Extending EDF for Soft Real-Time Scheduling on Unrelated Multiprocessors

Stephen Tang, Sergey Voronov, and James H. Anderson
Department of Computer Science, University of North Carolina at Chapel Hill
{sytang|rdkl|anderson}@cs.unc.edu

***Abstract*—Though recent work has established the soft real-time (SRT)-optimality of Earliest-Deadline-First (EDF) variants on multiprocessor models with limited heterogeneity (e.g., uniform speeds or affinity masks), such models are insufficient to describe modern multiprocessors, which have grown increasingly heterogeneous. This fact highlights the need to extend theoretical results to more asymmetric models, such as the unrelated multiprocessor model. This paper presents an EDF variant tailored for this model and proves that it is at least *nearly* SRT-optimal. Simulation results for random task systems are also presented that suggest that the proposed EDF variant may actually be SRT-optimal.**

***Index Terms*—Real-time scheduling theory, unrelated multiprocessors**

I. Introduction

The significance of the *unrelated* multiprocessor model, under which execution speed depends on both the task being executed and the processor being executed on, has increased with the heterogeneity of modern multiprocessors. Sources of this increasing heterogeneity include heterogeneous architectures such as big.LITTLE by ARM, accelerators such as Graphics Processing Units (GPUs) and Digital Signal Processors (DSPs), and features such as Dynamic Voltage and Frequency Scaling (DVFS) and processor affinities (per-task restrictions upon which processors said tasks may execute on).

Conversely, the theoretical understanding of unrelated multiprocessors for real-time scheduling has lagged behind their proliferation, often falling back on existing techniques for identical multiprocessors (where execution speeds do not vary). For example, a common approach for dealing with unrelated speeds is to partition tasks among clusters of processors, with the processors in each cluster being of the same type (an exemplar of this approach is given in [1], which itself cites several related works). While the illusion of homogeneity has allowed the real-time community to fall back on existing analyses, partitioning often relies on heuristics (due to the intractability of bin-packing) and results in capacity loss due to the inability to split tasks across clusters.

Fully migratory approaches to scheduling under unrelated multiprocessors can avoid capacity loss, but are less common. Work in this vein includes [2] and [3], both of which present schedulers that can optimally schedule tasks to meet all deadlines. A drawback of these schedulers is that they require partitioning time into slices between deadlines such that any task receives its proportionate share of execution within a time slice. This approach may be impractical due to frequent preemptions caused by short time slices, a tradeoff previously observed in work on Pfair scheduling [4].

This tradeoff was partially resolved for identical multiprocessors with Earliest-Deadline-First (EDF) scheduling. Unlike Pfair, preemptions under EDF are limited to job releases and completions. Consequently, deadlines may be missed under EDF; however, EDF is *soft real-time (SRT)-optimal* under identical multiprocessors [5], meaning that any task's tardiness is bounded if the system is feasible.

As discussed below, the SRT-optimality of EDF has been extended to consider processor speeds with limited heterogeneity (i.e., special cases of unrelated multiprocessors besides homogeneous multiprocessors). To our knowledge, no attempt has been made to extend these SRT-optimality results to fully unrelated multiprocessors. Such theoretical results are necessary to help inform the development of EDF implementations that will need to consider unrelated multiprocessors in the future, such as SCHED_DEADLINE [6] in Linux. We highlight SCHED_DEADLINE because its documentation explicitly mentions that response times are limited if the platform is not over-utilized [7].

EDF variants. Prior works that have extended the results of [5] to more heterogeneous processor models have done so by proposing *EDF variants* for their specific models. This is because naïve implementations of EDF that only schedule tasks with the earliest deadlines without care for which processors tasks are scheduled on fail to consider heterogeneity. This often results in capacity loss. Proposed EDF variants avoid capacity loss by adding rules to standard EDF that result in tasks being migrated more aggressively to better utilize any available processors; these variants reduce to standard global EDF for the special case where the multiprocessor is identical.

Relationships between EDF variants and their targeted platforms are illustrated in Fig. 1. EDF variants have been proven SRT-optimal for uniform multiprocessors [8] (in which execution speeds depend on the processor, but not the task) and for identical multiprocessors with affinities [9], [10]. We denote these variants as Ufm-EDF and Strong-APA-EDF, respectively. At a high level, Ufm-EDF migrates tasks such that tasks with earlier deadlines run on faster processors. Likewise, Strong-APA-EDF migrates tasks to maximize the

This was supported by NSF grants CNS 1563845, CNS 1717589, CPS 1837337, CPS 2038855, and CPS 2038960, ARO grant W911NF-20-1-0237, and ONR grant N00014-20-1-2698.

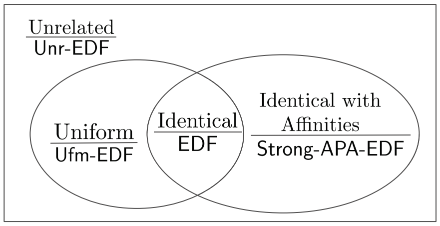


Fig. 1: Relationships between heterogeneous models and EDF variants. (Note that for Unr-EDF, the other variants are only approximately special cases.)

number of scheduled tasks.

Extending EDF's SRT-optimality to unrelated multiprocessors is challenging for two reasons. First, it is not immediately obvious how to migrate tasks to best utilize the available processors when the processor model is unrelated. For example, consider a task system with two tasks and two processors, with one task, τ_e, having a substantially earlier deadline than the other, τ_l. Suppose one processor has a fast execution speed for both tasks, while the other slower processor executes τ_e with moderate speed and cannot execute τ_l at all (due to its affinity setting). Scheduling τ_e on the faster processor (as would Ufm-EDF) underutilizes the platform by only scheduling one available task. In contrast, scheduling τ_e on the slower processor allows for both tasks to be scheduled (as would Strong-APA-EDF), but scheduling a higher-priority task on a slower processor seems antithetical to EDF.

The second reason is that a property, called HP-LAG-Compliance [10], which is upheld by Ufm-EDF and Strong-APA-EDF on their respective processor models, is not generally true under any scheduler in the unrelated model (see Sec. 8 of [10]). This is problematic because the SRT-optimality proofs of both Ufm-EDF and Strong-APA-EDF on their respective models heavily rely on HP-LAG-Compliance. Thus, any analysis for an EDF variant for the unrelated model requires fundamentally new insights and invariants.

Contributions. In this work, we propose Unr-EDF, an EDF variant for unrelated multiprocessors. Unr-EDF migrates tasks to best utilize the multiprocessor by solving instances of an *assignment problem*. We justify Unr-EDF as our choice of variant by proving that Unr-EDF (approximately) reduces to Ufm-EDF and Strong-APA-EDF for the special cases of unrelated multiprocessors where the multiprocessor is uniform or identical with affinities, respectively—a variant that reduces *exactly* to Ufm-EDF and Strong-APA-EDF is problematic for reasons we discuss in Sec. IV.

As solving an assignment problem at every scheduling event may result in impractically large overheads, we show that Unr-EDF can potentially be implemented more efficiently by leveraging a solution [11] to an online version of the assignment problem called the *incremental assignment problem*. This gives Unr-EDF comparable asymptotic time complexity to that of Strong-APA-EDF under arbitrary affinities.

We prove that Unr-EDF is at least *nearly* SRT-optimal. In particular, we prove that Unr-EDF guarantees bounded tardiness as long as no task or processor is tight—a task (resp., processor) is *tight* if any increase (resp., decrease) in its utilization (resp., capacity) results in an infeasible system. The tardiness bound we prove is inversely proportional to a task-system-dependent value ℓ, which approaches 0 as any task or processor approaches tightness.[1] Hence, tardiness becomes unbounded once any task or processor is tight.[2]

To evaluate our tardiness bounds, we simulated Unr-EDF on randomly generated task systems. Observed tardiness remained below the largest period as $\ell \to 0$, unlike what our tardiness bound predicts. This suggests that Unr-EDF may in fact be SRT-optimal.

