}}{2u_{min}}(2U_\tau - u_i)$. ◀

The theorem above is proved under the context of Unrelated. Thus, the theorem holds for HP-LAG-compliant schedulers under Uniform and Identical-Aff because these models are special cases of Unrelated. In Secs. 5 and 6, we demonstrate that the GEDF generalizations discussed in this work are HP-LAG-compliant.

5 GEDF Tardiness Bounds under the Uniform Model

In this section, we show that UG-GEDF, the generalization of IG-GEDF under Uniform in [9], is HP-LAG-compliant. This result enables us to apply Theorem 20 to obtain tardiness bounds for UG-GEDF that are superior to those in [9].

5.1 Refining GEDF for the Uniform Model

IG-GEDF is not SRT-optimal under Uniform. Consider a single-task system $\tau = \{\tau_1\}$ with $C_1 = 1$ and $T_1 = 2$ (hence $u_1 = 0.5$) running on $\pi = \{\pi_1, \pi_2\}$ with $s_1 = 1$ and $s_2 = 0.1$. This system is clearly feasible if τ_1 always executes on the faster processor π_1. Under IG-GEDF, however, it is legal for τ_1 to be continuously scheduled on the slower processor π_2. This would lead to unbounded tardiness, as π_2's speed is lower than τ_1's utilization.

Such counterexamples led Yang and Anderson to define UG-GEDF as below. For the remainder of this section, we assume that processors are indexed by decreasing speed.

UG-GEDF:	At any time instant, the ready job of the pending task with the k^{th} earliest deadline is scheduled on π_k for $k \in [1, m]$.

5.2 HP-LAG-Compliance for UG-GEDF

To prove HP-LAG-compliance, we reference the feasibility condition under Uniform, which references the following definition.

▶ **Definition 21.** *Let* $S_k = \sum_{i=1}^{k} s_i$ *for* $k \leq m$.

When tasks are indexed by decreasing utilization, the feasibility condition is as follows [6, 9].

$$\forall k \leq m : \sum_{i=i}^{k} u_i \leq S_k \tag{18}$$

$$U_\tau \leq S_m \tag{19}$$

In terms of subsets of tasks, this condition is equivalent to

$$\forall \tau' \subseteq \tau : U_{\tau'} \leq S_{\min(|\tau'|,m)}. \tag{20}$$

▶ **Lemma 22.** *UG-GEDF is HP-LAG-compliant under Uniform.*

Proof. By Def. 13, we need only consider the case that task system τ is feasible under Uniform. Let $\tau' \subseteq \tau$ be any subset as defined in HP-LAG for any time instant t. HP-LAG states that the tasks in τ' have earlier deadlines at time t than any tasks outside of τ'. Under UG-GEDF, tasks from τ' occupy the $\min(|\tau'|, m)$ fastest processors. Because UG-GEDF is a non-fluid scheduler (see Def. 4), for any time instant t, for some $\delta > 0$ we have that $(t, t+\delta)$ is a continuous scheduling interval (see Def. 2). For any $t' \in (t, t+\delta)$,

$$\begin{aligned}
\mathsf{LAG}(\tau',\ t') =& \{\text{by Def. 7}\} \\
& \mathsf{LAG}(\tau',\ t) + \sum_{\tau_i \in \tau'} A(\mathcal{I}, \tau_i, t, t') - \sum_{\tau_i \in \tau'} A(\mathcal{R}, \tau_i, t, t') \\
=& \{\text{by Defs. 3 and 6}\} \\
& \mathsf{LAG}(\tau',\ t) + \sum_{\tau_i \in \tau'} u_i(t'-t) - \sum_{\tau_i \in \tau'} A(\mathcal{R}, \tau_i, t, t') \\
=& \{\text{tasks from } \tau' \text{ occupy processors with speeds } s_1, ..., s_{\min(|\tau'|,m)}\} \\
& \mathsf{LAG}(\tau',\ t) + \sum_{\tau_i \in \tau'} u_i(t'-t) - \sum_{i=1}^{\min(|\tau'|,m)} s_i(t'-t) \\
=& \{\text{by Def. 21}\} \\
& \mathsf{LAG}(\tau',\ t) + \sum_{\tau_i \in \tau'} u_i(t'-t) - S_{\min(|\tau'|,m)} \cdot (t'-t) \\
=& \mathsf{LAG}(\tau',\ t) + (t'-t)\left[U_{\tau'} - S_{\min(|\tau'|,m)}\right] \\
\leq& \{t' > t \text{ by definition and } U_{\tau'} \leq S_{\min(|\tau'|,m)} \text{ by (20)}\} \\
& \mathsf{LAG}(\tau',\ t).
\end{aligned}$$

Therefore, UG-GEDF is HP-LAG-compliant. ◀

Because we have proven in Lemma 22 that UG-GEDF is HP-LAG-compliant, we have the following corollary of Theorem 20.

