On the Soft Real-Time Optimality of Global EDF on Uniform Multiprocessors*

Kecheng Yang and James H. Anderson
Department of Computer Science, University of North Carolina at Chapel Hill

Abstract

It has long been known that the global earliest-deadline-first (GEDF) scheduler is soft real-time (SRT) optimal *for sporadic task systems executing on identical multiprocessor platforms, regardless of whether task execution is preemptive or non-preemptive. This notion of optimality requires deadline tardiness to be provably bounded for any feasible task system. In recent years, there has been interest in extending these SRT optimality results to apply to uniform heterogeneous platforms, in which processors may have different* speeds*. However, it was recently shown that non-preemptive GEDF is* not *SRT optimal on such platforms. The remaining case, preemptive GEDF, has turned out to be quite difficult to tackle and has remained open for a number of years. In this paper, this case is resolved by showing that preemptive GEDF is indeed SRT optimal on uniform platforms, provided a certain job migration policy is used.*

1 Introduction

Multicore machines provide considerable processing power within a constrained size, weight, and power envelope. As a result, they are quickly emerging as the hardware platform of choice when hosting computationally intensive real-time applications. Although this emergence is well underway, research directed at finding good real-time schedulers for such platforms continues to be relevant. To fully harness the potential of multicore machines, such a scheduler should be reasonable to implement and preferably be *optimal*, meaning that it can correctly schedule any feasible task system. A task system is *feasible* if it is possible to schedule it (via *some* method) without violating timing constraints.

One multiprocessor real-time scheduler that has received considerable attention is the preemptive *global earliest-deadline-first* (*GEDF*) scheduler.[1] GEDF generalizes the preemptive uniprocessor earliest-deadline-first (EDF) scheduler by using a single run queue to schedule all processors. EDF is optimal on uniprocessors and is straightforward to implement, so the motivation for considering its generalization in GEDF is clear. Unfortunately, in hard real-time (HRT) systems, in which every deadline must be met, GEDF is not optimal, due to the Dhall Effect [6]. However, for soft real-time (SRT) systems, where timing correctness is defined by requiring deadline tardiness to be bounded, GEDF *is* optimal, assuming all processors are identical [5].

*Work supported by NSF grants CNS 1409175, CPS 1446631, and CNS 1563845, AFOSR grant FA9550-14-1-0161, ARO grant W911NF-14-1-0499, and funding from General Motors.

[1]References to "GEDF" without qualification should henceforth be taken to mean *preemptive* GEDF.

In fact, this optimality result extends to non-preemptive GEDF (NP-GEDF), under which jobs (*i.e.*, task invocations) execute non-preemptively.

Heterogeneous multiprocessor platforms. The multicore revolution is currently undergoing a second wave of innovation in the form of heterogeneous processing elements. This evolution is further complicating software design processes that were already being challenged on account of the significant parallelism that exists in homogeneous multicore platforms. With heterogeneity, choices must be made when allocating hardware resources to software components. The need to resolve such choices can add considerable complexity to resource allocation.

One heterogeneous platform model that has been studied extensively is the *uniform* model [13]. This model differs form the conventional *identical* model in that processors may be of different *speeds*. The uniform model is viewed as an important stepping stone between the identical model and the *unrelated* model, which allows different tasks to experience unequal speed differences when moving from one processor to another. Additionally, the uniform model can be seen as an idealization of various *restricted processor supply* models. Under such a supply model, a processor may be partially available to a given application and hence can be approximated as executing at a slower speed.

Real-time scheduling on uniform multiprocessors has been widely studied (*e.g.*, [1, 3, 4, 7, 9]), but mostly with respect to HRT systems. Recently, we ourselves devised a semi-partitioned scheduler [17] that can be configured to be either HRT or SRT optimal on uniform platforms by leveraging the classical Level Algorithm [10] as a subroutine. The Level Algorithm has also been applied by Funk *et al.* [8] to obtain a scheduler that is HRT optimal for implicit-deadline periodic task systems. An understanding of the Level Algorithm is useful for appreciating some of the subtleties that arise on uniform platforms, so we have provided an introduction to it in an appendix.

To our knowledge, all existing (HRT or SRT) schedulers that are optimal on uniform platforms require job priorities to vary during runtime, usually in a complicated way (like in the Level Algorithm). In contrast, GEDF requires simple logic to implement and is also a *job-level-fixed-priority* scheduler, which implies that existing real-time locking protocols can be potentially applied under it [2]. Therefore, while GEDF is not HRT optimal on uniform platforms, there are good reasons for investigating its SRT optimality in this context.

SRT optimality of GEDF on uniform platforms. Given the aforementioned SRT optimality results concerning

GEDF and NP-GEDF on identical platforms, one might expect that similar results can be easily shown for uniform platforms. However, we recently established the *non*-optimality of NP-GEDF in this context via a counterexample involving only two processors [16]. In the same paper, we were also able to establish the optimality of GEDF for such two-processor platforms. However, the general problem for the preemptive case has remained open. We ourselves have worked on this open problem intensively for over three years and have written about it in two papers [16, 18]. Others have considered it as well [14].

The elusiveness of this open problem stems from the fact that the feasibility condition for the uniform case is much more nuanced than that for the identical case. In particular, the uniform feasibility condition allows tasks to exist that have utilizations exceeding the speeds of certain processors. If such a task τ_i is continuously scheduled on a processor that is "too slow," then the tardiness of its jobs will increase without bound. In contrast, if τ_i were to be scheduled continuously on an identical platform, then its tardiness would not increase, assuming its utilization is at most one (which is required for feasibility in the identical case). In particular, none of its jobs would be able to execute for longer than T_i time units, where T_i is its period. This very issue of tasks executing on processors that are "too slow" directly "breaks" the prior GEDF optimality proof for identical platforms [5] when extended to the uniform case.

Contributions. In this paper, we close the longstanding open problem mentioned above by proving that GEDF is SRT optimal on uniform platforms. Because the prior proof for identical platforms "breaks" in the uniform case, this new result required devising several innovative proof techniques relevant to uniform platforms.

The version of GEDF that we consider uses an intuitive job migration policy: jobs with earlier deadlines are assigned to faster processors. Note that such policies are important in our context. For example, if a task were allowed to continuously execute on a processor that is "too slow," then as discussed above, its tardiness would be unbounded.

The tardiness bounds we provide are not tight. Still, they are the first such bounds ever derived for GEDF in the uniform case that apply to *any* feasible task system. Moreover, their existence eliminates the possibility of a counterexample in which a feasible task system exhibits unbounded tardiness under GEDF. In contrast, such counterexamples are known to exist under NP-GEDF [16], as mentioned earlier. Combining the results of this paper with those in [16], we now have a complete picture regarding the SRT optimality of GEDF and NP-GEDF on uniform platforms.

Organization. In the rest of the paper, we provide needed background (Sec. 2), establish our main tardiness-bound result (Sec. 3), and conclude (Sec. 4). Due to space constraints, we limit attention in the main body of the paper to periodic task systems that satisfy a certain technical restriction. In an appendix, we explain how to extend our results to sporadic task systems that are not restricted.

2 Background

When considering the issue of heterogeneity, processor speed becomes an important issue. The *speed* of a processor refers to the amount of work completed in one time unit when a task is executed on that processor. The following speed-oriented classification of multiprocessor platforms has been widely accepted [7, 13].

- **Identical multiprocessors.** Every task is executed on any processor at the same speed, which is usually normalized to be 1.0 for simplicity.
- **Uniform multiprocessors.** Different processors may have different speeds, but on a given processor, every task is executed at the same speed. The speed of processor p is denoted s_p.
- **Unrelated multiprocessors.** The execution speed of a task depends on both the processor on which it is executed and the task itself, *i.e.*, a given processor may execute different tasks at different speeds.

In this paper, our focus is uniform multiprocessors. Specifically, we consider the scheduling of a set τ of n sequential tasks on a uniform platform[2] π consisting of m processors, where the processors are indexed by their speeds in non-increasing order, *i.e.*, $s_i \geq s_{i+1}$ for $i = 1, 2, \ldots, m-1$. We denote the sum of k largest speeds on π as $S_k = \sum_{i=1}^{k} s_i$ for $k = 1, 2, \ldots, m$. Furthermore, we assume $m \geq 2$, for otherwise, uniprocessor analysis can be applied. We also assume $n \geq m$, for otherwise, there is no point in ever scheduling any task on any of the $m - n$ slower processors, so m and n can conceptually be deemed as equal in this case.

We consider tasks that are either *sporadic* or *periodic*. A sporadic (respectively, periodic) task τ_i releases a sequence of *jobs* with a minimum (respectively, exact) separation of T_i time units between invocations. The parameter T_i is called the *period* of τ_i. τ_i also has a *worst-case execution requirement* C_i, which is defined as the maximum execution time of any one job (invocation) of τ_i on a unit-speed processor. We let $C_{\max} = \max\{C_i \mid 1 \leq i \leq n\}$. The *utilization* of task τ_i is given by $u_i = C_i/T_i$. We assume that tasks are indexed in non-increasing order by utilization, *i.e.*, $u_i \geq u_{i+1}$ for $i = 1, 2, \ldots, n-1$. We denote the sum of k largest utilizations in τ as $U_k = \sum_{i=1}^{k} u_i$ for $k = 1, 2, \ldots, n$. Furthermore, we denote the ratio between the largest and the smallest utilizations as $\rho = u_1/u_n$. As for scheduling, we assume that deadlines are *implicit*, *i.e.*, each task τ_i has a *relative deadline* parameter equal to its period T_i. Furthermore, because tasks are *sequential*, intra-task parallelism is not allowed, and an invocation of a task cannot commence execution until all previous invocations of that task complete. Throughout the paper, we assume that time is continuous.