Placing our contributions in context. As presented in this work, the above contributions are likely not yet suitable for practical use. As an implementation is not provided, we lack grounds to argue that Unr-EDF has reasonable overheads. Our tardiness bounds are also likely overly pessimistic if any tasks or processors approach tightness. Nevertheless, this work has theoretical value as a first step towards extending EDF for SRT-optimality under the unrelated model.

This theoretical value is illustrated with the context of prior work on EDF variants. Prior efforts for designing EDF variants for multiprocessors with limited heterogeneity that lend themselves to practical implementations and tardiness bounds were non-trivial. Such efforts were spread over several submissions, authors, and years. For example, with respect to implementations, efforts to improve the practicality of Strong-APA-EDF (originally proposed in [9]) by restricting to special cases of affinity masks have warranted their own publications [15], [16]. With respect to optimality, the first proof of Ufm-EDF's SRT-optimality [8] followed a series of works considering whether any EDF variant was SRT-optimal on uniform multiprocessors [17], [18]. Also, reducing analytical tardiness bounds to match observed tardiness has remained an open problem even for EDF on identical multiprocessors, and this problem has inspired multiple works [19]–[21].

These prior works show that it is the norm for initial works on new EDF variants to require refinement by future work.

Organization. The remainder of this paper is organized as follows. In Sec. II, we cover needed background. In Sec. III, we define our new EDF variant Unr-EDF, prove that it approximately reduces to Ufm-EDF and Strong-APA-EDF for their respective special cases of multiprocessors, and demonstrate how it can be efficiently implemented by leveraging the incremental assignment problem. In Sec. IV, we prove our sufficient condition for bounded tardiness under Unr-EDF. In Sec. V, we evaluate our derived tardiness bound via simulation. We conclude in Sec. VI.

[1]The analysis in this work does not permit tightness, even for the special cases where the multiprocessor is uniform or identical with affinities. Unr-EDF is actually SRT-optimal for such special cases. This can be proven by showing that Unr-EDF is in the class of *window-constrained schedulers*, which are known to be SRT-optimal for these special cases [12].

[2]A parallel can be drawn to approximation schemes for feasibility analysis of fixed-priority uniprocessor schedulers [13], [14], where the runtime complexity of the approximation scheme is inversely proportional to the distance between the approximation and an optimal condition.

TABLE I: Notation and Terminology.

Symbol	Meaning
τ	Task system
τ_i	i^{th} task
n	Number of tasks
π	Processors
π_j	j^{th} processor
$s_{i,j}$	Speed of τ_i on π_j
$s_{\max}$	Largest speed
C_i	τ_i's worst-case execution requirement
T_i	τ_i's period
u_i	C_i/T_i
$T_{\max}$	Largest period
$u_{\max}$	Largest utilization
$u_{\min}$	Smallest utilization
$\tau_{i,j}$	j^{th} job of τ_i
$C_{i,j}$	Execution requirement of $\tau_{i,j}$
$r_{i,j}$	Release time of $\tau_{i,j}$
$d_{i,j}$	Deadline of $\tau_{i,j}$
current	Incomplete job with earliest release time
ready	Current job that has been released
pending	Task with ready job; able to be scheduled
$r_i(t)$	Release time of current job of τ_i at t
$d_i(t)$	Deadline of current job of τ_i at t
$C_i(t)$	Total execution requirement of job of τ_i that is current at t
$c_i(t)$	Remaining execution of current job of τ_i at t
$s_i(t)$	Speed of processor assigned to τ_i at t
MVM	Maximum Vertex Matching—see (1)
ϕ_i	Weight of τ_i in MVM
$w_{i,j}$	Weight of (τ_i, π_j) in assignment problem
$x_{i,j}$	Decision variables of MVM or assignment problem
$\tau^p(t)$	Set of pending tasks at t
I-Unr-EDF	Idealized Unrelated EDF—see (3)
Strong-APA-EDF	EDF variant for identical w/ affinities—see [9]
Ufm-EDF	EDF variant for uniform—see [8]
Unr-EDF	Unrelated EDF—see (4)
$R_i(t)$	Latest release time of any job of τ_i no later than t
$R'_i(t)$	Latest pseudo-release—see Def. 4
$D_i(t)$	Latest pseudo-deadline—see Def. 5
$\Phi_i(t)$	Weight function of τ_i used by Unr-EDF—see Def. 6
$vt_i(t)$	Virtual time of τ_i at t—see Def. 7
$\text{Dev}_i(t)$	Deviation of τ_i at t—see Def. 8
non-fluid	See Def. 9
x'	Solution (with ℓ) of sufficient condition (12)
ℓ	Parameter corresponding with tightness—see (12)
K	See (13)

II. Background

In this section, we present our system model and discuss several optimization problems of relevance to this work.

A. System Model

A table of notation is provided in Tbl. I.

We consider n tasks $\tau = \{\tau_1, \tau_2, \dots, \tau_n\}$ running on n processors $\pi = \{\pi_1, \pi_2, \dots, \pi_n\}$. Processor π_j executes task τ_i with speed $s_{i,j} \geq 0$. The largest $s_{i,j}$ is denoted $s_{\max}$. Tasks are sporadic, and we assume familiarity with the sporadic model. Task τ_i has worst-case execution requirement C_i (relative to an execution speed of 1.0), period T_i, and utilization $u_i \triangleq C_i/T_i$. For task τ_i, C_i, T_i, and u_i are all positive. The largest period and utilization are denoted $T_{\max}$ and $u_{\max}$. The smallest utilization is denoted $u_{\min}$.

We assume an equal number of tasks and processors to facilitate usage of theorems on the assignment problem, which canonically assumes two input sets of equal size. This can be assumed without loss of generality. If the tasks outnumber the processors, we can add processors such that $s_{i,j} = 0$ for any task τ_i. Tasks cannot make progress on such processors, so their addition does not affect our system. Likewise, if the processors outnumber the tasks, tasks with $u_i = 0$ may be added. Note that minor modifications to definitions and quantifiers in our analysis are required when accounting for such tasks to avoid division by zero (e.g., the $u_{\min}$ in the denominator of our tardiness bound (21) is changed to denote the smallest *positive* utilization). Such modifications are not discussed due to space constraints.

Task τ_i releases an infinite sequence of jobs with $\tau_{i,j}$ denoting the j^{th} job of τ_i for $j \geq 1$. Job $\tau_{i,j}$ has execution requirement $C_{i,j} \in (0, C_i]$, release time $r_{i,j}$, and deadline $d_{i,j} = r_{i,j} + T_i$. Release times are separated such that for any $j \geq 1$, we have $r_{i,j} + T_i \leq r_{i,j+1}$.

▷ **Def. 1.** At time t, the *current* job of task τ_i is the incomplete job of τ_i that has the earliest release time at time t. If the current job of τ_i at t is released by time t, this job is *ready*. Tasks with ready jobs are *pending*. ◁

We let $r_i(t)$, $d_i(t)$, $C_i(t)$, and $c_i(t)$ be the release time, deadline, total execution requirement, and remaining execution requirement of the current job of τ_i at t ($C_i(t)$ and $c_i(t)$ are also relative to an execution speed of 1.0). Task τ_i's *deadline* at time t is defined as $d_i(t)$.

It will be convenient for our analysis to define schedulers via how tasks are *assigned* processors. By this, we mean that a task τ_i executes on a processor π_j under a given scheduler if τ_i is assigned π_j by said scheduler and τ_i is pending. We make a distinction between a task being assigned and it being executed because the instances of the assignment problem used to define our EDF variant will always assign each task a processor, while said tasks will only execute if they are pending. Let $s_i(t)$ be $s_{i,j}$ when τ_i is assigned π_j.

We assume time is continuous and starts at 0.

B. Relevant Optimization Problems

Some EDF variants discussed later in Sec. III are defined via the maximum vertex matching (MVM) and assignment problems on bipartite graphs. For ease of notation, denote the node sets of the bipartite graph as τ and π. A matching on a bipartite graph is a subset of edges in the graph such that each vertex shares an edge with at most one other vertex.