▶ **Corollary 23.** *Under UG-GEDF on a Uniform multiprocessor executing a feasible task system, the tardiness of any task τ_i is at most (15).*

We assumed without loss of generality that tasks (processors) are indexed by decreasing utilizations (speeds) to make stating UG-GEDF and the Uniform feasibility condition simpler. We no longer keep these assumptions in the following sections.

6 GEDF Tardiness Bounds under the Identical Model with Affinities

In this section, we generalize the IG-GEDF scheduling rules under the context of Identical-Aff. We show that the resulting scheduling policy, IA-GEDF, is HP-LAG-compliant. Thus, Theorem 20 ensures bounded tardiness because Identical-Aff is a special case of Unrelated.

Under Identical-Aff, all processors have speed equal to 1.0. As in [8], we use in our analysis the concept of an affinity graph.

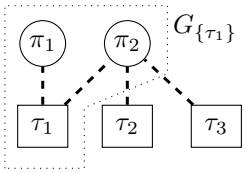


Figure 1 Affinity graph (AG) example.

▶ **Definition 24.** *An affinity graph (AG) G_τ of a task system τ on platform π with affinity masks is a undirected bipartite graph containing one vertex for each task and each processor. An edge exists between τ_i and π_j if and only if $\pi_j \in \alpha_i$. For any $\tau' \subseteq \tau$, we let $G_{\tau'}$ denote the subgraph of G_τ containing only the vertices that correspond to the tasks in τ' and the processors in the union of their affinity masks.*

▶ **Example 25.** An example G_τ for the task system $\tau = \{\tau_1, \tau_2, \tau_3\}$ on $\pi = \{\pi_1, \pi_2\}$ with $\alpha_1 = \{\pi_1, \pi_2\}$ and $\alpha_2 = \alpha_3 = \{\pi_2\}$ is given in Fig. 1. The same figure also shows $G_{\{\tau_1\}}$.

6.1 Refining GEDF for the Identical Model with Affinities

Unlike under Identical or Uniform, it is not always possible to schedule the m tasks with the earliest deadlines under Identical-Aff. Consider the example in Fig. 1 with two processors and three tasks. If all tasks are pending at a time instant t and the deadlines are such that $d_2(t) < d_3(t) < d_1(t)$ holds, then the tasks with the earliest deadlines, τ_2 and τ_3, cannot be simultaneously scheduled because they share a single processor.

The choice of processor assignments can also leave a processor idle, i.e., with no job to execute. Consider again Fig. 1 with the assumption that only τ_1 is pending and is assigned to π_2. Suppose that τ_2 at time instant t releases a job such that $d_1(t) < d_2(t)$. Under IG-GEDF, a lower-priority task such as τ_2 would be scheduled on π_1, but affinity-mask restrictions disallow this. Because $d_1(t) < d_2(t)$, τ_1 has higher priority, and τ_2 is not scheduled, leaving processor π_1 idle. However, forcing τ_1 to migrate to π_1 is a more efficient use of processor capacity in this example. The problem here is that the availability of processors for different tasks under affinity scheduling is not symmetrical.

Under IG-GEDF, a preemption only affects the preempting and preempted tasks and a single processor. We define IA-GEDF to extend this preemption rule to avoid unnecessary idleness. To formally specify IA-GEDF, we require several graph-theory definitions.

▶ **Definition 26.** *A matching of a graph G with edge set E is an edge set $M \subseteq E$ such that no two edges in M share a common vertex. A vertex is matched if it is an endpoint of one of the edges in the matching; otherwise, the vertex is unmatched.*

▶ **Example 27.** An example matching can be found in the left graph of Fig. 2. We have the matching $M = \{(\tau_1, \pi_2), (\tau_3, \pi_3), (\tau_4, \pi_4)\}$, with vertices π_1 and τ_2 being unmatched.

Observe that any assignment of tasks to processors at runtime that obeys the tasks' affinity masks defines a matching of G_τ (and vice versa). While the concept of a matching can be applied to any graph, we restrict attention to only AGs and their subgraphs.

▶ **Definition 28.** *An alternating path p of a matching in a graph is a path that begins with an unmatched "task" vertex, has edges that alternately are in the matching and not in the matching, and ends with a "processor" vertex. An augmenting path p of a matching is an alternating path that ends with an unmatched vertex.*

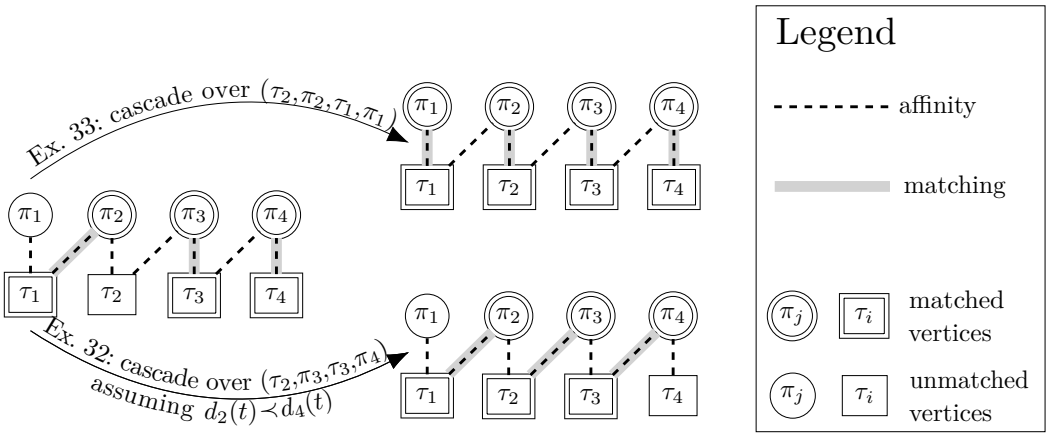


Figure 2 Two examples of a scheduling cascade.