[2]We use the terminology "uniform platform" and "uniform multiprocessor" interchangeably.

We let $J_{i,j}$ denote the j^{th} job (or invocation) of task τ_i. Job $J_{i,j}$ has a release (or arrival) time denoted $a_{i,j}$, an absolute deadline denoted $d_{i,j} = a_{i,j} + T_i$, and a completion (or finish) time denoted $f_{i,j}$. The *tardiness* of $J_{i,j}$ is defined by $\max\{0, f_{i,j} - d_{i,j}\}$ and its *response time* by $f_{i,j} - a_{i,j}$. The *tardiness* of *task* τ_i in a schedule is the maximum tardiness of any of its jobs in that schedule. A job is *pending* if it is released but has not completed, and is *ready* if it is pending and all preceding jobs of the same task have completed.

A task set is *HRT schedulable* (respectively, *SRT schedulable*) under a given scheduler if each task can be guaranteed zero (respectively, bounded) tardiness under that scheduler. A task set is *HRT feasible* (respectively, *SRT feasible*) if it is HRT schedulable (respectively, SRT schedulable) under some scheduler. A given scheduler is *HRT optimal* (respectively, *SRT optimal*) if any HRT feasible (respectively, SRT feasible) task set is HRT schedulable (respectively, SRT schedulable) under it.

Ideal schedule. We define an *ideal multiprocessor* $\pi_{\mathcal{I}}$ for the task set τ as one that consists of n uniform processors where the speeds of the n processors exactly match the utilizations of the n tasks in τ, respectively, *i.e.*, the speed of the i^{th} processor is $s_i^{\mathcal{I}} = u_i$ for $i = 1, 2, \ldots, n$. We define the *ideal schedule* $\mathcal{I}$ to be the partitioned schedule for τ on $\pi_{\mathcal{I}}$, where each task τ_i in τ is assigned to the processor of speed $s_i^{\mathcal{I}}$. Then, in $\mathcal{I}$, every job in τ commences execution at its release time and completes execution within one period (it exactly executes for one period if and only if its actual execution requirement matches its worst-case execution requirement). Thus, all deadlines are met in $\mathcal{I}$.

Definition of lag**.** Let $\mathsf{A}(\mathcal{S}, \tau_i, t_1, t_2)$ denote the cumulative processor capacity allocated to task τ_i in an arbitrary schedule $\mathcal{S}$ within the time interval $[t_1, t_2)$. Because intra-task parallelism is strictly forbidden, we have

$$0 \leq \mathsf{A}(\mathcal{S}, \tau_i, t_1, t_2) \leq s_1 \cdot (t_2 - t_1). \tag{1}$$

Furthermore, by the definition of the ideal schedule $\mathcal{I}$,

$$0 \leq \mathsf{A}(\mathcal{I}, \tau_i, t_1, t_2) \leq u_i \cdot (t_2 - t_1). \tag{2}$$

Also, if τ_i releases jobs periodically and every job's actual execution requirement equals its worst case of C_i, then for any t_1 and t_2 such that $a_{i,1} \leq t_1 \leq t_2$,

$$\mathsf{A}(\mathcal{I}, \tau_i, t_1, t_2) = u_i \cdot (t_2 - t_1). \tag{3}$$

For an arbitrary schedule $\mathcal{S}$, we denote the difference between the allocation to a task τ_i in $\mathcal{I}$ and in $\mathcal{S}$ within time interval $[0, t)$ as

$$\mathsf{lag}(\tau_i, t, \mathcal{S}) = \mathsf{A}(\mathcal{I}, \tau_i, 0, t) - \mathsf{A}(\mathcal{S}, \tau_i, 0, t). \tag{4}$$

The lag function captures the allocation difference between an arbitrary actual schedule $\mathcal{S}$ and the ideal schedule $\mathcal{I}$. If $\mathsf{lag}(\tau_i, t, \mathcal{S})$ is positive, then $\mathcal{S}$ has performed less work on τ_i until time t, *i.e.*, τ_i is "under-allocated," while if $\mathsf{lag}(\tau_i, t, \mathcal{S})$ is negative, then τ_i is "over-allocated." Also, for any two time instants t_1 and t_2 where $t_1 \leq t_2$, we have

$$\begin{aligned}\mathsf{lag}(\tau_i, t_2, \mathcal{S}) = {} & \mathsf{lag}(\tau_i, t_1, \mathcal{S}) + \\ & \mathsf{A}(\mathcal{I}, \tau_i, t_1, t_2) - \mathsf{A}(\mathcal{S}, \tau_i, t_1, t_2).\end{aligned} \tag{5}$$

A necessary and sufficient SRT feasibility condition. For HRT task sets, Funk *et al.* [8] showed that a set of implicit-deadline periodic tasks is feasible on a uniform platform if and only if the following constraints hold:

$$U_n \leq S_m, \tag{6}$$

$$U_k \leq S_k, \text{ for } k = 1, 2, \ldots, m-1. \tag{7}$$

It can be shown that this constraint set is also a feasibility condition for implicit-deadline *sporadic* tasks. Furthermore, the sufficiency of this constraint set for HRT task sets implies its sufficiency for SRT task sets. In fact, these constraints are necessary for SRT task sets as well. To see this, note that if $U_n > S_m$ holds (contrary to (6)), then the total workload over-utilizes the platform, so some task will be unboundedly tardy if tasks release jobs as soon as possible and always execute for their worst-case costs. Furthermore, if $U_k > S_k$ holds (contrary to (7)), then the set of k highest-utilization tasks will be "under-allocated" at every time instant if they release jobs as soon as possible and always execute for their worst-case costs. This is because k tasks can be allocated to at most k processors at any time instant and the sum of the speeds of *any* k processors is at most S_k. Thus, $\sum_{i=1}^{k} \mathsf{lag}(\tau_i, t, \mathcal{S})$ will increase unboundedly, which implies that $\mathsf{lag}(\tau_i, t, \mathcal{S})$ increases unboundedly for some i. This implies that task τ_i will be unboundedly tardy.

To summarize, (6) and (7) are also a necessary and sufficient feasibility condition for SRT task sets. Therefore, when henceforth referring to this constraint set as a feasibility condition, we do not need to further specify whether this is meant for HRT or SRT task sets.

The GEDF scheduler. From a scheduling point of view, uniform platforms differ from identical ones in a significant way: on a uniform platform, besides *which* tasks are scheduled at any time, the scheduler must also decide *where* they are scheduled, because different processors may have different speeds. Thus, we must refine the notion of GEDF scheduling to be clear about where tasks are scheduled. In this paper, we assume the following.

> **(G)** If at most m jobs are ready, then all ready jobs are scheduled; otherwise, the m ready jobs with earliest deadlines are scheduled. At any time, the ready job with the k^{th} earliest deadline is scheduled on the k^{th} fastest processor for any k. (Note that this implies that a job may migrate from one processor to another during its execution.) Deadline ties are broken arbitrarily.

Henceforth, all references to GEDF are assumed to mean GEDF as defined by Policy (G), unless specified otherwise.

Strictly speaking, Policy (G) implies that a release or completion of a single job could cause up to $m-1$ running jobs to migrate. However, in practice, the number of different speeds is usually quite limited and there is no point in migrating a job between same-speed processors, so such frequent migrations are not actually necessary. We could incorporate such details into Policy (G) without altering any of our results, but we will refrain from doing so for ease of exposition.

At a given time instant t, we say that a *task* is *pending* if it has any pending jobs at time t. If task τ_i is pending at time t, then it has exactly one ready job $J_{i,j}$ at time t. The deadline of that job is called the *effective* deadline of τ_i at time t and is denoted $d_i(t) = d_{i,j}$. Similarly, the *effective* release time of τ_i at time t is denoted $a_i(t) = a_{i,j}$. Because deadlines are implicit, $d_i(t) = a_i(t) + T_i$. Also, $a_i(t) \leq t$ holds, for otherwise, $J_{i,j}$ would not be ready at time t. The following lemma gives a sufficient lag-based condition for a task to be pending.

Lemma 1. *If* $\mathsf{lag}(\tau_i, t, \mathcal{S}) > 0$*, then* τ_i *is pending at time* t *in* $\mathcal{S}$*.*

Proof. Suppose that $\mathsf{lag}(\tau_i, t, \mathcal{S}) > 0$ holds but τ_i is not a pending task at time t in $\mathcal{S}$. Then, all jobs of τ_i released at or before t have completed by time t. Thus, letting W denote the total actual execution requirement of all such jobs, we have $\mathsf{A}(\mathcal{S}, \tau_i, 0, t) = W$. In the ideal schedule $\mathcal{I}$, only released jobs can be scheduled and will not execute for more than their actual execution requirement. Thus, $\mathsf{A}(\mathcal{I}, \tau_i, 0, t) \leq W$ holds as well. By (4), these facts imply $\mathsf{lag}(\tau_i, t, \mathcal{S}) = \mathsf{A}(\mathcal{I}, \tau_i, 0, t) - \mathsf{A}(\mathcal{S}, \tau_i, 0, t) \leq 0$. This contradicts our assumption that $\mathsf{lag}(\tau_i, t, \mathcal{S}) > 0$ holds. $\square$

Prior results and the remaining open problem. As mentioned earlier in Sec. 1, the key difficulty faced when trying to extend prior tardiness analysis for identical platforms [5, 12] to uniform ones is that, in the uniform case, tasks can execute on processors that are "too slow." The specific problematic property required in the prior analysis is the following.

(P) If any job $J_{i,j}$ executes continuously, then it must complete within T_i time units, regardless of the processor on which it executes.

Clearly, (P) can be violated if $J_{i,j}$ executes entirely on processors of speed less than u_i.