MVM. MVM seeks to pair vertices such that the most valuable vertices are paired. Under MVM, each vertex $\tau_i \in \tau$ has weight $\phi_i \geq 0$ and E denotes the edges in the bipartite graph. MVM can be expressed as follows.

$$
\left.\begin{aligned}
&\max \sum_{\tau_i \in \tau} \phi_i \sum_{(\tau_i,\pi_j)\in E} x_{i,j} \text{ s.t.}\\
&\quad \forall \tau_i \in \tau : \sum_{\pi_j \in \pi} x_{i,j} = 1\\
&\quad \forall \pi_j \in \pi : \sum_{\tau_i \in \tau} x_{i,j} = 1\\
&\quad \forall \tau_i \in \tau \wedge \pi_j \in \pi : x_{i,j} \in \{0,1\}
\end{aligned}\right\} \quad (1)
$$

The constraints of this integer program require that each node in τ is matched with a unique node in π.

The assignment problem. Under the assignment problem, each edge (τ_i, π_j) has weight $w_{i,j} \geq 0$. The assignment problem can be expressed as follows.

$$
\left.\begin{aligned}
&\max \sum_{\tau_i \in \tau} \sum_{\pi_j \in \pi} w_{i,j} x_{i,j} \text{ s.t.}\\
&\quad \forall \tau_i \in \tau : \sum_{\pi_j \in \pi} x_{i,j} = 1\\
&\quad \forall \pi_j \in \pi : \sum_{\tau_i \in \tau} x_{i,j} = 1\\
&\quad \forall \tau_i \in \tau \wedge \pi_j \in \pi : x_{i,j} \in \{0,1\}
\end{aligned}\right\} \quad (2)
$$

Note that (1) is a special case of (2) where $w_{i,j} = \phi_i$ if $(\tau_i, \pi_j) \in E$ and 0 otherwise.

Linear program relaxation. The following well-known theorem about the assignment problem is used by our analysis. A proof can be found in Theorem 3.2.1 of [22].

Theorem 1. *The optimum value of (2) remains optimal if $x_{i,j} \in \{0,1\}$ is relaxed to $x_{i,j} \geq 0$.*

Theorem 1 will be used to lower bound the efficacy of Unr-EDF (defined via the assignment problem) assuming our sufficient condition (expressed as a linear program) is true.

III. I-Unr-EDF and Unr-EDF

Here we show that EDF variants Ufm-EDF and Strong-APA-EDF are special cases of a new EDF variant we call Idealized Unr-EDF (I-Unr-EDF) on their respective targeted platforms (see Fig. 1). This justifies that (I-)Unr-EDF are EDF variants. We also explain why I-Unr-EDF is not directly implementable and propose Unr-EDF.

▷ **Def. 2.** Let $\tau^p(t) \subseteq \tau$ be the pending tasks at time t. ◁

I-Unr-EDF is defined via an optimization problem.

$$
\left.\begin{aligned}
&\max \sum_{\tau_i \in \tau^p(t)} (T_{\max} + t - d_i(t)) \sum_{\pi_j \in \pi} s_{i,j} x_{i,j} \text{ s.t.}\\
&\quad \forall\ \tau_i \in \tau \sum_{\pi_j \in \pi} x_{i,j} = 1\\
&\quad \forall\ \pi_j \in \pi : \sum_{\tau_i \in \tau} x_{i,j} = 1\\
&\quad \forall\ \tau_i \in \tau \wedge \pi_j \in \pi : x_{i,j} \in \{0,1\}
\end{aligned}\right\} \quad (3)
$$

I-Unr-EDF: At time t, assign tasks such that $\tau_i \in \tau$ is assigned on π_j if $x_{i,j} = 1$ in the solution to the optimization problem in (3).

The expression $T_{\max} + t - d_i(t)$ in the objective function of (3) rewards assigning tasks with earlier deadlines (hence, $-d_i(t)$) to faster processors. $T_{\max} + t$ is an offset used to guarantee non-negative weights.

A. Existing EDF Variants are Special Cases of I-Unr-EDF

For identical multiprocessors with affinities, $s_{i,j} = 1.0$ if task τ_i has affinity for processor π_j and $s_{i,j} = 0$ otherwise. The existing Strong-APA-EDF algorithm [9] for identical multiprocessors with affinities is defined as follows.

Strong-APA-EDF: At time t, assign to each pending task τ_i a value $\phi_i \geq 0$ such that $d_{i_1}(t) < d_{i_2}(t) \Leftrightarrow \phi_{i_1} > \phi_{i_2}$. Assign $\phi_i = 0$ for non-pending tasks. Solve the instance of MVM (see (1)) that results when an edge exists in E if its corresponding task has affinity for its corresponding processor, and assign τ_i on π_j if $x_{i,j} = 1$.

Note that [9], which primarily considered fixed-priority scheduling, does not specify how to assign ϕ_i for EDF.

Lemma 1. *Strong-APA-EDF on identical multiprocessors with affinities is a special case of I-Unr-EDF.*

Proof. We prove this lemma by showing that (3), which is an instance of the assignment problem, reduces to the MVM problem specified by Strong-APA-EDF for the special case where speeds are identical and tasks have specified affinities. Because MVM and the assignment problem only differ in their objective functions, it is sufficient to show that the objective function of (3) reduces to that of the MVM problem instance. The MVM problem in [9] has node sets $\tau^p(t)$ and π, with $(\tau_i, \pi_j) \in E$ if τ_i has affinity for π_j.

Let $\phi_i = T_{\max} + t - d_i(t)$. To see that these weights are well-defined, we must show that they are non-negative and that tasks with earlier deadlines have higher weights.

To show non-negativity, note that $\tau_i \in \tau^p(t)$ implies the current job of τ_i is ready. Thus, $t \geq r_i(t) \Rightarrow T_i + t \geq r_i(t) + T_i = d_i(t) \Rightarrow T_{\max} + t \geq d_i(t) \Rightarrow \phi_i = T_{\max} + t - d_i(t) \geq 0$.

To show $d_{i_1}(t) < d_{i_2}(t) \Leftrightarrow \phi_{i_1} > \phi_{i_2}$, note that $d_{i_1}(t) < d_{i_2}(t)$ implies that $\phi_{i_1} = T_{\max} + t - d_{i_1}(t) > T_{\max} + t - d_{i_2}(t) = \phi_{i_2}$. This reasoning can be applied in reverse.

As $s_{i,j} = 1$ if $(\tau_i, \pi_j) \in E$, and $s_{i,j} = 0$ otherwise, the objective function of (3) reduces to

$$
\begin{aligned}
&\sum_{\tau_i \in \tau^p(t)} (T_{\max} + t - d_i(t)) \sum_{\pi_j \in \pi} s_{i,j} x_{i,j}\\
&= \sum_{\tau_i \in \tau^p(t)} \phi_i \sum_{\pi_j \in \pi} s_{i,j} x_{i,j}\\
&= \sum_{\tau_i \in \tau^p(t)} \phi_i \sum_{(\tau_i,\pi_j)\in E} x_{i,j}
\end{aligned}
$$

Thus, the objective function of (3) reduces to the objective function of MVM (1), which is our proof obligation. □

It remains to show that Ufm-EDF is a special case of I-Unr-EDF for uniform multiprocessors. Under uniform, each processor π_j has speed s_j such that for any task τ_i, $s_{i,j} = s_j$. Formally, Ufm-EDF is defined as follows [8].

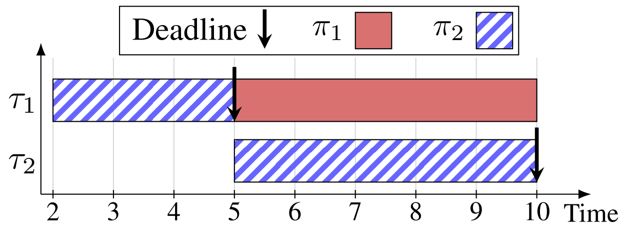


Fig. 2: Reschedule occurs without scheduling event under I-Unr-EDF.

> **Ufm-EDF**: Assign the task with the earliest deadline on the fastest processor, the task with the second earliest deadline on the second fastest processor, etc.

Lemma 2. *Ufm-EDF on a uniform multiprocessor is a special case of I-Unr-EDF.*

Proof. We prove the lemma by showing that (3) has the same behavior as Ufm-EDF when the multiprocessor is uniform without affinities, which we prove by contradiction.