▶ **Example 29.** In the left graph of Fig. 2, $(\tau_2, \pi_3, \tau_3, \pi_4)$ is an alternating path that is not an augmenting path for the given matching, and $(\tau_2,\pi_2,\tau_1,\pi_1)$ is an augmenting path.

Observe that an alternating path must have an odd number of edges because its first and last vertices are task and processor vertices, respectively. The following lemma establishes a relationship between augmenting and alternating paths.

▶ **Lemma 30.** *Consider a matching M for G_τ and $\tau' \subseteq \tau$. Let M' denote the set of edges of M that are present in $G_{\tau'}$. Then, M' is a matching in $G_{\tau'}$. Furthermore, if p is an augmenting path of the matching M' in $G_{\tau'}$, then p is an alternating path of M in G_τ.*

Proof. By Def. 26, no two edges in M share a common vertex. This does not change when removing edges and vertices from G_τ and edges from M, through which we get $G_{\tau'}$ and M', respectively. Thus, M' is also a matching in $G_{\tau'}$.

Let the first vertex of p (as mentioned in the lemma statement) be task τ_i. By Def. 28, τ_i is unmatched in $G_{\tau'}$. By Def. 24, $G_{\tau'}$ contains τ_i and all the processors in its affinity mask. Hence, all edges from τ_i that are in G_τ are also in $G_{\tau'}$. Thus, τ_i is also unmatched in G_τ.

By definition, M' contains all the edges in M that are also in $G_{\tau'}$. Hence, an edge of $G_{\tau'}$ is in M' if and only if it is also in M, which applies to all edges of p because it is contained in $G_{\tau'}$. Because an augmenting path is a special case of an alternating path, edges of p are alternately in and not in M', and hence, alternately in and not in M. This fact and the fact that the first vertex of p, task τ_i, is unmatched in M make p an alternating path in G_τ. ◀

We now have sufficient terminology to define a generalized notion of a preemption that occurs under Identical-Aff and that we call a *scheduling cascade*. Informally, in a scheduling cascade, an unscheduled task is scheduled by making an idle processor busy or by unscheduling a different task, perhaps on a different processor, with a later deadline. Note that this may require several migrations.

▶ **Definition 31.** *A scheduling cascade at time t via the alternating path p in G_τ changes the task-to-processor assignments (i.e., the matching M at t in G_τ) via Alg. 1.*

We can write $p = (\tau_{i_1}, \pi_{j_1}, \tau_{i_2}, \pi_{j_2}, \ldots, \tau_{i_k}, \pi_{j_k})$ for some $k \geq 1$ and task (resp., processor) indicies $i_1, i_2, \ldots, i_k$ (resp., $j_1, j_2, \ldots, j_k$) because the first vertex of p is a task by Def. 28. Because p is alternating, $\forall r \in [1, k-1] : (\tau_{i_{r+1}}, \pi_{j_r}) \in M$ and $\forall r \in [1, k] : (\tau_{i_r}, \pi_{j_r}) \notin M$.

Note that M remains a matching after a scheduling cascade. All tasks and processors in p are unmatched prior to line 6 and each iteration of line 7 matches a distinct task and processor from the other iterations.

Input : Matching M of the AG G_τ at time t;
Alternating path $p = (\tau_{i_1}, \pi_{j_1}, \tau_{i_2}, \pi_{j_2}, \ldots, \tau_{i_k}, \pi_{j_k})$ such that if τ_ℓ exists such that $(\tau_\ell, \pi_{j_k}) \in M$, then $d_{i_1}(t) \prec d_\ell(t)$

1 **if** π_{j_k} *is matched in* M **then**
2 | $\tau_\ell \leftarrow \pi_{j_k}$'s matched task;
3 | Remove edge (τ_ℓ, π_{j_k}) from M;
4 **for** $r \in [1, k-1]$ **do**
5 | Remove edge $(\tau_{i_{r+1}}, \pi_{j_r})$ from M;
6 **for** $r \in [1, k]$ **do**
7 | Add edge (τ_{i_r}, π_{j_r}) to M;
8 **return**

Algorithm 1: The scheduling cascade algorithm.