It is tempting to obviate all problematic issues pertaining to Property (P) by simply enforcing scheduling policies that uphold it. Such an approach was taken by Tong and Liu [14], who considered a variant of GEDF in which each task τ_i is only allowed to execute on processors with speed at least u_i. However, such a requirement results in non-optimal scheduling. For example, Tong and Liu's GEDF variant is not able to correctly schedule a set of two tasks on two processors such that $u_1 = 2$, $u_2 = 2$, $s_1 = 3$, and $s_2 = 1$. From (6) and (7), we see that this task set is feasible. However, under their algorithm, a task τ_i can only execute on a processor of speed at least u_i, so both tasks in this example must exclusively execute on the processor with speed $s_1 = 3$. That processor will be over-utilized if each task releases jobs as soon as possible and always executes for its worst-case cost, since $u_1 + u_2 = 4 > s_1 = 3$.

In other work, we successfully eliminated the need for Property (P) by relaxing the task model to allow consecutive jobs of the same task to execute in parallel. Under this relaxed task model, we were able to establish the SRT optimality of GEDF on uniform platforms [15].

For the sequential task model being considered in this paper, we also found that Property (P) is not necessary if the underlying uniform platform has only two processors [16]. However, we believe that it is unlikely that the particular proof strategy used in that paper can be extended to the more general m-processor case.

To clearly place our contribution in its proper context in light of this prior work, we emphasize here several assumptions made hereafter in solving the open problem considered in this paper:

- we are interested in *any* feasible task set, *i.e.*, no constraints on task utilizations other than (6) and (7) are assumed;
- intra-task parallelism is strictly forbidden, *i.e.*, jobs of the same task must execute in sequence;
- the uniform platform may have m processors, where m can be any positive integer value.

3 Tardiness Bounds

In this section, we prove tardiness bounds for an arbitrary feasible *periodic* task set on a uniform platform π, under the following assumption.

(A) Every job of any task executes for its worst-case execution requirement of C_i.

In the rest of this section, we restrict attention to a periodic task set τ for which Assumption (A) holds. Our objective is to derive tardiness bounds when the GEDF scheduler is used to schedule τ. We do so by reasoning about lag values in an arbitrary GEDF schedule $\mathcal{S}$ for τ. The concept of lag is useful for our purposes because a task that has positive lag at one of its deadlines will have a tardy job. Focusing on periodic task sets where Assumption (A) holds facilitates much of the lag-based reasoning that is needed. Our results can be extended so that Assumption (A) is not required and so that sporadic tasks can be supported. We defer consideration of such extensions until later.

Properties of lag values and deadlines. We begin by proving a number of properties concerning lag values and deadlines and relationships between the two. The first such property is given in the following lemma.

Lemma 2. *If task* τ_i *is pending at time* t *in* $\mathcal{S}$*, then its effective deadline* $d_i(t)$ *has the following relationship with* $\mathsf{lag}(\tau_i, t, \mathcal{S})$*.*

$$t - \frac{\mathsf{lag}(\tau_i, t, \mathcal{S})}{u_i} < d_i(t) \leq t - \frac{\mathsf{lag}(\tau_i, t, \mathcal{S})}{u_i} + T_i \qquad (8)$$

Proof. Let $e_i(t)$ denote the remaining execution requirement for the ready job $J_{i,j}$ of τ_i at time t in $\mathcal{S}$. Because $J_{i,j}$ is ready at time t, it has not finished execution by then, so

$$0 < e_i(t) \le C_i. \tag{9}$$

Furthermore, all jobs of τ_i prior to $J_{i,j}$ have completed by time t in $\mathcal{S}$. Let W denote the total execution requirement for all of these jobs. Then, given Assumption (A),[3]

$$\mathsf{A}(\mathcal{S}, \tau_i, 0, t) = W + C_i - e_i(t). \tag{10}$$

Now consider the ideal schedule $\mathcal{I}$. In it, all jobs of τ_i prior to $J_{i,j}$ have completed by time $a_i(t) \le t$. Given Assumption (A), within $[a_i(t), t)$, $\mathcal{I}$ continuously[4] executes job $J_{i,j}$ at a rate of u_i. Thus,

$$\mathsf{A}(\mathcal{I}, \tau_i, 0, t) = W + (t - a_i(t))u_i. \tag{11}$$

Therefore, an expression for $\mathsf{lag}(\tau_i, t, \mathcal{S})$ can be derived as follows.

$$\begin{aligned}
\mathsf{lag}(\tau_i, t, \mathcal{S}) = {} & \{\text{by (4)}\} \\
& \mathsf{A}(\mathcal{I}, \tau_i, 0, t) - \mathsf{A}(\mathcal{S}, \tau_i, 0, t) \\
= {} & \{\text{by (10) and (11)}\} \\
& (t - a_i(t))u_i - (C_i - e_i(t)) \\
= {} & \{\text{because } d_i(t) = a_i(t) + T_i\} \\
& (t - d_i(t) + T_i)u_i - (C_i - e_i(t)) \\
= {} & \{\text{because } T_i \cdot u_i = C_i\} \\
& (t - d_i(t))u_i + e_i(t)
\end{aligned}$$

By (9) and the above expression for $\mathsf{lag}(\tau_i, t, \mathcal{S})$, we have

$$(t - d_i(t))u_i < \mathsf{lag}(\tau_i, t, \mathcal{S}) \le (t - d_i(t))u_i + C_i. \tag{12}$$

Rearranging the terms in (12) yields (8). □

Corollary 1. *If* $\mathsf{lag}(\tau_i, t, \mathcal{S}) \le L$ *for all* t*, then the tardiness of task* τ_i *is at most* L/u_i*.*

Proof. Suppose that

$$\mathsf{lag}(\tau_i, t, \mathcal{S}) \le L \tag{13}$$

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

$$t - d_{i,j} > L/u_i. \tag{14}$$

Because $J_{i,j}$ is pending at time t, τ_i is a pending task at time t and its ready job at time t cannot be a job released later than $J_{i,j}$. Thus, τ_i's effective deadline at t satisfies $d_i(t) \le d_{i,j}$. Therefore,

$$\begin{aligned}
t - d_i(t) \ge {} & t - d_{i,j} \\
> {} & \{\text{by (14)}\} \\
& L/u_i \\
\ge {} & \{\text{by (13)}\} \\
& \mathsf{lag}(\tau_i, t, \mathcal{S})/u_i.
\end{aligned}$$

That is, $t - \mathsf{lag}(\tau_i, t, \mathcal{S})/u_i > d_i(t)$, which contradicts Lemma 2. □

Recall that if $\mathsf{lag}(\tau_i, t, \mathcal{S})$ is negative, then τ_i is over-allocated in schedule $\mathcal{S}$ compared to schedule $\mathcal{I}$. However, the actual schedule $\mathcal{S}$ cannot execute jobs that are not released and therefore can never get more than a full job "ahead" of $\mathcal{I}$. Thus, we have the following trivial lower bound[5] on $\mathsf{lag}(\tau_i, t, \mathcal{S})$, which we state without proof.

Lemma 3. $\mathsf{lag}(\tau_i, t, \mathcal{S}) \ge -C_{\max}$.

The following lemma uses the relationship between effective deadlines and lag values established in Lemma 2 to obtain a sufficient lag-based condition for one task to have an earlier effective deadline than another.

Lemma 4. *If tasks* τ_i *and* τ_k *are both pending at time* t*, and if*

$$\mathsf{lag}(\tau_i, t, \mathcal{S}) \ge \frac{u_i}{u_k} \cdot \mathsf{lag}(\tau_k, t, \mathcal{S}) + C_i \tag{15}$$

holds, then $d_i(t) < d_k(t)$*.*

Proof.

$$\begin{aligned}
d_i(t) \le {} & \{\text{by Lemma 2}\} \\
& t - \frac{\mathsf{lag}(\tau_i, t, \mathcal{S})}{u_i} + T_i \\
\le {} & \{\text{by (15)}\} \\
& t - \frac{\frac{u_i}{u_k} \cdot \mathsf{lag}(\tau_k, t, \mathcal{S}) + C_i}{u_i} + T_i \\
= {} & \{\text{canceling } u_i \text{ and using } C_i/u_i = T_i\} \\
& t - \frac{\mathsf{lag}(\tau_k, t, \mathcal{S})}{u_k} \\
< {} & \{\text{by Lemma 2}\} \\
& d_k(t)
\end{aligned}$$

The lemma follows. □

Corollary 2. *If tasks* τ_i *and* τ_k *are both pending at time* t *and if* $\mathsf{lag}(\tau_i, t, \mathcal{S}) \ge \rho \cdot \mathsf{lag}(\tau_k, t, \mathcal{S}) + C_{\max}$ *holds, where* $\rho = u_1/u_n$*, then* $d_i(t) < d_k(t)$*.*

Proof. Because tasks are indexed from highest utilization to lowest, $\mathsf{lag}(\tau_i, t, \mathcal{S}) \ge \rho \cdot \mathsf{lag}(\tau_k, t, \mathcal{S}) + C_{\max} = \frac{u_1}{u_n} \cdot \mathsf{lag}(\tau_k, t, \mathcal{S}) + C_{\max} \ge \frac{u_i}{u_k} \cdot \mathsf{lag}(\tau_k, t, \mathcal{S}) + C_i$. By Lemma 4, the corollary follows. □

[3]Without Assumption (A), $J_{i,j}$ may execute in total for less than its worst-case execution requirement of C_i, and therefore only "$\le$" can be claimed in (10).

[4]Without Assumption (A), $J_{i,j}$ might not execute "continuously," so only "$\le$" can be claimed in (11).

[5]A tighter bound is possible, but this simple bound is sufficient for our purposes.