Suppose otherwise that at time t, I-Unr-EDF schedules some task τ_{i_1} on processor π_{j_1} and task τ_{i_2} on processor π_{j_2} such that $d_{i_1}(t) < d_{i_2}(t)$ and $s_{j_2} > s_{j_1}$. Thus, in an optimal solution of (3) at time t, we have $x_{i_1,j_1} = x_{i_2,j_2} = 1$. Observe that an alternative solution that instead has $x_{i_1,j_2} = x_{i_2,j_1} = 1$ and is otherwise identical decreases the objective function of (3) by $(T_{\max} + t - d_{i_1}(t))s_{j_1} + (T_{\max} + t - d_{i_2}(t))s_{j_2}$ and increases the objective function by $(T_{\max} + t - d_{i_1}(t))s_{j_2} + (T_{\max} + t - d_{i_2}(t))s_{j_1}$. The net change is (in this and future derivations, justifications for certain steps are given in brackets)

$$\begin{aligned}
&(T_{\max} + t - d_{i_1}(t))s_{j_2} + (T_{\max} + t - d_{i_2}(t))s_{j_1}\\
&\quad - (T_{\max} + t - d_{i_1}(t))s_{j_1} - (T_{\max} + t - d_{i_2}(t))s_{j_2}\\
=\ &(T_{\max} + t - d_{i_1}(t))(s_{j_2} - s_{j_1})\\
&\quad + (T_{\max} + t - d_{i_2}(t))(s_{j_1} - s_{j_2})\\
=\ &(T_{\max} + t - d_{i_1}(t) - T_{\max} - t + d_{i_2}(t))(s_{j_2} - s_{j_1})\\
=\ &(d_{i_2}(t) - d_{i_1}(t))(s_{j_2} - s_{j_1})\\
>\ &\{d_{i_2}(t) > d_{i_1}(t) \wedge s_{j_2} > s_{j_1}\}\\
&0.
\end{aligned}$$

Because the objective function takes a higher value with the alternative solution and I-Unr-EDF chooses the solution that maximizes the objective function, I-Unr-EDF choosing the supposed solution is a contradiction. □

B. Approximating I-Unr-EDF with Unr-EDF

Under the special cases of identical with affinities or uniform, the solution to (3) is only dependent on the relative order of $d_i(t)$ and not the magnitude of $T_{\max} + t - d_i(t)$. This does not hold for unrelated, as shown in the following example.

► **Ex. 1.** This example is illustrated by Fig. 2. Consider a two-task and two-processor system with $s_{1,1} = 1$, $s_{1,2} = 2$, and $s_{2,1} = 0$ and $s_{2,2} = 2$. Let $T_{\max} = 10$ and suppose both tasks are pending over $[2, 10]$ with $d_1(t) = 5$ and $d_2(t) = 10$.

At time $t = 4$, we have $T_{\max} + t - d_1(t) = 10 + 4 - 5 = 9$ and $T_{\max} + t - d_2(t) = 10 + 4 - 10 = 4$. The solution of (3) is $x_{1,2} = x_{2,1} = 1$ with objective value $9s_{1,2} + 4s_{2,1} = 9(2) + 4(0) = 18$ (compared to $x_{1,1} = x_{2,2} = 1$ with value $9s_{1,1} + 4s_{2,2} = 9(1) + 4(2) = 17$).

However, at time $t = 6$, $T_{\max} + t - d_1(t) = 10 + 6 - 5 = 11$ and $T_{\max} + t - d_2(t) = 10 + 6 - 10 = 6$. The optimal solution of (3) at time $t = 6$ is then $x_{1,1} = x_{2,2} = 1$ with value $11s_{1,1} + 6s_{2,2} = 11(1) + 6(2) = 23$ (compared to $x_{1,2} = x_{2,1} = 1$ with value $11s_{1,2} + 6s_{2,1} = 11(2) + 6(0) = 22$).

Thus, a rescheduling occurs in $[2, 10]$ even though the tasks' deadlines did not change. ◄

This makes I-Unr-EDF impractical because rescheduling may occur at any time instant. The cause of this problem is that the coefficients in the objective function of (3) change at every time instant. We circumvent this by replacing t with the *latest pseudo-deadline*, defined below, which changes discretely.

▷ **Def. 3.** For task τ_i, $R_i(t) \triangleq \max\{0\} \cup \{r_{i,j} \mid r_{i,j} \le t\}$. ◁

With the exception of when $r_{i,1} \neq 0$, $R_i(t)$ is the latest release time of any job of τ_i by time t. Treating time 0 as a special case simplifies the following definition of pseudo-release times. Pseudo-releases simulate periodic job releases within any inter-release time greater than a period.

▷ **Def. 4.** The *latest pseudo-release* of task τ_i is $R'_i(t) \triangleq \max\{R_i(t) + kT_i \mid k \in \mathbb{N}_0 \wedge R_i(t) + kT_i \le t\}$. ◁

The definition of latest pseudo-deadline follows.

▷ **Def. 5.** For task τ_i, $D_i(t) \triangleq R'_i(t) + T_i$. ◁

Because the latest pseudo-release updates at least once every T_i time units, $D_i(t)$ is a reasonable *approximation* of (i.e., stays within a bounded interval around) t.

► **Ex. 2.** This example is illustrated by Fig. 3. Let task τ_i with $T_i = 10$ have initial release times $r_{i,1} = 12$, $r_{i,2} = 22$, and $r_{i,3} = 50$. Pseudo-releases within $[0, 50]$ occur at times 0, 10, 12, 22, 32, 42, and 50. $D_i(t)$ then changes values at $D_i(0) = 10$, $D_i(10) = 20$, $D_i(12) = 22$, $D_i(22) = 32$, $D_i(32) = 42$, $D_i(42) = 52$, and $D_i(50) = 60$. ◄

▷ **Def. 6.** The Unr-EDF weight function is

$$\Phi_i(t) \triangleq \begin{cases} 0 & \tau_i \notin \tau^p(t) \\ T_{\max} + D_i(t) - d_i(t) & \tau_i \in \tau^p(t) \end{cases} \quad ◁$$

Unr-EDF replaces $T_{\max} + t - d_i(t)$ in the objective function of (3) with $\Phi_i(t)$. Rescheduling is limited to job completions and pseudo-releases (though unlike prior EDF variants, tasks that are already pending may still be rescheduled due to their own pseudo-releases), as $\Phi_i(t)$ only changes with such events.

Consider the following instance of the assignment problem (2) in which $w_{i,j} = \Phi_i(t) \sum_{\pi_j \in \pi} s_{i,j}$.

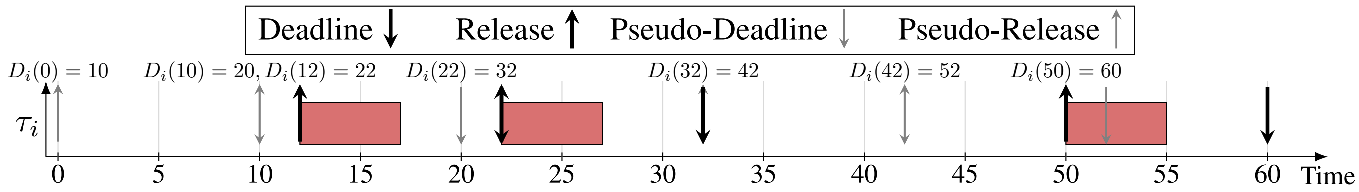


Fig. 3: Pseudo-deadline example.

$$\left.\begin{aligned}&\max \sum_{\tau_i \in \tau} \Phi_i(t) \sum_{\pi_j \in \pi} s_{i,j} x_{i,j} \text{ s.t.}\\ &\forall\ \tau_i \in \tau : \sum_{\pi_j \in \pi} x_{i,j} = 1\\ &\forall\ \pi_j \in \pi : \sum_{\tau_i \in \tau} x_{i,j} = 1\\ &\forall\ \tau_i \in \tau \wedge \pi_j \in \pi : x_{i,j} \in \{0,1\}\end{aligned}\right\} \tag{4}$$

Unr-EDF: At time t, for some optimal solution to (4), $\tau_i \in \tau$ is assigned π_j if $x_{i,j} = 1$.