▶ **Example 32.** Consider the lower scheduling cascade in Fig. 2. In the scheduling cascade, we have $\tau_{i_1} = \tau_2$ and $\pi_{j_k} = \pi_4$. Because π_4 is matched to task τ_4, we have $\tau_\ell = \tau_4$. Thus, the condition $d_{i_1}(t) \prec d_\ell(t)$ becomes $d_2(t) \prec d_4(t)$ in this example, and we remove edge (τ_4, π_4) from the matching (line 3). Of the edges in the alternating path, we remove edge (τ_3, π_3) (line 5) and add edges (τ_2, π_3) and (τ_3, π_4) (line 7). This results in a new matching, as indicated in Fig. 2.

▶ **Example 33.** Consider the higher scheduling cascade in Fig. 2. In the scheduling cascade, we have $\tau_{i_1} = \tau_2$ and $\pi_{j_k} = \pi_1$. We do not execute line 3 because π_1 is unmatched. Of the edges in the alternating path, we remove edge (τ_1, π_2) (line 5) and add edges (τ_2, π_2) and (τ_1, π_1) (line 7). This results in a new matching, as indicated in Fig. 2.

The net result of a scheduling cascade is that task τ_{i_1} that was not scheduled prior to the scheduling cascade is now scheduled, and task τ_ℓ (if it exists) with later deadline than τ_{i_1} is now not scheduled. All other tasks that were scheduled in matching M continue to be scheduled after the scheduling cascade, though the other tasks in p have migrated.

To define our GEDF scheduling policy with affinity masks, we define the task-to-processor assignments at scheduling events, i.e., job releases or job completions, and assume these assignments hold between scheduling events.

IA-GEDF:	At every scheduling event, task-to-processor assignments are made such that afterwards no scheduling cascade is possible. These assignments do not change until the next scheduling event.

One might assume that a simpler scheduling policy may be sufficient, but issues arise when weaker scheduling rules are used. For example, `SCHED_DEADLINE` under Identical-Aff is not HP-LAG-compliant. These details can be found in App. C, available online [7].

Because we do not explicitly specify the task-to-processor assignments, there may exist multiple assignments that satisfy IA-GEDF at every scheduling event. Note that every scheduling cascade either schedules an additional task or replaces a scheduled task with an unscheduled task with an earlier deadline. Thus, the total number of possible scheduling cascades per scheduling event is finite. To show IA-GEDF is possible to implement, we created an $O(mn)$ algorithm that computes task-to-processor assignments at every scheduling event that obeys IA-GEDF. With offline preprocessing, the time complexity is reduced to $O(m + \log n)$ per scheduling event. The algorithm and preprocessing details are available in App. A, available online [7]. In App. C, also available online [7], we show that `SCHED_DEADLINE` under Identical-Aff has higher time complexity per scheduling event.

Note that a preemption in IG-GEDF can be interpreted as a scheduling cascade with an alternating path of a single edge. Thus, IA-GEDF reduces to IG-GEDF under Identical.

6.2 HP-LAG-Compliance for IA-GEDF

Here we consider only feasible task systems τ because HP-LAG-compliance is defined only for such systems. Exact feasibility conditions under Identical-Aff were established in [8], but in our reasoning about IA-GEDF's HP-LAG-compliance, we only use a necessary condition for a feasible task system, provided in Lemma 35.

▶ **Definition 34.** *A matching M of a graph G is a maximal matching if $|M| \geq |M'|$ for any matching M' of G, where $|M|$ denotes the number of edges of the matching M.*

▶ **Lemma 35.** *If a task system τ is feasible under Identical-Aff, then for any task subset $\tau' \subseteq \tau$, a maximal matching M' under $G_{\tau'}$ has $|M'| \geq U_{\tau'}$ edges.*

Proof. Suppose otherwise that τ is feasible and there exists τ' such that a maximal matching M' under $G_{\tau'}$ has $|M'|$ edges with $|M'| < U_{\tau'}$. Because M' is maximal, the tasks of τ' are scheduled on at most $|M'|$ processors at any time instant, and hence, for any schedule $\mathcal{R}$ and time instant t, $\sum_{\tau_i \in \tau'} A(\mathcal{R}, \tau_i, 0, t) \leq |M'|t$. Hence, by Defs. 3, 6, 7, and the definition of $U_{\tau'}$, we have $\mathsf{LAG}(\tau',\ t) = \sum_{\tau_i \in \tau'} A(\mathcal{I}, \tau_i, 0, t) - \sum_{\tau_i \in \tau'} A(\mathcal{R}, \tau_i, 0, t) \geq U_{\tau'}t - |M'|t$. Because $U_{\tau'} > |M'|$, $U_{\tau'}t - |M'|t \to \infty$ as $t \to \infty$. Thus, $\mathsf{LAG}(\tau',\ t)$ is unbounded under any schedule $\mathcal{R}$, making τ' unfeasible, which contradicts the asumption that τ is feasible. ◀

We use Berge's Theorem to prove that IA-GEDF is HP-LAG-compliant. Berge's definition of an augmenting path reduces to Def. 28 in the context of an AG and its subgraphs.