Proof strategies for deriving tardiness bounds. Given the relationships established above between lag values and deadlines, we are now ready to derive tardiness bounds. According to Corollary 1, lag bounds directly imply tardiness bounds. Thus, one natural strategy is to attempt to derive n individual lag bounds, one per task. However, we were unable to make this strategy work. Intuitively, this is because, in deriving n individual per-task lag bounds, we must consider how all tasks interact as they are scheduled together. When doing this, it is difficult to avoid a case explosion that causes the entire proof to collapse. In particular, the feasibility condition given by (6) and (7) must ultimately be exploited in the proof. Every attempt we made in deriving per-task lag bounds resulted in a case explosion that was so unwieldy, we could not discern how (6) and (7) could possibly factor into the proof.

Notice that (6) simply requires that the platform is not over-utilized, which is something required in reasoning about identical platforms as well. The constraints in (7), however, are unique to the uniform case. Observe that these constraints reference the sum of the k largest utilizations and speeds. Accordingly, we switched from working on proof strategies that focus on per-task lag bounds to one that focuses on the sum of the k largest lag values.

Our proof strategy, formally explained. In order to describe this proof strategy more formally, we let $\hat{\tau}_\ell(t)$ denote the task that has the ℓ^{th} largest lag at time instant t, with ties broken arbitrarily. We also denote the ℓ^{th} largest lag at time instant t as $\mathsf{Lag}_l(t)$, *i.e.*, $\mathsf{Lag}_l(t) = \mathsf{lag}(\hat{\tau}_l(t), t, \mathcal{S})$. Furthermore, we let $\mathcal{T}_\ell(t)$ denote the set of tasks corresponding to $\mathsf{Lag}_1(t), \mathsf{Lag}_2(t), \ldots, \mathsf{Lag}_\ell(t)$, *i.e.*, $\mathcal{T}_\ell(t) = \{\hat{\tau}_1(t), \hat{\tau}_2(t), \ldots, \hat{\tau}_\ell(t)\}$.

To derive tardiness bounds, we show that the following $m+1$ inequalities, (B$_1$), …, (B$_m$), and (B$_n$), hold at any time t.

Inequality Set (B):

$$\mathsf{Lag}_1(t) \le \beta_1 \tag{B$_1$}$$
$$\mathsf{Lag}_1(t) + \mathsf{Lag}_2(t) \le \beta_2 \tag{B$_2$}$$
$$\mathsf{Lag}_1(t) + \mathsf{Lag}_2(t) + \mathsf{Lag}_3(t) \le \beta_3 \tag{B$_3$}$$
$$\vdots \qquad\qquad \vdots$$
$$\mathsf{Lag}_1(t) + \mathsf{Lag}_2(t) + \cdots + \mathsf{Lag}_k(t) \le \beta_k \tag{B$_k$}$$
$$\vdots \qquad\qquad \vdots$$
$$\mathsf{Lag}_1(t) + \mathsf{Lag}_2(t) + \cdots + \mathsf{Lag}_m(t) \le \beta_m \tag{B$_m$}$$
$$\mathsf{Lag}_1(t) + \mathsf{Lag}_2(t) + \cdots + \mathsf{Lag}_n(t) \le \beta_n \tag{B$_n$}$$

If all constraints in the inequality set (B) hold at all time instants t, where $\beta_1, \beta_2, \ldots, \beta_m$, and β_n are constants (which will depend on task-set parameters), then, by the definition of $\mathsf{Lag}_1(t)$, β_1 is an upper bound on $\mathsf{lag}(\tau_i, t, \mathcal{S})$ for any i and for any t. Given such an upper bound, by Corollary 1, tardiness bounds will follow.

In order to prove that the constraints in (B) hold at all time instants t, we must carefully define $\beta_1, \beta_2, \ldots, \beta_m$, and β_n. They are defined as follows.

$$\beta_1 = x_1 \tag{X$_1$}$$
$$\beta_2 = \beta_1 + x_2 \tag{X$_2$}$$
$$\beta_3 = \beta_2 + x_3 \tag{X$_3$}$$
$$\vdots \qquad\qquad \vdots$$
$$\beta_k = \beta_{k-1} + x_k \tag{X$_k$}$$
$$\vdots \qquad\qquad \vdots$$
$$\beta_m = \beta_{m-1} + x_m \tag{X$_m$}$$
$$\beta_n = \beta_m + x_n \tag{X$_n$}$$

where

$$x_n = -(n-m-1) \cdot C_{\max} \tag{Y$_1$}$$
$$x_m = (n-m+1) \cdot C_{\max} \tag{Y$_2$}$$
$$x_i = \rho \cdot x_{i+1} + C_{\max}, \text{ for } i = m-1, m-2, \ldots, 1 \tag{Y$_3$}$$

Note that, in (Y$_3$), $\rho = u_1/u_n$.

To see that $\beta_1, \beta_2, \ldots, \beta_m$, and β_n are well-defined, observe that x_n and x_m can be directly calculated by (Y$_1$) and (Y$_2$) for any given task set. Then, $x_{m-1}, x_{m-2}, \ldots, x_1$ can be calculated inductively by (Y$_3$). Finally, $\beta_1, \beta_2, \ldots, \beta_m$, and β_n can be calculated by (X$_1$), …, (X$_m$), and (X$_n$).

Formal derivation of tardiness bounds. Having set up our proof strategy, we next present a critical mathematical property of the lag and Lag functions when they are viewed as a function of t.

Property 1. *For a given task τ_i and a given schedule $\mathcal{S}$, $\mathsf{lag}(\tau_i, t, \mathcal{S})$ is a continuous function of t. For a given schedule $\mathcal{S}$, $\mathsf{Lag}_\ell(t)$ is a continuous function of t for each ℓ.*

Proof. $\mathsf{lag}(\tau_i, t, \mathcal{S})$ is a continuous function of t because, by (4), $\mathsf{lag}(\tau_i, t, \mathcal{S}) = \mathsf{A}(\mathcal{I}, \tau_i, 0, t) - \mathsf{A}(\mathcal{S}, \tau_i, 0, t)$, and $\mathsf{A}(\mathcal{I}, \tau_i, 0, t)$ and $\mathsf{A}(\mathcal{S}, \tau_i, 0, t)$ are both (clearly) continuous functions of t. Furthermore, since taking the maximum value of a set of continuous functions is also a continuous function, $\mathsf{Lag}_1(t)$ is a continuous function of t. For similar reasons, $\mathsf{Lag}_2(t)$, $\mathsf{Lag}_3(t)$, …, $\mathsf{Lag}_n(t)$ are all continuous functions of t as well. □

We are now ready to prove our main theorem.

Theorem 1. *At every time instant $t \ge 0$, each inequality in the set (B) holds.*

Proof. Suppose, to the contrary, that the statement of the theorem is not true, and let t_c denote the first time instant such that *any* inequality in (B) is false. We show that the existence of t_c leads to a contradiction.

Claim 1. $t_c > 0$.

Proof. It follows by induction using (Y$_2$) and (Y$_3$) (and our assumption from Sec. 2 that $n \ge m$) that $x_i > 0$ for $i = 1, 2, \ldots, m$. By induction again, this time using (X$_1$), …, (X$_m$), it further follows that $\beta_i > 0$ for $i = 1, 2, \ldots, m$. Finally, by (X$_m$), (X$_n$), (Y$_1$), and (Y$_2$), $\beta_n = \beta_{m-1} + 2C_{\max} > 0$. Thus, because $\mathsf{Lag}_i(0) = 0$ holds for all i, all of the inequalities in (B) are true at time 0, implying that $t_c > 0$. $\square$

Let $t_c^- = t_c - \varepsilon$, where $\varepsilon \to 0^+$.[6] By Claim 1, $t_c^- \geq 0$, *i.e.*, t_c^- is well-defined. Because t_c is the *first* time instant at which *any* inequality in (B) is false, *all* such inequalities hold prior to t_c, including at time t_c^-. Also, because the length of the interval $[t_c^-, t_c)$ is arbitrarily small, a task scheduled on a processor at time t_c^- will be continuously scheduled within $[t_c^-, t_c)$.

We call an inequality in (B) *critical* if and only if it is false at time t_c. If (B$_k$) is critical, then $\mathsf{Lag}_1(t_c^-) + \cdots + \mathsf{Lag}_k(t_c^-) = \beta_k$. This is because (B$_k$) holds for any time instant before t_c but is falsified at t_c and the left-hand-side of (B$_k$) is a continuous function of t, by Property 1.[7] We now consider two cases, which depend on which inequalities are critical.