C. Implementing Unr-EDF

As (4) is an instance of the assignment problem, independently solving (4) at every scheduling event can be done with time complexity $O(n^3)$ using the Hungarian algorithm [23].

This can be implemented more efficiently by leveraging an algorithm for the incremental assignment problem presented in [11]. We cover this at a high level due to space constraints. The input of the incremental assignment problem is an instance of the assignment problem of size $n-1$ alongside its optimal solution $x^{(n-1)}$ (as well as dual variables of $x^{(n-1)}$). The incremental problem then considers how to optimally compute $x^{(n)}$ from $x^{(n-1)}$ when new vertices τ_n and π_n (as well as their corresponding edges) are added. Computing $x^{(n)}$ from $x^{(n-1)}$ requires $O(n^2)$ time.

Assuming simultaneous scheduling events are serialized, let τ_k be the unique task whose value of Φ changed with some scheduling event. Let π_p be the unique processor τ_k was assigned to via x^{prev}, the solution of (4) before said scheduling event. Note that the reduced solution $x^{\text{prev},(n-1)}$ derived by removing $x_{k,p} = 1$ is itself an optimal solution of (4) if τ_k and π_p are removed from τ and π, respectively. Otherwise, prior to the scheduling event, a better solution than x^{prev} could have been constructed by combining the better solution than $x^{\text{prev},(n-1)}$ for $\tau \setminus \{\tau_k\}$ and $\pi \setminus \{\pi_p\}$ with $x_{k,p} = 1$. This would contradict the definition of Unr-EDF.

Because $x^{\text{prev},(n-1)}$ is an optimal assignment for $\tau \setminus \{\tau_k\}$ and $\pi \setminus \{\pi_p\}$ and τ_k and π_p can be treated as additional vertices, the algorithm in [11] can be applied to compute the optimal assignment after the scheduling event.

IV. A Sufficient SRT Condition

Due to space constraints, certain proofs that highly resemble proofs from prior works or are fairly intuitive are omitted. Omitted proofs are available in the appendix.

Proof strategy. We present the high-level steps of our proof for bounding tardiness under Unr-EDF assuming τ satisfies our sufficient condition (12). Consider any Unr-EDF schedule for any task system that satisfies (12).

Step 1: For any time $t \geq 0$, the state of each task τ_i is mapped to a scalar using a function called deviation (Dev). $\text{Dev}_i(t)$ (Def. 8) is a function of time t and the state of task τ_i's current job at t, and is defined such that the tardiness of task τ_i is roughly proportional to the largest $\text{Dev}_i(t)$ for any time $t \geq 0$ (Lemma 5).

Step 2: A necessary condition (6) on the assignment by Unr-EDF at time t is derived such that $\sum_{\tau_i \in \tau} u_i(\text{Dev}_i(t))^2$ is non-increasing at t (Lemma 7).

Step 3: Theorem 1, which relates instances of the assignment problem (such as Unr-EDF (4)) with linear programs (the sufficient condition (12)), is used to show that Unr-EDF satisfies the necessary condition (6) of Step 2 if $\sum_{\tau_i \in \tau} u_i(\text{Dev}_i(t))^2 = K$ for some K. That $\sum_{\tau_i \in \tau} u_i(\text{Dev}_i(t))^2$ is non-increasing is used to prove that $\sum_{\tau_i \in \tau} u_i(\text{Dev}_i(t))^2 \leq K$ for all $t \geq 0$ (Lemma 15).

Step 4: The bound $\sum_{\tau_i \in \tau} u_i(\text{Dev}_i(t))^2 \leq K$ is used to derive an upper bound on $\text{Dev}_i(t)$ for each task τ_i. Because $\text{Dev}_i(t)$ is proportional to tardiness and $\text{Dev}_i(t)$ is bounded, tardiness bounds can be derived (Theorem 2).

Steps 1 and 2 are covered in Sec. IV-A, and Steps 3 and 4 in Sec. IV-B.

A. Deviation Properties

Steps 1 and 2 are accomplished by proving properties about *deviation* [12], a measure of how behind a task's execution is at a specific time instant.[3] Deviation is similar to the well-known concept of lag, but is more closely tied to deadlines than lag when releases are sporadic and jobs do not execute to their worst-case requirement.

The definition of deviation relies on that of virtual time.

▷ **Def. 7.** The *virtual time* of task τ_i is

$$vt_i(t) \triangleq r_i(t) + T_i \frac{C_i(t) - c_i(t)}{C_i(t)}. \quad \triangleleft$$

While a job $\tau_{i,j}$ is current, $vt_i(t)$ interpolates between $\tau_{i,j}$'s release time and deadline based on what fraction of $\tau_{i,j}$'s execution requirement has been completed. As jobs do not receive negative execution, it is intuitive that $vt_i(t)$ is non-decreasing. This is formalized in Lemma 3, which is analogous to Lemma 4 of [12].

Lemma 3. *For task τ_i, $\forall t \geq 0 : \ \forall \epsilon > 0 : \ vt_i(t+\epsilon) \geq vt_i(t)$.*

[3] The definitions in this work vary slightly from those in [12]. Our proof reasons about $(\text{Dev}_i(t))^2$, which behaves undesirably when $\text{Dev}_i(t) < 0$. As such, we chose definitions such that $\forall t \geq 0 : \ \text{Dev}_i(t) \geq 0$.

In Steps 1 and 3, Lemma 3 is used in proofs by contradiction via showing that scenarios are impossible unless $vt_i(t)$ decreased for some τ_i. We present it early for this reason.

We now begin Step 1 by formally defining deviation.

$\triangleright$ **Def. 8.** τ_i has deviation $\text{Dev}_i(t) \triangleq \max\{0, t - vt_i(t)\}$. $\triangleleft$

Before completing Step 1 by proving Lemma 5, we require intermediate Lemma 4, which relates $vt_i(t)$ and $d_i(t)$.

Lemma 4. $vt_i(t) < d_i(t) \leq vt_i(t) + T_i$.

Proof. By Def. 7, $vt_i(t) = r_i(t) + T_i \frac{C_i(t) - c_i(t)}{C_i(t)}$. Because $0 < c_i(t) \leq C_i(t)$, we have $r_i(t) \leq vt_i(t) < r_i(t) + T_i$. Because $d_i(t) = r_i(t) + T_i$, we have $d_i(t) - T_i \leq vt_i(t) < d_i(t)$. Rearrangement yields the lemma statement. □

Step 1 is completed by showing that tardiness is bounded if deviation is bounded in Lemma 5.

Lemma 5. *If for some $L > 0$, for all $t \geq 0$, we have $Dev_i(t) \leq L$, then the tardiness of τ_i is at most L.*

Proof. We prove the contrapositive: if tardiness exceeds L then for some time instant t we have $\text{Dev}_i(t) > L$.

Let $\tau_{i,j}$ be a job with tardiness exceeding L. Then at $t' \triangleq d_{i,j} + L$, $\tau_{i,j}$ is released and incomplete. Either $\tau_{i,j}$ or an earlier job must be the current job of τ_i at t', so $d_i(t') \leq d_{i,j}$. Thus, $t' \geq d_i(t') + L \Rightarrow t' - vt_i(t') \geq d_i(t') - vt_i(t') + L$. By Lemma 4, $d_i(t') > vt_i(t')$, so $t' - vt_i(t') > L$. Because $L > 0$, by Def. 8, we have $\text{Dev}_i(t') > L$. □

Step 2 requires that we prove a condition under which $\sum_{\tau_i \in \tau} u_i (\text{Dev}_i(t))^2$ is non-increasing. In this context, this means the sum's value at t upper bounds the sum's value over some interval beginning at t. This will be shown in Lemma 7. This proof is simplified using Lemma 6, which considers the change in $(\text{Dev}_i(t))^2$ for a single task τ_i over such an interval. The proof of Lemma 6 relies on the concept of non-fluidity.