▶ **Theorem 36** (Theorem 1 of [3], Berge's Theorem)**.** *A matching M of a graph G is maximal if and only if there is no augmenting path for M and G.*

▶ **Lemma 37.** *IA-GEDF is HP-LAG-compliant under Identical-Aff.*

Proof. We use Fig. 3 to illustrate the key steps of the proof. Consider a matching M in G_τ defined by the IA-GEDF task-to-processor assignments at time t. Let τ' be as defined in HP-LAG at time t ($\tau' = \{\tau_1, \tau_2, \tau_3\}$ in Fig. 3). Consider a matching M' in $G_{\tau'}$ defined by the assignments of tasks in τ' to processors ($M' = \{(\tau_1, \pi_1), (\tau_3, \pi_2)\}$).

We will first show that M' is maximal in $G_{\tau'}$. Suppose otherwise that M' is not maximal in $G_{\tau'}$. Then, by Theorem 36, there exists an augmenting path p for M' in $G_{\tau'}$ ($p = (\tau_2, \pi_2, \tau_3, \pi_3)$). By Lemma 30, p is an alternating path for M in G_τ.

Consider the task τ_h that is represented by the first vertex of p (τ_2), and the processor π_j that is represented by the last vertex of p (π_3). If there is a task τ_ℓ that is scheduled on π_j (τ_4 on π_3), then $\tau_h \in \tau'$ and $\tau_\ell \in \tau/\tau'$ because p is an augmenting path in $G_{\tau'}$. By the definition of τ' in HP-LAG, we have $d_h(t) < d_\ell(t)$ ($d_2(t) < d_4(t)$). Thus, because we have satisfied the input requirements of Alg. 1, we can perform a scheduling cascade via p at time t (remove (τ_4, π_3) and (τ_3, π_2) and add (τ_3, π_3) and (τ_2, π_2) to the matching), which contradicts our definition of IA-GEDF. Hence, M' is maximal.

Because IA-GEDF produces a non-fluid schedule (Def. 4), the interval $(t, t + \delta)$ must be a continuous scheduling interval for some $\delta > 0$. Thus, for all $t' \in (t, t + \delta)$,

$\mathsf{LAG}(\tau',\ t') =$ {by Def. 7}

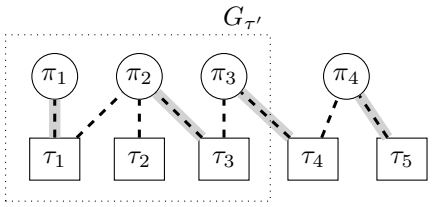


Figure 3 An example graph $G_{\tau'}$ for the proof of Lemma 37. Solid gray edges represent the task-to-processor assignments prior to a scheduling cascade and dashed edges represent affinity.

$$
\begin{aligned}
&\mathsf{LAG}(\tau',\ t) + \sum_{\tau_i \in \tau'} A(\mathcal{I}, \tau_i, t, t') - \sum_{\tau_i \in \tau'} A(\mathcal{R}, \tau_i, t, t') \\
&= \{\text{by Defs. 3 and 6}\} \\
&\quad \mathsf{LAG}(\tau',\ t) + \sum_{\tau_i \in \tau'} (t' - t)u_i - \sum_{\tau_i \in \tau'} A(\mathcal{R}, \tau_i, t, t') \\
&= \{\text{the number of processors scheduling tasks of } \tau' \text{ is } |M'|\} \\
&\quad \mathsf{LAG}(\tau',\ t) + \sum_{\tau_i \in \tau'} (t' - t)u_i - (t' - t)|M'| \\
&= \{\sum_{\tau_i \in \tau'} u_i = U_{\tau'}\} \\
&\quad \mathsf{LAG}(\tau',\ t) + (t' - t)(U_{\tau'} - |M'|) \\
&\leq \{t < t' \text{ and by Lemma 35, } U_{\tau'} \leq |M'|\} \\
&\quad \mathsf{LAG}(\tau',\ t).
\end{aligned}
$$

Therefore, IA-GEDF is HP-LAG-compliant. ◀

Applying Theorem 20 yields the following.

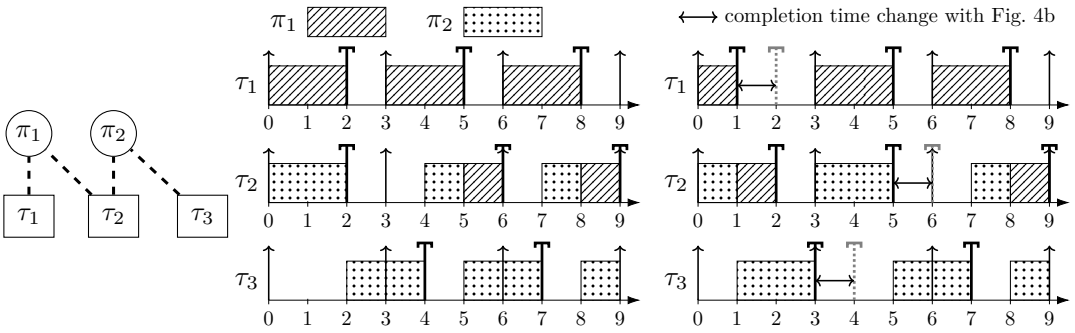
▶ **Corollary 38.** *Under IA-GEDF on a Identical-Aff multiprocessor executing a feasible task system, the tardiness of any task τ_i is at most (15).*

7 Extending to the Sporadic Task Model

As in [9], we made assumptions (P) and (W). It was proven in Theorems 3 and 4 of [9] that any tardiness bounds derived for UG-GEDF under Uniform assuming (P) and (W) hold without these assumptions. Thus, Corollary 23, which establishes tardiness bounds for UG-GEDF under Uniform, applies to sporadic tasks.