Case 1: (B$_n$) is critical. In this case,

$$\mathsf{Lag}_1(t_c^-) + \mathsf{Lag}_2(t_c^-) + \cdots + \mathsf{Lag}_n(t_c^-) = \beta_n. \qquad (16)$$

Therefore,

$$\begin{aligned}
&\mathsf{Lag}_m(t_c^-) + \mathsf{Lag}_{m+1}(t_c^-) + \cdots + \mathsf{Lag}_n(t_c^-)\\
=\ &\{\text{by (16)}\}\\
&\beta_n - (\mathsf{Lag}_1(t_c^-) + \mathsf{Lag}_2(t_c^-) + \cdots + \mathsf{Lag}_{m-1}(t_c^-))\\
\geq\ &\{\text{because (B}_{m-1}\text{) holds at time } t_c^-\}\\
&\beta_n - \beta_{m-1}\\
=\ &\{\text{by (X}_m\text{), (X}_n\text{), (Y}_1\text{), and (Y}_2\text{)}\}\\
&2C_{\max}. \qquad (17)
\end{aligned}$$

Furthermore, by definition, $\mathsf{Lag}_m(t_c^-) \geq \mathsf{Lag}_{m+1}(t_c^-) \geq \cdots \geq \mathsf{Lag}_n(t_c^-)$, so $\mathsf{Lag}_m(t_c^-)$ is at least the average of these $n - m + 1$ values. Therefore,

$$\begin{aligned}
\mathsf{Lag}_m(t_c^-) \geq\ &\frac{\mathsf{Lag}_m(t_c^-) + \mathsf{Lag}_{m+1}(t_c^-) + \cdots + \mathsf{Lag}_n(t_c^-)}{n - m + 1}\\
\geq\ &\{\text{by (17)}\}\\
&\frac{2C_{\max}}{n - m + 1}\\
>\ &\{\text{because } C_{\max} > 0 \text{ and } n \geq m\}\\
&0. \qquad (18)
\end{aligned}$$

Because $\mathsf{Lag}_m(t_c^-)$ denotes the m^{th} largest lag at time t_c^-, (18) implies that, at time t_c^-, at least m tasks have positive lag. Thus, by Lemma 1, at least m tasks are pending at time t_c^-. Therefore, all of the m processors are busy during the time interval $[t_c^-, t_c)$. Thus, by (6), the total lag in the system does not increase during the interval $[t_c^-, t_c)$. That is, $\mathsf{Lag}_1(t_c) + \mathsf{Lag}_2(t_c) + \cdots + \mathsf{Lag}_n(t_c) = \sum_{i=1}^{n} \mathsf{lag}(\tau_i, t_c, \mathcal{S}) \leq \sum_{i=1}^{n} \mathsf{lag}(\tau_i, t_c^-, \mathcal{S}) = \mathsf{Lag}_1(t_c^-) + \mathsf{Lag}_2(t_c^-) + \cdots + \mathsf{Lag}_n(t_c^-)$, which by (16), implies

$$\mathsf{Lag}_1(t_c) + \mathsf{Lag}_2(t_c) + \cdots + \mathsf{Lag}_n(t_c) \leq \beta_n.$$

This contradicts the assumption of Case 1 that (B$_n$) is critical.

Case 2: (B$_k$) is critical for some k such that $1 \leq k \leq m$. In this case,

$$\mathsf{Lag}_1(t_c^-) + \mathsf{Lag}_2(t_c^-) + \cdots + \mathsf{Lag}_k(t_c^-) = \beta_k. \qquad (19)$$

Our proof for Case 2 utilizes a number of claims, which we prove in turn.

Claim 2. $\mathsf{Lag}_k(t_c^-) \geq x_k$.

Proof. If $k = 1$, then by (19), $\mathsf{Lag}_1(t_c^-) = \beta_1$. Also, by (X$_1$), $\beta_1 = x_1$, from which $\mathsf{Lag}_1(t_c^-) \geq x_1$ follows. The remaining possibility, $2 \leq k \leq m$, is addressed as follows.

$$\begin{aligned}
&\mathsf{Lag}_k(t_c^-)\\
=\ &\{\text{by (19)}\}\\
&\beta_k - (\mathsf{Lag}_1(t_c^-) + \mathsf{Lag}_2(t_c^-) + \cdots + \mathsf{Lag}_{k-1}(t_c^-))\\
\geq\ &\{\text{since (B}_{k-1}\text{) holds at time } t_c^-\}\\
&\beta_k - \beta_{k-1}\\
=\ &\{\text{by (X}_k\text{)}\}\\
&x_k
\end{aligned}$$

$\square$

Claim 3. *If* $k \leq m - 1$, *then* $\mathsf{Lag}_{k+1}(t_c^-) \leq x_{k+1}$.

Proof. Because $k \leq m - 1$ and (by assumption) $n \geq m$, $\mathsf{Lag}_{k+1}(t_c^-)$ is well-defined. The claim is established by the following reasoning.

$$\begin{aligned}
&\mathsf{Lag}_{k+1}(t_c^-)\\
\leq\ &\{\text{because (B}_{k+1}\text{) holds at time } t_c^-\}\\
&\beta_{k+1} - (\mathsf{Lag}_1(t_c^-) + \mathsf{Lag}_2(t_c^-) + \cdots + \mathsf{Lag}_k(t_c^-))\\
=\ &\{\text{by (19)}\}\\
&\beta_{k+1} - \beta_k\\
=\ &\{\text{by (X}_{k+1}\text{)}\}\\
&x_{k+1}
\end{aligned}$$

$\square$

Claim 4. *If* $k = m$ *and* $n > m$, *then* $\mathsf{Lag}_{k+1}(t_c^-) \leq 0$.

Proof. $k = m$ implies that (19) can be re-written as

$$\mathsf{Lag}_1(t_c^-) + \mathsf{Lag}_2(t_c^-) + \cdots + \mathsf{Lag}_m(t_c^-) = \beta_m. \quad (20)$$

[6] ε does not have to be infinitely close to 0. Instead, it only needs to be a sufficiently small positive constant. However, the criteria for "sufficiently small" are rather tedious, so we merely define $\varepsilon \to 0^+$ here for simplicity. Whenever ε is used, we will further elaborate on its definition in that context.

[7] If $\mathsf{Lag}_1(t_c^-) + \cdots + \mathsf{Lag}_k(t_c^-) < \beta_k$, then a time $t \in [t_c^-, t_c)$ must exist such that $\mathsf{Lag}_1(t) + \cdots + \mathsf{Lag}_k(t) = \beta_k$. Therefore, a smaller ε could have been selected so that $t_c^- = t$.

Also, because (B_n) holds at time t_c^-,

$$\mathsf{Lag}_1(t_c^-)+\mathsf{Lag}_2(t_c^-)+\cdots+\mathsf{Lag}_n(t_c^-)\leq\beta_n. \quad (21)$$

Therefore, given $n > m$ (from the statement of the claim), by (20) and (21), we have

$$\mathsf{Lag}_{m+1}(t_c^-)+\mathsf{Lag}_{m+2}(t_c^-)+ \cdots+\mathsf{Lag}_n(t_c^-)\leq\beta_n-\beta_m. \quad (22)$$

Therefore,

$$\begin{aligned}
&\mathsf{Lag}_{m+1}(t_c^-)\\
\leq\ &\{\text{by (22)}\}\\
&\beta_n-\beta_m-(\mathsf{Lag}_{m+2}(t_c^-)+\cdots+\mathsf{Lag}_n(t_c^-))\\
\leq\ &\{\text{by Lemma 3}\}\\
&\beta_n-\beta_m-((-C_{\max})\cdot(n-m-1))\\
=\ &\{\text{by } (\mathrm{X}_n) \text{ and } (\mathrm{Y}_1)\}\\
&-(n-m-1)C_{\max}+(n-m-1)C_{\max}\\
=\ &\{\text{canceling}\}\\
&0. \qquad (23)
\end{aligned}$$

Because $k = m$, the claim follows from (23). □

Claim 5. *If $k \leq m-1$ or $n > m$, then* $\mathsf{Lag}_k(t_c^-) \geq \rho\cdot\mathsf{Lag}_{k+1}(t_c^-)+C_{\max}$.

Proof. If $k \leq m-1$, then

$$\begin{aligned}
\mathsf{Lag}_k(t_c^-)\geq\ &\{\text{by Claim 2}\}\\
&x_k\\
=\ &\{\text{by } (\mathrm{Y}_3)\}\\
&\rho\cdot x_{k+1}+C_{\max}\\
\geq\ &\{\text{by Claim 3}\}\\
&\rho\cdot\mathsf{Lag}_{k+1}(t_c^-)+C_{\max}. \qquad (24)
\end{aligned}$$

If $k = m$ (recall that $k \leq m$ by the specification of Case 2) and $n > m$, then by Claim 4,

$$\mathsf{Lag}_{m+1}(t_c^-)\leq 0. \quad (25)$$

Furthermore,

$$\begin{aligned}
\mathsf{Lag}_m(t_c^-)\geq\ &\{\text{by Claim 2}\}\\
&x_m\\
=\ &\{\text{by } (\mathrm{Y}_2)\}\\
&(n-m+1)C_{\max}\\
>\ &\{\text{because } C_{\max}>0 \text{ and } n>m\}\\
&C_{max}\\
\geq\ &\{\text{by (25), and because } \rho=u_1/u_n>0\}\\
&\rho\cdot\mathsf{Lag}_{m+1}(t_c^-)+C_{\max}. \qquad (26)
\end{aligned}$$

Thus, by (24) and (26) the claim follows. □

In considering the next claim, recall the following: at time t, $\mathsf{Lag}_\ell(t)$ denotes the ℓ^{th} largest lag among all n tasks and $\mathcal{T}_\ell(t)$ denotes the set of ℓ tasks with largest lag values.

Claim 6. *The k tasks in $\mathcal{T}_k(t_c^-)$ are scheduled on the k fastest processors within $[t_c^-, t_c)$.*

Proof. Because $\rho = u_1/u_n \geq 1$ and (by assumption) $n \geq m$, by (Y_2) and (Y_3), it can be shown that $x_k \geq C_{\max}$ for $1 \leq k \leq m$. Therefore, by Claim 2, we have $\mathsf{Lag}_k(t_c^-) \geq x_k \geq C_{\max} > 0$. By Lemma 1, this implies that each of the k tasks in $\mathcal{T}_k(t_c^-)$ is pending at time t_c^-. Therefore, by Policy (G), it suffices to prove that the k tasks in $\mathcal{T}_k(t_c^-)$ have the k earliest effective deadlines at time t_c^- (with no tie with the $(k+1)^{st}$ earliest effective deadline).[8] If $k = m$ and $n = m$, then there are k tasks in total in the system. In this case, the k pending tasks in $\mathcal{T}_k(t_c^-)$ clearly have the k earliest effective deadlines at time t_c^-.