$\triangleright$ **Def. 9.** A scheduler is *non-fluid* if at any time t, if task τ_i is assigned processor π_j, then there exists $\delta > 0$ such that task τ_i is assigned processor π_j over $[t, t+\delta)$. $\triangleleft$

For a scheduler to be fluid, there must be some finite time interval in which the scheduler has infinitely many preemptions. Thus, any implementable scheduler is non-fluid.

Non-fluidity allows us to assume that tasks' rates of execution (i.e., $s_i(t)$) are constant over small time intervals. This is useful for reasoning about changes in $(\text{Dev}_i(t))^2$ over small intervals, as will be done in Lemma 6. Note that the proof of Lemma 6 is subdivided into Cases 6.1-6.3 depending on which argument of the $\max$ function is greater in Def. 8.

Lemma 6. *For a non-fluid scheduler,* $\forall \tau_i \in \tau : \ \forall t \geq 0 : \exists \delta > 0 : \ \forall \epsilon \in [0, \delta) :$

$$\begin{aligned} &(Dev_i(t+\epsilon))^2 \\ \leq\ &(Dev_i(t))^2 + 2\epsilon Dev_i(t)(1 - s_i(t)T_i/C_i(t)) \\ &+ \epsilon^2 (1 - s_i(t)T_i/C_i(t))^2. \end{aligned} \tag{5}$$

Proof. Restrict δ to be small enough such that the current job of τ_i and $s_i(t)$ are both constant over $[t, t+\delta)$ (as allowed by Def. 9). There are three cases: $t < vt_i(t)$, $t \geq vt_i(t) \wedge t + \epsilon < vt_i(t+\epsilon)$, or $t \geq vt_i(t) \wedge t + \epsilon \geq vt_i(t+\epsilon)$.

Case 6.1. $t < vt_i(t)$.

Further restrict δ such that $\delta \in (0, vt_i(t) - t)$. By Lemma 3, for any $\epsilon \in [0, \delta)$, $vt_i(t+\epsilon) - (t+\epsilon) \geq vt_i(t) - (t+\epsilon)$. Because $\epsilon < \delta < vt_i(t) - t$, we have $vt_i(t+\epsilon) - (t+\epsilon) > 0$. Thus, $t + \epsilon < vt_i(t+\epsilon)$.

By Def. 8 and because $t < vt_i(t)$ and $t + \epsilon < vt_i(t+\epsilon)$, we have $\text{Dev}_i(t) = \text{Dev}_i(t+\epsilon) = 0$. This satisfies (5).

Case 6.2. $t \geq vt_i(t)$ *and* $t + \epsilon < vt_i(t+\epsilon)$.

$$\begin{aligned} &(\text{Dev}_i(t+\epsilon))^2 \\ =\ &\{\text{By Def. 8}\} \\ &(\max\{0, t + \epsilon - vt_i(t+\epsilon)\})^2 \\ =\ &0 \ \{t + \epsilon - vt_i(t+\epsilon) < 0\} \\ \leq\ &\{\text{Squares of real numbers are non-negative}\} \\ &(\text{Dev}_i(t) + \epsilon(1 - s_i(t)T_i/C_i(t)))^2 \\ =\ &(\text{Dev}_i(t))^2 + 2\epsilon \text{Dev}_i(t)(1 - s_i(t)T_i/C_i(t)) \\ &\quad + \epsilon^2(1 - s_i(t)T_i/C_i(t))^2 \end{aligned}$$

Case 6.3. $t' \geq vt_i(t)$ *and* $t + \epsilon \geq vt_i(t+\epsilon)$.

$$\begin{aligned} &(\text{Dev}_i(t+\epsilon))^2 \\ =\ &\{\text{By Def. 8}\} \\ &(\max\{0, t + \epsilon - vt_i(t+\epsilon)\})^2 \\ =\ &\{t + \epsilon - vt_i(t+\epsilon) \geq 0\} \\ &(t + \epsilon - vt_i(t+\epsilon))^2 \end{aligned}$$

By Def. 7, $(\text{Dev}_i(t+\epsilon))^2 = (t + \epsilon - r_i(t+\epsilon) - T_i \frac{C_i(t+\epsilon) - c_i(t+\epsilon)}{C_i(t+\epsilon)})^2$. Because the current job of τ_i is constant over the interval $[t, t+\epsilon] \subset [t, t+\delta)$, $(\text{Dev}_i(t+\epsilon))^2 = (t + \epsilon - r_i(t) - T_i \frac{C_i(t) - c_i(t+\epsilon)}{C_i(t)})^2$. Because $s_i(t)$ is constant over this interval, $(\text{Dev}_i(t+\epsilon))^2 = (t + \epsilon - r_i(t) - T_i \frac{C_i(t) - c_i(t) + \epsilon s_i(t)}{C_i(t)})^2$. Thus,

$$\begin{aligned} &(\text{Dev}_i(t+\epsilon))^2 \\ =\ &\{\text{By Def. 7}\} \\ &(t - vt_i(t) + \epsilon\,[1 - s_i(t)T_i/C_i(t)])^2 \\ =\ &\{\text{By Def. 8 and } t - vt_i(t) \geq 0\} \\ &(\text{Dev}_i(t) + \epsilon\,[1 - s_i(t)T_i/C_i(t)])^2 \\ =\ &(\text{Dev}_i(t))^2 \\ &\quad + 2\epsilon \text{Dev}_i(t)(1 - s_i(t)T_i/C_i(t)) \\ &\quad + \epsilon^2(1 - s_i(t)T_i/C_i(t))^2 \end{aligned}$$

For all cases, (5) holds. □

Lemma 6 demonstrated the conditions on $\mathrm{Dev}_i(t)$, $s_i(t)$, T_i, and $C_i(t)$ for $(\mathrm{Dev}_i(t))^2$ to be non-increasing at t for a single task τ_i. Using Lemma 6, we can infer what conditions on the task system and scheduler assignment as a whole are necessary for sum $\sum_{\tau_i \in \tau} u_i(\mathrm{Dev}_i(t))^2$ to be non-increasing.

Lemma 7. *For a non-fluid scheduler, for any time t, if we have*

$$\sum_{\tau_i \in \tau} \mathit{Dev}_i(t)s_i(t) > \Delta + \sum_{\tau_i \in \tau} \mathit{Dev}_i(t)u_i \tag{6}$$

for some $\Delta > 0$ and $\sum_{\tau_i \in \tau} \mathit{Dev}_i(t) > 0$, then

$$\exists \delta > 0 : \forall \epsilon \in [0, \delta) : \\ \sum_{\tau_i \in \tau} u_i(\mathit{Dev}_i(t))^2 > \sum_{\tau_i \in \tau} u_i(\mathit{Dev}_i(t+\epsilon))^2. \tag{7}$$

Proof.

Claim 7.1. *We have*

$$\sum_{\tau_i \in \tau} u_i \mathit{Dev}_i(t)(1 - s_i(t)T_i/C_i(t)) < -\Delta.$$

Proof.

$$\begin{aligned}
&-\Delta \\
&> \{\text{By (6)}\} \\
&\quad \sum_{\tau_i \in \tau} \mathrm{Dev}_i(t)u_i - \sum_{\tau_i \in \tau} \mathrm{Dev}_i(t)s_i(t) \\
&= \sum_{\tau_i \in \tau} \mathrm{Dev}_i(t)(u_i - s_i(t)) \\
&= \sum_{\tau_i \in \tau} u_i \mathrm{Dev}_i(t)(1 - s_i(t)/u_i) \\
&\geq \{C_i(t)/T_i \leq u_i \Rightarrow -1/u_i \geq -T_i/C_i(t)\} \\
&\quad \sum_{\tau_i \in \tau} u_i \mathrm{Dev}_i(t)(1 - s_i(t)T_i/C_i(t))
\end{aligned}$$

■

By Lemma 6, for any time t, for each task τ_i, there exists $\delta > 0$ such that (5) is true. Let δ_i denote this δ for task τ_i. Let

$$\left.\begin{aligned}
\delta_{\max} &\triangleq \frac{2\Delta}{\sum_{\tau_i \in \tau} u_i(1 - s_i(t)T_i/C_i(t))^2} \\
\delta' &\triangleq \min\{\delta_1, \delta_2, \ldots, \delta_n, \delta_{\max}\}.
\end{aligned}\right\} \tag{8}$$