It remains to show that these assumptions can be removed from Corollary 38, which pertains to IA-GEDF on Identical-Aff. We use reasoning similar to [9] to show this, though some details are changed due to the different scheduler and platform. Due to space constraints, we present a proof sketch and provide the formal proofs in App. B, available online [7].

Removing assumption (W). The intuition here is that reducing the execution requirement of a job cannot cause jobs to complete later. Consider Figs. 4b and 4c, which show two schedules under IA-GEDF for two periodic instances of the task system defined in Fig. 4a. In Fig. 4b, assumption (W) is true, while in Fig. 4c, job $J_{1,1}$ completes with 1.0 less execution unit than τ_1's WCER of 2.0. As a result, jobs $J_{2,2}$ and $J_{3,1}$ complete earlier in Fig. 4c than in



(a) Affinity graph. **(b)** IA-GEDF schedule with (W). **(c)** IA-GEDF schedule where the first job of τ_1 completes early.

Figure 4 An example task system where removing assumption (W) only causes jobs to complete earlier. When the completion times in the two schedules differ for a job, a dashed gray marker in 4c indicates its original completion time in 4b. Every task has $(C, T) = (2, 3)$.

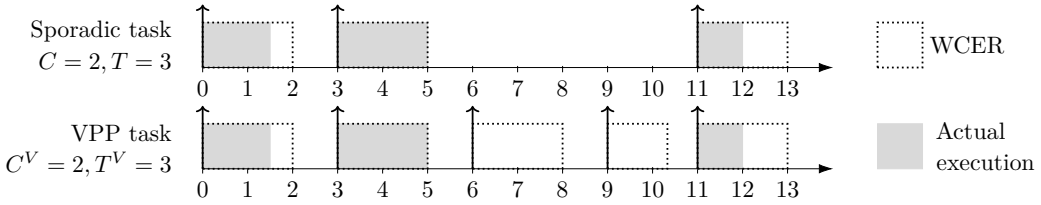


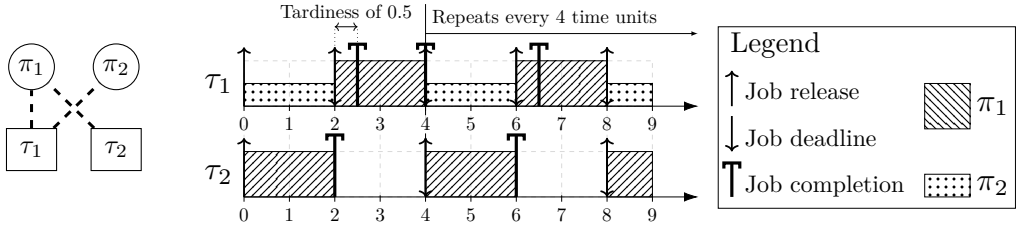
Figure 5 Transformation of a sporadic task to a VPP task.

Fig. 4b, while all other jobs complete at the same time in both schedules (the two schedules converge at time $t = 6$). Thus, reducing the execution time of a job did not increase tardiness for any other job. We prove in App. B, available online [7], that this is always the case. By applying this fact inductively, it follows that tardiness bounds derived assuming (W) hold in systems without (W) under IA-GEDF on Identical-Aff.

Removing assumption (P). In order to remove (P), we use the *varying-period periodic (VPP)* task model, defined in [9]. A VPP task τ_i^V of task set τ^V is defined through its utilization u_i^V. Every job $J_{i,j}^V$ has its own WCER $C_{i,j}^V$. $\max_j C_{i,j}^V$ is denoted as C_i^V. Unlike the sporadic task model, each VPP task τ_i^V releases its first job $J_{i,1}^V$ at time $t = 0$. Afterwards, each job $J_{i,j+1}$ is released *exactly* $T_{i,j}^V = C_{i,j}^V / u_i^V$ time units after the release of $J_{i,j}^V$ for $j \geq 1$. C_i^V / u_i^V is denoted as T_i^V.

It was shown in App. A of [9] that sporadic task systems are special cases of VPP task systems. For example, in Fig. 5, the first timeline represents a sporadic release of jobs by some task τ_i and the second timeline represents a VPP release of jobs by a VPP task τ_i^V with $u_i = u_i^V$ and $C_i = C_i^V$. When the separation between jobs of τ_i is greater than T_i, arbitrarily small VPP jobs with an execution requirement of 0.0 are inserted between the gap in releases, thereby making the second timeline a valid VPP release sequence. Thus, the sporadic release of jobs is also a VPP release of jobs.