In the rest of the proof, we consider the remaining possibility, *i.e.*, $k \leq m-1$ or $n > m$. By Claim 5,

$$\mathsf{Lag}_k(t_c^-)\geq\rho\cdot\mathsf{Lag}_{k+1}(t_c^-)+C_{\max}. \quad (27)$$

By the definition of Lag, (27) implies that for any i and j such that $1 \leq i \leq k$ and $k+1 \leq j \leq n$, we have $\mathsf{Lag}_i(t_c^-) \geq \rho\cdot\mathsf{Lag}_j(t_c^-)+C_{\max}$. This implies that, for any task $\tau_p \in \mathcal{T}_k(t_c^-)$ and any task $\tau_q \notin \mathcal{T}_k(t_c^-)$, $\mathsf{lag}(\tau_p, t_c^-, \mathcal{S}) \geq \rho\cdot\mathsf{lag}(\tau_q, t_c^-, \mathcal{S})+C_{\max}$. Therefore, if a task $\tau_q \notin \mathcal{T}_k(t_c^-)$ is pending at time t_c^-, then by Lemma 4, its effective deadline is strictly greater than the effective deadline of any task $\tau_p \in \mathcal{T}_k(t_c^-)$; if τ_q is not pending at time t_c^-, then it has no pending jobs and no effective deadline by definition. Therefore, the k tasks in $\mathcal{T}_k(t_c^-)$ have the k earliest effective deadlines at time t_c^-. The claim follows. □

Claim 7. $\mathcal{T}_k(t_c^-) = \mathcal{T}_k(t_c)$.

Proof. If $k = m$ and $n = m$, then there are k tasks in total in the system, so the claim clearly holds. Therefore, in the rest of the proof, we assume $k \leq m-1$ or $n > m$, which implies that either $k \leq m-1$ holds or $k = m$ and $n > m$ hold, by the specification of Case 2. If $k \leq m-1$, then

$$\begin{aligned}
\mathsf{Lag}_k(t_c^-)\geq\ &\{\text{by Claim 2}\}\\
&x_k\\
=\ &\{\text{by } (\mathrm{Y}_3)\}\\
&\rho\cdot x_{k+1}+C_{max}\\
\geq\ &\{\text{because } \rho=u_1/u_n\geq 1 \text{ and } x_{k+1}\geq 0\}\\
&x_{k+1}+C_{\max}\\
\geq\ &\{\text{by Claim 3}\}\\
&\mathsf{Lag}_{k+1}(t_c^-)+C_{\max}. \qquad (28)
\end{aligned}$$

[8]Note that ε can be selected to be small enough to ensure that no scheduling event happens within the interval $[t_c^-, t_c)$, including job completions. Therefore, any task scheduled at time t_c^- will continuously execute during time interval $[t_c^-, t_c)$ on the same processor.

If $k = m$ and $n > m$, then

$$\begin{aligned}
\mathsf{Lag}_k(t_c^-) \geq\ & \{\text{by Claim 2 and because } k = m\} \\
& x_m \\
=\ & \{\text{by } (\text{Y}_2)\} \\
& (n - m + 1)C_{max} \\
>\ & \{\text{because } n > m\} \\
& C_{\max} \\
\geq\ & \{\text{because } \mathsf{Lag}_{k+1}(t_c^-) \leq 0, \text{ by Claim 4}\} \\
& \mathsf{Lag}_{k+1}(t_c^-) + C_{\max}. \qquad (29)
\end{aligned}$$

Thus, for $k \leq m - 1$ or $n > m$, by (28) and (29), we have

$$\mathsf{Lag}_k(t_c^-) - \mathsf{Lag}_{k+1}(t_c^-) \geq C_{\max}. \qquad (30)$$

By the definition of Lag, (30) implies that for any i and j such that $1 \leq i \leq k$ and $k+1 \leq j \leq n$, we have $\mathsf{Lag}_i(t_c^-) - \mathsf{Lag}_j(t_c^-) \geq C_{\max}$. This implies that, for any task $\tau_p \in \mathcal{T}_k(t_c^-)$ and any task $\tau_q \notin \mathcal{T}_k(t_c^-)$, we have

$$\mathsf{lag}(\tau_p, t_c^-, \mathcal{S}) - \mathsf{lag}(\tau_q, t_c^-, \mathcal{S}) \geq C_{\max}. \qquad (31)$$

Therefore,

$$\begin{aligned}
& \mathsf{lag}(\tau_p, t_c, \mathcal{S}) - \mathsf{lag}(\tau_q, t_c, \mathcal{S}) \\
=\ & \{\text{by } (5)\} \\
& \mathsf{lag}(\tau_p, t_c^-, \mathcal{S}) + \mathsf{A}(\mathcal{I}, \tau_p, t_c^-, t_c) - \mathsf{A}(\mathcal{S}, \tau_p, t_c^-, t_c) \\
& - \mathsf{lag}(\tau_q, t_c^-, \mathcal{S}) - \mathsf{A}(\mathcal{I}, \tau_q, t_c^-, t_c) + \mathsf{A}(\mathcal{S}, \tau_q, t_c^-, t_c) \\
\geq\ & \{\text{by (1) and (2)}\} \\
& \mathsf{lag}(\tau_p, t_c^-, \mathcal{S}) + 0 - \varepsilon \cdot s_1 - \mathsf{lag}(\tau_q, t_c^-, \mathcal{S}) - \varepsilon \cdot u_1 + 0 \\
\geq\ & \{\text{by (31) and rearranging}\} \\
& C_{\max} - \varepsilon \cdot (s_1 + u_1) \\
>\ & \{\text{because } C_{\max} > 0 \text{ and } \varepsilon < C_{\max}/(s_1 + u_1)\}^9 \\
& 0.
\end{aligned}$$

Thus, at time t_c, any task in $\mathcal{T}_k(t_c^-)$ has a strictly greater lag than any task not in $\mathcal{T}_k(t_c^-)$. This implies that $\mathcal{T}_k(t_c)$ and $\mathcal{T}_k(t_c^-)$ consist of the same set of tasks. □

By Claim 6 and Claim 7, the k tasks in $\mathcal{T}_k(t_c)$ are continuously scheduled on the k fastest processor during time interval $[t_c^-, t_c)$, so

$$\sum_{\tau_i \in \mathcal{T}_k(t_c)} \mathsf{A}(\mathcal{S}, \tau_i, t_c^-, t_c) = S_k \cdot \varepsilon. \qquad (32)$$

[9] ε can be selected small enough to ensure $\varepsilon < C_{\max}/(s_1 + u_1)$.

Also, by (2),

$$\sum_{\tau_i \in \mathcal{T}_k(t_c)} \mathsf{A}(\mathcal{I}, \tau_i, t_c^-, t_c) \leq \sum_{\tau_i \in \mathcal{T}_k(t_c)} u_i \cdot \varepsilon \leq U_k \cdot \varepsilon. \qquad (33)$$

Therefore,

$$\begin{aligned}
& \mathsf{Lag}_1(t_c) + \mathsf{Lag}_2(t_c) + \cdots + \mathsf{Lag}_k(t_c) \\
=\ & \{\text{by Claim 7 and by (5)}\} \\
& \mathsf{Lag}_1(t_c^-) + \mathsf{Lag}_2(t_c^-) + \cdots + \mathsf{Lag}_k(t_c^-) + \\
& \sum_{\tau_i \in \mathcal{T}_k(t_c)} \mathsf{A}(\mathcal{I}, \tau_i, t_c^-, t_c) - \sum_{\tau_i \in \mathcal{T}_k(t_c)} \mathsf{A}(\mathcal{S}, \tau_i, t_c^-, t_c) \\
\leq\ & \{\text{by (32) and (33), and because } (\text{B}_k) \text{ holds at time } t_c^-\} \\
& \beta_k + (U_k - S_k) \cdot \varepsilon \\
\leq\ & \{\text{by (6) and (7); note that } U_m \leq U_n\} \\
& \beta_k.
\end{aligned}$$

This contradicts the assumption of Case 2 that (B_k) is critical.

Finishing up. We have shown that both Case 1 and Case 2 lead to a contradiction. That is, none of the conditions $(\text{B}_1), \ldots, (\text{B}_m)$, or (B_n) is critical at t_c. This contradicts the definition of t_c as the first time instant at which some inequality in (B) is false, *i.e.*, such a t_c does not exist. The theorem follows. □

Using Theorem 1, we can easily derive a tardiness bound for every task as follows.

Theorem 2. *In $\mathcal{S}$, the tardiness Δ_i of task τ_i is bounded as follows.*

If $\rho = 1$, then $\Delta_i \leq \dfrac{n \cdot C_{\max}}{u_i}$;

if $\rho > 1$, then

$$\Delta_i \leq \frac{\rho^{m-1} \cdot (n - m + 1)C_{\max} + \frac{\rho^{m-1} - 1}{\rho - 1} \cdot C_{\max}}{u_i}.$$

Proof. By Theorem 1, $\mathsf{Lag}_1(t) \leq \beta_1$ holds at every time instant t. By the definition of $\mathsf{Lag}_1(t)$, this implies that, for each i, $\mathsf{lag}(\tau_i, t, \mathcal{S}) \leq \beta_1$ holds for all t. Thus, by Corollary 1, task τ_i has a tardiness bound of β_1/u_i. By (X_1), $\beta_1 = x_1$, so to complete the proof, we merely need to calculate x_1.

If $\rho = 1$ (*i.e.*, $u_1/u_n = 1$, which implies that every task has the same utilization), then by (Y_3), $x_1 = x_m + (m - 1)C_{\max}$. Thus, by (Y_2), we have $x_1 = n \cdot C_{\max}$.