If the denominator of $\delta_{\max}$ is 0, then $\delta_{\max} \triangleq \infty$. We have $\delta' > 0$ because $\delta_i > 0$ holds for each i and, by the lemma statement, $\Delta > 0$. By Lemma 6 and because $\delta' \leq \delta_i$ for every task τ_i, then for each task τ_i we have

$$\begin{aligned}
\forall \epsilon \in [0, \delta') : (\mathrm{Dev}_i(t+\epsilon))^2 &\leq (\mathrm{Dev}_i(t))^2 \\
&+ 2\epsilon \mathrm{Dev}_i(t)(1 - s_i(t)T_i/C_i(t)) \\
&+ \epsilon^2(1 - s_i(t)T_i/C_i(t))^2.
\end{aligned}$$

Summing over all tasks and multiplying by u_i, we have for any $\epsilon \in [0, \delta')$,

$$\begin{aligned}
&\sum_{\tau_i \in \tau} u_i(\mathrm{Dev}_i(t+\epsilon))^2 \\
&\leq \sum_{\tau_i \in \tau} u_i \begin{bmatrix} (\mathrm{Dev}_i(t))^2 \\ +2\epsilon \mathrm{Dev}_i(t)(1 - s_i(t)T_i/C_i(t)) \\ +\epsilon^2(1 - s_i(t)T_i/C_i(t))^2 \end{bmatrix} \\
&= \left[\sum_{\tau_i \in \tau} u_i(\mathrm{Dev}_i(t))^2\right] \\
&\quad + 2\left[\epsilon \sum_{\tau_i \in \tau} u_i \mathrm{Dev}_i(t)(1 - s_i(t)T_i/C_i(t))\right] \\
&\quad + \epsilon^2\left[\sum_{\tau_i \in \tau} u_i(1 - s_i(t)T_i/C_i(t))^2\right] \\
&= \left[\sum_{\tau_i \in \tau} u_i(\mathrm{Dev}_i(t))^2\right] \\
&\quad + \epsilon \left(\begin{aligned} &2\left[\sum_{\tau_i \in \tau} u_i \mathrm{Dev}_i(t)(1 - s_i(t)T_i/C_i(t))\right] \\ &+\epsilon\left[\sum_{\tau_i \in \tau} u_i(1 - s_i(t)T_i/C_i(t))^2\right] \end{aligned} \right) \\
&< \{\text{By Claim 7.1}\} \\
&\quad \left[\sum_{\tau_i \in \tau} u_i(\mathrm{Dev}_i(t))^2\right] \\
&\quad + \epsilon\left(-2\Delta + \epsilon\left[\sum_{\tau_i \in \tau} u_i(1 - s_i(t)T_i/C_i(t))^2\right]\right) \\
&< \left\{\text{By (8)}, \epsilon < \delta' \leq \frac{2\Delta}{\sum_{\tau_i \in \tau} u_i(1 - s_i(t)T_i/C_i(t))^2}\right\} \\
&\quad \sum_{\tau_i \in \tau} u_i(\mathrm{Dev}_i(t))^2 + \epsilon(0) \\
&= \sum_{\tau_i \in \tau} u_i(\mathrm{Dev}_i(t))^2
\end{aligned}$$

Thus (7), the proof obligation, is true. □

Thus, the sum $\sum_{\tau_i \in \tau} u_i(\mathrm{Dev}_i(t))^2$ is non-increasing if (7) is true. This completes Step 2.

B. Analysis of Unr-EDF

Step 3 requires that we prove Unr-EDF satisfies condition (6) if $\sum_{\tau_i \in \tau} u_i(\mathrm{Dev}_i(t))^2 = K$ for some K. Note that the best scheduler for satisfying (6) (i.e., yields the largest difference between the left- and right-hand-sides of the inequality) is the scheduler whose choice of $s_i(t)$ at a given time t maximizes $\sum_{\tau_i \in \tau} \mathrm{Dev}_i(t)s_i(t)$. In comparison, Unr-EDF (4) maximizes $\sum_{\tau_i \in \tau} \Phi_i(t)s_i(t)$. We show Unr-EDF is related to this 'best' scheduler by showing $\Phi_i(t) \approx \mathrm{Dev}_i(t)$ in Lemma 11, whose proof requires intermediate Lemmas 8-10.

Lemma 8. $t < D_i(t) \leq t + T_i$.

Proof. Follows from Defs. 4 and 5. □

Lemma 9. $\tau_i \in \tau^p(t) \Rightarrow d_i(t) \leq t + T_i$.

Proof. Because $\tau_i \in \tau^p(t)$, we also have $r_i(t) \le t \Rightarrow r_i(t) + T_i \le t + T_i \Rightarrow d_i(t) \le t + T_i$. $\square$

Lemma 10. *For any $t \ge 0$ and task τ_i, $\Phi_i(t) \ge 0$.*

Proof. By Def. 6, we need only consider the case where $\tau_i \in \tau^p(t)$. We have $\Phi_i(t) = T_{\max} + D_i(t) - d_i(t)$. By Lemma 9, $\Phi_i(t) \ge T_{\max} + D_i(t) - t - T_i \ge D_i(t) - t$. By Lemma 8, $\Phi_i(t) > 0$. $\square$

That $\Phi_i(t) \approx \text{Dev}_i(t)$ is formalized in Lemma 11.

Lemma 11. *$\Phi_i(t) - 2T_{max} < Dev_i(t) \le \Phi_i(t)$.*

Proof. We consider three cases.

Case 11.1. $\tau_i \notin \tau^p(t)$.

By Def. 6, $\Phi_i(t) = 0$. By Def. 7, $t - vt_i(t) = t - r_i(t) - T_i\frac{C_i(t)-c_i(t)}{C_i(t)}$. Because $c_i(t) \le C_i(t)$, $t - vt_i(t) \le t - r_i(t)$. Because $\tau_i \notin \tau^p(t)$, the current job of τ_i at t is not released by t, thus $t - r_i(t) < 0$. Thus, $t - vt_i(t) < 0$, and by Def. 8, $\text{Dev}_i(t) = 0$. Thus, $\text{Dev}_i(t) = \Phi_i(t)$, which satisfies the lemma statement.

Case 11.2. $\tau_i \in \tau^p(t) \wedge t - vt_i(t) \ge 0$.

By Lemmas 4 and 8, we have $t - vt_i(t) - T_i < D_i(t) - d_i(t) < t - vt_i(t) + T_i$. Rearrangement yields $D_i(t) - d_i(t) - T_i < t - vt_i(t) < D_i(t) - d_i(t) + T_i$. By Def. 8 and $t - vt_i(t) \ge 0$, we have $D_i(t) - d_i(t) - T_i < \text{Dev}_i(t) < D_i(t) - d_i(t) + T_i \Rightarrow D_i(t) - d_i(t) - T_{\max} < \text{Dev}_i(t) < D_i(t) - d_i(t) + T_{\max}$. By Def. 6, we have $\Phi_i(t) - 2T_{\max} < \text{Dev}_i(t) < \Phi_i(t)$.

Case 11.3. $\tau_i \in \tau^p(t) \wedge t - vt_i(t) < 0$.

By Def. 8 and $t - vt_i(t) < 0$, we have $\text{Dev}_i(t) = 0$.

By $t - vt_i(t) < 0$, we have $t < vt_i(t)$. By Def. 7, we have $t < r_i(t) + T_i\frac{C_i(t)-c_i(t)}{C_i(t)}$. Because $c_i(t) \in (0, C_i(t)]$, we have $t < r_i(t) + T_i = d_i(t)$. Thus, $t < d_i(t)$. Furthermore, by Lemma 9, we have $t < d_i(t) \le t + T_i$.

Because $t < d_i(t) \le t + T_i$ and by Lemma 8, $t < D_i(t) \le t + T_i$, by Def. 6, we have $T_{\max} - T_i < \Phi_i(t) < T_{\max} + T_i \Rightarrow 0 < \Phi_i(t) < 2T_{\max}$. Because $\text{Dev}_i(t) = 0$, we have the lemma statement.