Because sporadic task systems are special cases of VPP task systems, our proof obligation is to show that our tardiness bounds apply under the VPP model. The tardiness bounds under IA-GEDF depend only on properties of HP-LAG schedulers, which in turn depend only on the properties in Sec. 3. The fact that these properties hold under the VPP model with assumption (W) (with some reinterpretation of parameters, e.g., substituting u_i with u_i^V and C_i with $C_{i,j}^V$ in proofs) was shown in App. A of [9]. Hence, because removing (W)



(a) Affinity graph. **(b)** A SRT schedule (with a maximum tardiness of 0.5) of a periodic instance of the task system in Theorem 39. The height of a scheduling interval represents the speed of the allocated processor.

Figure 6 Task system considered in the proof of Theorem 39.

does not increase tardiness, Corollary 38 holds with VPP task systems (substituting C_{max} with $\max_i C_i^V$ in (15)). If the VPP task system is also a sporadic system, as in Fig. 5, then $\max_i C_i^V = \max_i C_i = C_{max}$. Thus, the tardiness bound in Corollary 38 for sporadic tasks is exactly as written in (15).

8 Problems with Extending to the Uniform Model with Affinities

Of the models listed in Sec. 2.1, we have not addressed Uniform-Aff and Unrelated. In this section, we explain why extending our proof techniques to these models is difficult. The exact proof strategy used in this work cannot be directly applied to the more general models in Sec. 2.1 because HP-LAG may not hold.

▶ **Theorem 39.** *No non-fluid scheduler always satisfies HP-LAG under Uniform-Aff.*

Proof. We prove the theorem by constructing a feasible task system, deadline ordering, and Uniform-Aff platform for which no scheduler satisfies HP-LAG. Consider the task system $\tau = \{\tau_1, \tau_2\}$ with $(C_1, T_1) = (3, 2)$ and $(C_2, T_2) = (4, 4)$. Then, $u_1 = 1.5$ and $u_2 = 1$. τ runs on two processors $\pi = \{\pi_1, \pi_2\}$ with $s_1 = 2$ and $s_2 = 1$. G_τ is illustrated in Fig. 6a.

We know this system is feasible from the schedule in Fig. 6b, which contains a timeline for each task that describes what processor, if any, schedules the task at any time instant. The schedule repeats every four time units. This schedule provides six units of execution to τ_1 and four units of execution to τ_2 every four time units. Because the schedule provides execution to each task at a long-run rate equal to its utilization ($6/4 = 3/2 = u_1$ and $4/4 = u_2$), both tasks have bounded tardiness.

Let the deadline ordering at some time instant t be $d_1(t) < d_2(t)$ (e.g., $t = 0$). Suppose some non-fluid scheduling algorithm is HP-LAG-compliant at t. By Def. 4, we have that $(t, t + \delta)$ is a continuous scheduling interval for some $\delta > 0$. Note that τ' as defined in HP-LAG for this task system may be either $\{\tau_1\}$ or $\{\tau_1, \tau_2\}$ at time t. HP-LAG states for these two task subsets that $\forall t' \in (t, t + \delta)$:

$$\mathsf{LAG}(\{\tau_1\},\ t') \leq \mathsf{LAG}(\{\tau_1\},\ t) \tag{21}$$

$$\mathsf{LAG}(\{\tau_1,\ \tau_2\}, t') \leq \mathsf{LAG}(\{\tau_1,\ \tau_2\}, t) \tag{22}$$

We show that τ_1 must execute on π_1 over $(t, t + \delta)$ by contradiction. Consider otherwise that there exists a time instant in $(t, t + \delta)$ where τ_1 executes on π_2. Because $(t, t + \delta)$ is a continuous scheduling interval, by Def. 2, τ_1 executes on π_2 for all $t' \in (t, t + \delta)$. Thus,

$\mathsf{LAG}(\{\tau_1\},\ t') =$ {by Def. 7}

$$
\begin{aligned}
& \mathsf{LAG}(\{\tau_1\},\ t) + A(\mathcal{I}, \tau_1, t, t') - A(\mathcal{R}, \tau_1, t, t') \\
= & \{\text{by Defs. 3 and 6}\} \\
& \mathsf{LAG}(\{\tau_1\},\ t) + u_1(t' - t) - A(\mathcal{R}, \tau_1, t, t') \\
= & \{\text{by Def. 3 and the assumption that } \tau_1 \text{ is scheduled on } \pi_2\} \\
& \mathsf{LAG}(\{\tau_1\},\ t) + u_1(t' - t) - s_2(t' - t) \\
= & \{u_1 = 1.5 \text{ and } s_2 = 1\} \\
& \mathsf{LAG}(\{\tau_1\},\ t) + 0.5(t' - t) \\
> & \{t' - t > 0 \text{ by definition}\} \\
& \mathsf{LAG}(\{\tau_1\},\ t),
\end{aligned}
$$

which contradicts (21). A similar contradiction arises when τ_1 is not executing for any instant in $(t, t + \delta)$, so any HP-LAG compliant scheduler *must* schedule τ_1 on π_1 during $(t, t + \delta)$. Scheduling τ_1 on π_1 means τ_2 is not scheduled over this interval, because π_1 is the only processor in τ_2's affinity mask.