If $\rho > 1$, then rearranging (Y_3) results in

$$x_i + \frac{C_{\max}}{\rho - 1} = \rho \left(x_{i+1} + \frac{C_{\max}}{\rho - 1} \right).$$

By iterating this recurrence, we have

$$x_1 + \frac{C_{\max}}{\rho - 1} = \rho^{m-1} \cdot \left(x_m + \frac{C_{\max}}{\rho - 1} \right).$$

Finally, applying (Y_2) yields

$$x_1 = \rho^{m-1} \cdot (n - m + 1)C_{\max} + \frac{\rho^{m-1} - 1}{\rho - 1} \cdot C_{\max}.$$

The theorem follows. □

Discussion. We comment here on several aspects of the tardiness bounds in Theorem 2.

First, it may seem counterintuitive that u_i appears in the denominator of both stated bounds, because this means that the bounds are greater for tasks of lower utilization. This effect follows from our proof strategy, which involved bounding the maximum lag value in the system. We allow that any task may be the one that achieves this maximum value. Given a fixed value of lag, a lower-utilization task will tend to require greater tardiness to achieve that value of lag.

Second, the term ρ^{m-1} appears in the second bound, *i.e.*, the bound is exponential with respect to the processor count m. Despite this deficiency, we have still succeeded in establishing for the first time that tardiness under GEDF is bounded on uniform platforms. Moreover, global schedulers tend to be practical in practice (*i.e.*, have reasonable overheads) only for modest processor counts [2], so typically on large multiprocessor platforms, such schedulers are applied within clusters of processors that are not too large. In such a setting, our bounds may be reasonable.

Third, the term $\rho = u_1/u_n$ appears in the second bound. This bound could therefore be rather large if $u_1 \gg u_n$. Under a clustered-scheduling approach, this issue could be eased by using heuristics to assign tasks with relatively similar utilizations to the same cluster. Also, it might be possible to encapsulate several ordinary tasks into a higher-utilization "container" task in a way that increases u_n. However, such an approach would warrant further study.

Extending to sporadic tasks. In this section, we have limited attention to periodic task systems satisfying Assumption (A). However, it is possible to eliminate the need for Assumption (A) and extend the results of this section to apply to sporadic task systems. Due to space constraints, we consider these issues in an appendix.

4 Conclusion

In this paper, we have shown that the GEDF scheduler is SRT optimal on uniform multiprocessor platforms, closing a longstanding open problem. Together with prior results establishing the *non*-optimality of NP-GEDF in this context [16], we now have a complete picture concerning the SRT optimality of both preemptive and non-preemptive GEDF on uniform platforms.

We established the SRT optimality of GEDF in the considered context by deriving tardiness bounds for any sporadic task system that is feasible according to the constraints specified in (6) and (7). The key to this optimality result proved to be finding a way to work these constraints into the argument. The need to exploit the constraints in (7) in particular resulted in the necessity for a proof strategy that is very different from that which had been employed previously in similar work pertaining to identical platforms [5, 12].

The establishment of tardiness bounds under GEDF on identical platforms [5] led to follow-up work in which such bounds were derived for a broader class of schedulers (particularly, "window-constrained" schedulers) [12] and for schedulers that must function in settings where processor supply is only partially available [11]. In future work, we plan to consider similar extensions to the analysis presented in this paper. We also plan to investigate whether tighter tardiness bounds can be obtained generally and whether such bounds can be further honed if processors of only two different speeds exist.

Acknowledgement: In the three years we have been working on the open problem addressed herein, we have discussed it with many people. Discussions with Zhishan Guo and Rui Liu were particularly helpful.

References

[1] S. Baruah and J. Goossens. Rate-monotonic scheduling on uniform multiprocessors. *IEEE Transactions on Computers*, 52(7):966–970, 2003.

[2] B. Brandenburg. *Scheduling and Locking in Multiprocessor Real-Time Operating Systems*. PhD thesis, University of North Carolina, Chapel Hill, NC, 2011.

[3] S. Chen and C. Hsueh. Optimal dynamic-priority real-time scheduling algorithms for uniform multiprocessors. In *29th RTSS*, 2008.

[4] L. Cucu and J. Goossens. Feasibility intervals for fixed-priority real-time scheduling on uniform multiprocessors. In *11th ETFA*, 2006.

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

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

[7] S. Funk. *EDF Scheduling on Heterogeneous Multiprocessors*. PhD thesis, University of North Carolina, Chapel Hill, NC, 2004.

[8] S. Funk, J. Goossens, and S. Baruah. On-line scheduling on uniform multiprocessors. In *22nd RTSS*, 2001.

[9] S. Funk and V. Nanadur. LRE-TL: An optimal multiprocessor scheduling algorithm for sporadic task sets. In *17th RTNS*, 2009.

[10] E. Horvath, S. Lam, and R. Sethi. A level algorithm for preemptive scheduling. *Journal of the ACM*, 24(1):3243, 1977.

[11] H. Leontyev and J. Anderson. A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees. In *20th ECRTS*.

[12] H. Leontyev and J. Anderson. Generalized tardiness bounds for global multiprocessor scheduling. *Real-Time Systems*, 44(1):26–71, 2010.

[13] M. Pinedo. *Scheduling, Theory, Algorithms, and Systems*. Prentice Hall, 1995.

[14] G. Tong and C. Liu. Supporting soft real-time sporadic task systems on heterogeneous multiprocessors with no uilitzation loss. *IEEE Transactions on Parallel and Distributed Systems*, 27(9):2740–2752, 2016.

[15] K. Yang and J. Anderson. Optimal GEDF-based schedulers that allow intra-task parallelism on heterogeneous multiprocessors. In *12th ESTIMedia*, 2014.

[16] K. Yang and J. Anderson. On the soft real-time optimality of global EDF on multiprocessors: From identical to uniform heterogeneous. In *21st RTCSA*, 2015.

[17] K. Yang and J. Anderson. An optimal semi-partitioned scheduler for uniform heterogeneous multiprocessors. In *27th ECRTS*, 2015.

[18] K. Yang and J. Anderson. Tardiness bounds for global EDF scheduling on a uniform multiprocessor. In *7th RTSOPS*, 2016.

Appendix A: Extending to Sporadic Tasks

In this appendix, we show that the results of Sec. 3 can be extended to sporadic task systems. We begin by showing that Assumption (A) can be removed for periodic task systems.

Theorem 3. *For a given periodic task set τ, let $\mathcal{S}$ denote a GEDF schedule that satisfies* (A)*, and let $\mathcal{S}'$ denote a corresponding GEDF schedule where* (A) *does not hold ("corresponding" means that $\mathcal{S}$ and $\mathcal{S}'$ include exactly the same jobs, released at exactly the same time instants). Then, no job finishes later in $\mathcal{S}'$.*

Proof. We prove the theorem by considering jobs inductively in deadline order. (We assume that deadline ties are broken the same way in both schedules.) Note that, by Policy (G), the scheduling of a given job is not impacted by any lower priority jobs.

Base case. The highest-priority job will execute continuously on the fastest processor once released, so it cannot finish later in $\mathcal{S}'$.

Inductive step. Let $\mathcal{J}$ denote the set of k highest-priority jobs, and assume that these jobs do not finish later in $\mathcal{S}'$. Also, let J denote the $(k+1)^{st}$ highest-priority job. We show that J also does not finish later in $\mathcal{S}'$.

Since no job in $\mathcal{J}$ finishes later in $\mathcal{S}'$, at any time instant t, the number of *ready* jobs in $\mathcal{S}'$ does not increase[10] compared to $\mathcal{S}$. Therefore, by Policy (G), up to any time instant t after its release, J is allocated in $\mathcal{S}'$ no less computing capacity than in $\mathcal{S}$, unless J has completed in $\mathcal{S}'$ but not in $\mathcal{S}$ prior to time t. Finally, because J's execution requirement is no greater in $\mathcal{S}'$ than in $\mathcal{S}$, it cannot finish later in $\mathcal{S}'$. □

Theorem 3 implies that the tardiness bounds stated in Theorem 2 also apply to periodic tasks without Assumption (A). We next show that they also apply to sporadic tasks.

For this purpose, we introduce a new task model, which we call the *varying-period periodic* (*VPP*) task model. An implicit-deadline VPP task τ_i^V has a pre-defined utilization u_i^V and also releases a sequence of jobs. However, in contrast to the ordinary periodic task model, each VPP job $J_{i,j}^V$ has its own worst-case execution requirement, denoted $C_{i,j}$. After its first invocation, a VPP task τ_i^V will release each job $J_{i,j+1}^V$ *exactly* $T_{i,j} = C_{i,j}/u_i$ time units after $J_{i,j}^V$'s release. Also, each job $J_{i,j}^V$ has a deadline $T_{i,j}$ time units after its release. For each VPP task τ_i^V, C_i^V is defined as $C_i^V = \max\{C_{i,j} \mid j \geq 1\}$ and T_i^V is defined as $T_i^V = \max\{T_{i,j} \mid j \geq 1\}$. Note that an ordinary periodic task τ_i is a special case of a VPP task where $C_{i,j} = C_{i,k}$ holds (and hence $T_{i,j} = T_{i,k}$ holds) for any j and k.

VPP tasks with Assumption (A). Given Assumption (A),[11] a VPP task set will have exactly the same ideal schedule as the corresponding ordinary periodic task set that includes the same tasks as in the VPP task set, where each such task has the same utilization as in the VPP task set, and releases its first jobs at the same time as in the VPP task set. Given this observation, we can re-work the proof in Sec. 3, assuming the VPP task model instead of the ordinary periodic one. In particular, with few exceptions (see below), every lemma, property, claim, and theorem in Sec. 3 can be *literally* re-stated and re-proved to pertain to VPP tasks (with Assumption (A)) by replacing C_i, T_i, and $C_{\max}$ by C_i^V, T_i^V, and $C_{\max}^V$, respectively, where $C_{\max}^V = \max\{C_i^V \mid 1 \leq i \leq n\}$.

The only modification that is a bit tricky is that, in the proof of Lemma 2, (9) and (10) should *not* be re-written as $0 < e_i(t) \leq C_i^V$ and $\mathsf{A}(\mathcal{S}, \tau_i, 0, t) = W + C_i^V - e_i(t)$, respectively, because the latter might not necessarily be true even with Assumption (A). Instead, they should be re-written as $0 < e_i(t) \leq C_{i,j}$ and $\mathsf{A}(\mathcal{S}, \tau_i, 0, t) = W + C_{i,j} - e_i(t)$, respectively. Then, (12) should be re-written as $(t - d_i(t))u_i < \mathsf{lag}(\tau_i, t, \mathcal{S}) \leq (t - d_i(t))u_i + C_{i,j}$, which implies

$$t - \frac{\mathsf{lag}(\tau_i, t, \mathcal{S})}{u_i} < d_i(t) \leq t - \frac{\mathsf{lag}(\tau_i, t, \mathcal{S})}{u_i} + T_{i,j}.$$

Finally, by definition, because $T_i^V \geq T_{i,j}$, we can obtain

$$t - \frac{\mathsf{lag}(\tau_i, t, \mathcal{S})}{u_i} < d_i(t) \leq t - \frac{\mathsf{lag}(\tau_i, t, \mathcal{S})}{u_i} + T_i^V.$$

After re-working all of the proofs in Sec. 3, the following theorem, which is the VPP counterpart to Theorem 2, can be proven, assuming Assumption (A).

Theorem 4. *In $\mathcal{S}$, the tardiness Δ_i^V of VPP task τ_i^V is bounded as follows.*

If $\rho = 1$, then $\Delta_i^V \leq \dfrac{n \cdot C_{\max}^V}{u_i}$;

if $\rho > 1$, then

$$\Delta_i^V \leq \frac{\rho^{m-1} \cdot (n-m+1)C_{\max}^V + \frac{\rho^{m-1}-1}{\rho-1} \cdot C_{\max}^V}{u_i}.$$

Because it would be almost identical to the proofs in Sec. 3, we do not provide a full proof of this theorem here. As an alternative, we could have considered only the VPP task model in Sec. 3, but this would have further complicated the proof there, both conceptually and notationally.

VPP tasks without the Assumption (A). Notice that Theorem 3 was established by reasoning only about jobs: it does not matter whether task periods are constant or variable. Thus, by Theorem 3, Theorem 4 also holds for VPP tasks *without* Assumption (A).

Sporadic tasks. We now show that tardiness bounds can be determined for any sporadic task set by viewing any instance[12] of such a task set as an instance of a VPP task set

[10] To see this, note that, if a job J' in $\mathcal{J}$ is not ready in $\mathcal{S}$ but ready in $\mathcal{S}'$ at time t, then J' must be *pending* at time t in both $\mathcal{S}$ and $\mathcal{S}'$, and some preceding job of the same task must be ready in $\mathcal{S}$ but completed in $\mathcal{S}'$.

[11] That is, every VPP job $J_{i,j}^V$ executes for *exactly* its execution requirement $C_{i,j}$.

[12] An instance of a task set is defined by specifying a set of *concrete* job release times and *actual* execution requirements in accordance with each

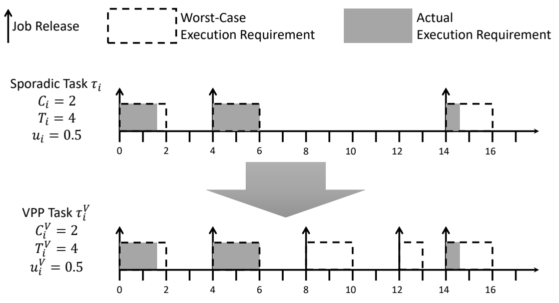


Figure 1: Transforming a sporadic task into a VPP task.

where Assumption (A) may not hold. The transformation to a VPP task set is depicted in Fig. 1 and explained next.

In the sporadic task model, a task $\tau_i = (C_i, T_i)$ might have two consecutive jobs that have a release separation of more than T_i time units. Let $J_{i,j}$ and $J_{i,j+1}$ be such two jobs, *i.e.*, $a_{i,j+1} - a_{i,j} > T_i$. Let us denote $a_{i,j+1} - a_{i,j}$ as $(k+1)T_i + r$ where k is an integer such that $k \geq 0$ and r is a real number such that $0 \leq r < T_i$ (k and r can be easily calculated from $a_{i,j+1} - a_{i,j}$). To transform to a VPP task, we add k jobs, with the ℓ^{th} one released at time $a_{i,j} + \ell \cdot T_i$ for $1 \leq \ell \leq k$, all with a worst-case execution requirement of C_i, plus an additional job, if $r > 0$, released at time $a_{i,j+1} - r$, where this job has a worst-case execution requirement of $r \cdot C_i / T_i$. We do this for every task τ_i whenever $a_{i,j+1} - a_{i,j} > T_i$ holds for two consecutive job releases $a_{i,j}$ and $a_{i,j+1}$. The resulting task set a VPP task set, where each VPP task τ_i^V has the parameters $u_i^V = C_i/T_i$, $C_i^V = C_i$, and $T_i^V = T_i$. Now, to complete the transformation, we can simply define the *actual* execution requirement of each added job to be zero.

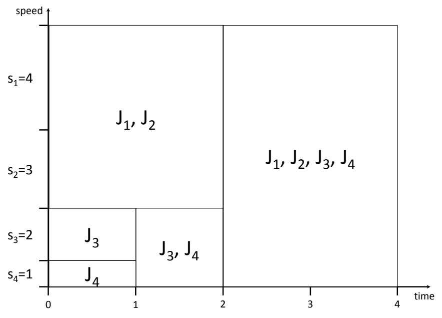
As we discussed earlier, Theorem 4 holds for VPP tasks without Assumption (A). Thus, the tardiness bounds in Theorem 4 apply to every instance of a VPP task set resulting from the transformation process above, and therefore to every instance of the original sporadic task set. By substituting the notation pertaining to the original sporadic task set into Theorem 4, it follows that the tardiness bounds in Theorem 2 hold for sporadic tasks as well.

Appendix B: Level Algorithm

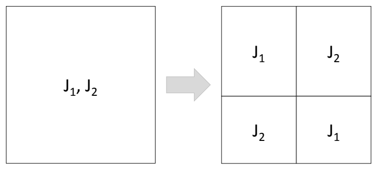
The Level Algorithm was proposed by Horvath *et al.* [10] for scheduling a set of one-shot jobs on a uniform multiprocessor and minimizing *makespan*, *i.e.*, the time required for finishing all jobs. A job's *level* is defined by its remaining execution time. The greater a job's level, the faster the processor on which it is scheduled, and all jobs that attain the same level are thereafter *jointly executed*, equally sharing the processors on which they are scheduled. The following example illustrates the Level Algorithm.

Ex. 1. Consider using the Level Algorithm to schedule four jobs, with execution requirements $J_1 = 12$, $J_2 = 12$, $J_3 = 8.5$, and $J_4 = 7.5$, on a uniform platform $\pi = \{s_1 = 4, s_2 = 3, s_3 = 2, s_4 = 1\}$. J_1 and J_2 have the same

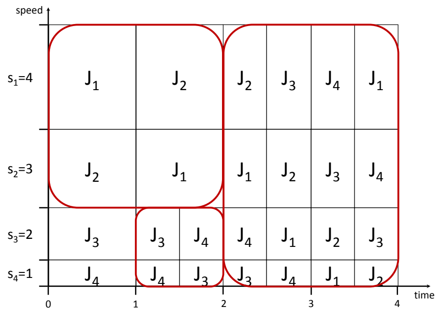
task's specification.



(a) Conceptual Level Algorithm schedule



(b) Real schedule for "jointly execute"


speed
s_1=4
s_2=3
s_3=2
s_4=1
J_1 J_2 J_2 J_3 J_4 J_1
J_2 J_1 J_1 J_2 J_3 J_4
J_3 J_3 J_4 J_4 J_1 J_2 J_3
J_4 J_4 J_3 J_3 J_4 J_1 J_2
0 1 2 3 4
time

(c) Real schedule for all jobs

Figure 2: Level Algorithm resulting schedule for Ex. 1.

execution cost, or level, so they are jointly executed from the beginning; J_3 and J_4 attain the same level at time 1, so they are jointly executed after time 1. At time 2, all jobs attain the same level, and hence all jobs are jointly executed afterward. Fig. 2(a) shows the resulting schedule produced by the Level Algorithm for this example. Fig. 2(b) shows the real schedule for "jointly executing." Fig. 2(c) shows the real schedule for all four considered jobs.

As proposed in [8], implicit-deadline periodic tasks can be supported by applying the Level Algorithm to consecutive small time intervals of length δ, ensuring that each task τ_i executes for $u_i \cdot \delta$ time units in each such interval. For example, four tasks with utilizations $u_1 = 12$, $u_2 = 12$, $u_3 = 8.5$, and $u_4 = 7.5$ would have a schedule similar to Fig. 2(c) during each δ-length time interval. In order to guarantee HRT correctness, δ must evenly divide all periods.

As seen in Ex. 1, the Level Algorithm is quite complicated, requiring a sophisticated implementation, and is prone to producing frequent preemptions and migrations. Furthermore, in a dynamic system, the schedule would need to be re-calculated every time a task joins or leaves the system. GEDF can be much more easily used in such a context.