Consider the LAG of $\{\tau_1, \tau_2\}$ for all $t' \in (t, t + \delta)$ given that τ_1 executes on π_1 and τ_2 is not executing over this interval. Through reasoning similar to that above, we can conclude $\mathsf{LAG}(\{\tau_1,\ \tau_2\}, t') = \mathsf{LAG}(\{\tau_1,\ \tau_2\}, t) + 0.5(t' - t) > \mathsf{LAG}(\{\tau_1,\ \tau_2\}, t)$, which contradicts (22). Because no non-fluid scheduler can simultaneously satisfy (21) and (22), no non-fluid scheduler can satisfy HP-LAG for this feasible task system. ◀

9 Conclusion

We have derived the first ever GEDF tardiness bounds that are polynomial in the number of processors under Uniform. We have also derived for the first time generalized GEDF scheduling rules that are provably SRT-optimal under Identical-Aff. This result shows that the open problem mentioned by Peter Zijlstra and Luca Abeni can be resolved by altering `SCHED_DEADLINE` to be HP-LAG-compliant. In App. A, available online [7], we have provided an algorithm that implements our generalized GEDF scheduling rules with lower time complexity per scheduling event than `SCHED_DEADLINE`, given some preprocessing.

Note that the proofs in Sec. 4 only require that the values of β satisfy certain linear constraints. Thus, lower β values than ours can be derived using linear programming (though the constraint set grows exponentially with the task count). This suggests that the properties in Sec. 3 are sufficient to derive tighter analytical tardiness bounds than ours.

In future work, we will investigate dynamic task systems, where tasks may enter or exit the system and affinity masks may change at runtime. The interaction of affinity masks with dynamic task systems is particularly relevant to `SCHED_DEADLINE`, as admission control with affinity masks is currently broken. We plan to investigate what restrictions must be placed on these dynamics to avoid compromising bounded tardiness, as done in prior work [4] on GEDF without affinity masks. We are also interested in how overhead accounting and non-preemptive sections might be handled as well as how algorithms that satisfy IA-GEDF might be constructed with lower time complexity than in App. A.

The SRT-optimality of GEDF on more general platforms also remains an open problem. We have shown via a counterexample that HP-LAG, a property that applies to both Uniform and Identical-Aff individually, does not hold when the models are combined. It is unknown whether a weaker version of HP-LAG exists that applies to the more general processor models while still being sufficient to bound tardiness. If not, these open problems may require new proof techniques.

References

1. Luca Abeni. SCHED_DEADLINE: a real-time CPU scheduler for Linux. 2nd TuTor at the 38th IEEE Real-Time Systems Symposium, 2017. URL: https://tutor2017.inria.fr/sched_deadline/.
2. Luca Abeni et al. Deadline task scheduling. Linux kernel documentation, 2018. URL: https://github.com/torvalds/linux/blob/master/Documentation/scheduler/sched-deadline.txt.
3. Claude Berge. Two theorems in graph theory. *Proceedings of the National Academy of Sciences*, 43(9):842–844, 1957.
4. Aaron Block, UmaMaheswari C. Devi, and James H. Anderson. Task reweighting under global scheduling on multiprocessors. In *Proceedings of the 18th Euromicro Conference on Real-Time Systems*, pages 123–167, 2006.
5. UmaMaheswari C. Devi and James H. Anderson. Tardiness bounds under global EDF scheduling on a multiprocessor. *Real-Time Systems*, 38(2):133–189, 2008.
6. Shelby Funk, Joel Goossens, and Sanjoy Baruah. On-line scheduling on uniform multiprocessors. In *Proceedings of the 22th IEEE Real-Time Systems Symposium*, pages 183–192, 2001.
7. Stephen Tang, Sergey Voronov, and James H. Anderson. GEDF tardiness: Open problems involving uniform multiprocessors and affinity masks resolved. Full version of this paper, available at http://www.cs.unc.edu/~anderson/papers.html.
8. Sergey Voronov and James H. Anderson. AM-Red: An optimal semi-partitioned scheduler assuming arbitrary affinity masks. In *Proceedings of the 39th IEEE Real-Time Systems Symposium*, pages 408–420, 2018.
9. Kecheng Yang and James H. Anderson. On the soft real-time optimality of global EDF on uniform multiprocessors. In *Proceedings of the 38th IEEE Real-Time Systems Symposium*, pages 319–330, 2017.
10. Peter Zijlstra. An update on real-time scheduling on Linux. Keynote talk at the 29th Euromicro Conference on Real-Time Systems, 2017. URL: https://www.ecrts.org/archives/index.php?id=284.