In all cases, we have the lemma statement. $\square$

By Lemma 11, the relative difference between $\sum_{\tau_i \in \tau} \Phi_i(t)$ and $\sum_{\tau_i \in \tau} \text{Dev}_i(t)$ decreases as both sums increase. Lemma 12 establishes how large K must be for these sums to have a given magnitude.

Lemma 12. *Associate each task $\tau_i \in \tau$ with a decision variable y_i. For any $K > 0$, the problem*

$$\min \sum_{\tau_i \in \tau} y_i \text{ such that}$$

$$\sum_{\tau_i \in \tau} u_i y_i^2 = K \tag{9}$$

$$y \ge 0 \tag{10}$$

has optimal value $\sqrt{K/u_{max}}$.

Proof. This problem is optimized when $y_i = \sqrt{K/u_i}$ for some unique y_i where $u_i = u_{\max}$, and $y_j = 0$ for all $j \ne i$.

We prove this by showing that the objective value of any other solution can be decreased.

Claim 12.1. *Let τ_i and τ_j be two tasks such that for some solution vector y, we have $y_j > 0$. The vector*

$$y'_k = \begin{cases} 0 & k = j \\ \sqrt{y_i^2 + \frac{u_j}{u_i} y_j^2} & k = i \\ y_k & \text{otherwise} \end{cases}$$

is also a solution.

Proof. We need to show that y' satisfies (9) and (10). (10) is true because y is a solution.

For (9), note that $\sum_{\tau_k \in \tau} u_k (y'_k)^2 = \sum_{\tau_k \in \tau \setminus \{\tau_i, \tau_j\}} u_k y_k^2 + u_i (y'_i)^2 + u_j (y'_j)^2 = K - u_i y_i^2 - u_j y_j^2 + u_i (y'_i)^2 + u_j (y'_j)^2 = K - u_i y_i^2 - u_j y_j^2 + u_i (y_i^2 + \frac{u_j}{u_i} y_j^2) + u_j(0) = K$. ■

Claim 12.2. *Let τ_i and τ_j be two tasks such that for some solution y, we have $y_j > 0$ and $u_{max} = u_i > u_j$. Solution y' as defined in Claim 12.1 has a lower objective value than y.*

Proof. Consider y_i and y_j to be the length of the legs of a right triangle (possibly of 0 area). Then

$$\begin{aligned} & y_i + y_j \\ \ge & \begin{Bmatrix} y_i, y_j \ge 0, \\ \text{Pythagorean Theorem and Triangle Inequality} \end{Bmatrix} \\ & \sqrt{y_i^2 + y_j^2} \\ > & \{u_i > u_j \wedge y_j > 0\} \\ & \sqrt{y_i^2 + \frac{u_j}{u_i} y_j^2}. \end{aligned}$$

Thus, the objective value of y' is

$$\left.\begin{aligned} & \sum_{\tau_k \in \tau} y'_k \\ &= y'_i + y'_j + \sum_{\tau_k \in \tau \setminus \{\tau_i, \tau_j\}} y'_k \\ &= y'_i + y'_j + \sum_{\tau_k \in \tau \setminus \{\tau_i, \tau_j\}} y_k \\ &= \sqrt{y_i^2 + \frac{u_j}{u_i} y_j^2} + 0 + \sum_{\tau_k \in \tau \setminus \{\tau_i, \tau_j\}} y_k \\ &< \sum_{\tau_k \in \tau} y_k \end{aligned}\right\} \tag{11}$$

■

Claim 12.3. *Let τ_i and τ_j be two tasks such that for some solution y, we have $y_i, y_j > 0$ and $u_{max} = u_i = u_j$. y' as defined in Claim 12.1 has a lower objective value than y.*

Proof. Consider y_i and y_j to be the non-zero length legs of a right triangle.

$$\begin{aligned}&y_i + y_j\\ > &\begin{Bmatrix} y_i, y_j > 0, \\ \text{Pythagorean Theorem and Triangle Inequality}\end{Bmatrix}\\ &\sqrt{y_i^2 + y_j^2}\\ = &\sqrt{y_i^2 + \frac{u_j}{u_i} y_j^2}\end{aligned}$$

Thus, the objective value of y' is then less than that of y by the same reasoning as (11). ■

Observe that any solution that is not the optimal solution described at the beginning of this proof can be improved by being modified as described by Claims 12.2 and 12.3. □

There are two remaining lemmas needed to prove Lemma 15 (required by Step 3). Lemma 13 establishes the value of $\text{Dev}_i(t)$ at time 0.

Lemma 13. *For any task τ_i, $Dev_i(0) = 0$.*

Proof. At time 0, by Def. 7, $vt_i(0) = r_i(0) + T_i \frac{C_i(0) - c_i(0)}{C_i(0)}$. Because at time 0, the current job of τ_i is $\tau_{i,1}$ and $\tau_{i,1}$ has not yet executed, we have $vt_i(0) = r_{i,1} + T_i \frac{C_{i,1} - C_{i,1}}{C_{i,1}} = r_{i,1}$. Because $r_{i,1} \geq 0$, we have $vt_i(0) \geq 0$.

Because $vt_i(0) \geq 0$, we have $-vt_i(0) \leq 0$. By Def. 8, $\text{Dev}_i(0) = \max\{0, 0 - vt_i(0)\} = 0$. □

Lemma 14 establishes that $\text{Dev}_i(t)$ is always finite.

Lemma 14. *For any $t \geq 0$ and task τ_i, we have $Dev_i(t) \leq t$.*

Proof. By Def. 8, $\text{Dev}_i(t) = \max\{0, t - vt_i(t)\}$. If $t - vt_i(t) \leq 0$, then $\text{Dev}_i(t) = 0 \leq t$.

Otherwise, $\text{Dev}_i(t) = t - vt_i(t)$. By Lemma 4, $vt_i(t) + T_i \geq d_i(t) = r_i(t) + T_i$. Because jobs are not released prior to time 0, $vt_i(t) \geq r_i(t) \geq 0$. Thus, $\text{Dev}_i(t) = t - vt_i(t) \leq t$. □

We can now present our sufficient condition and prove that an invariant on squares of deviations is maintained if it is true (note that x' is indexed by task and processor while ℓ is scalar).

$$\left.\begin{aligned} \exists x' \geq 0, \ell \in (0,1) : \forall\ \tau_i \in \tau : \sum_{\pi_j \in \pi} s_{i,j} x'_{i,j} \geq u_i \\ \forall\ \tau_i \in \tau : \sum_{\pi_j \in \pi} x'_{i,j} = 1 - \ell \\ \forall\ \pi_j \in \pi : \sum_{\tau_i \in \tau} x'_{i,j} = 1 - \ell \end{aligned}\right\} \quad (12)$$

Note that were we to allow $\ell = 0$ in the latter two constraints, (12) would be equivalent to the feasibility condition for any scheduler on unrelated multiprocessors [2].

Lemma 15. *For any $\Delta > 0$, let*

$$K \triangleq u_{max} \left(\frac{2nT_{max}s_{max} + \Delta}{\ell u_{min}} \right)^2. \quad (13)$$

Under Unr-EDF, if $\exists x', \ell$ such that (12) is satisfied, then for any time $t \geq 0$,

$$\sum_{\tau_i \in \tau} u_i(Dev_i(t))^2 \leq K. \quad (14)$$

Proof. We prove the lemma by contradiction. Suppose otherwise that there exist time instants such that (14) does not hold. By Lemma 13, (14) holds at time 0. Let t_b be the last time instant such that (14) holds over $[0, t_b)$. In other words,

$$\forall t \in [0, t_b) : \sum_{\tau_i \in \tau} u_i(\text{Dev}_i(t))^2 \leq K \quad (15)$$

$$\forall \delta > 0 : \exists \epsilon \in [0, \delta) : \sum_{\tau_i \in \tau} u_i(\text{Dev}_i(t_b + \epsilon))^2 > K \quad (16)$$

Claim 15.1. $\sum_{\tau_i \in \tau} u_i(Dev_i(t_b))^2 = K$.

Proof. We prove the claim by contradiction. Suppose otherwise that $\sum_{\tau_i \in \tau} u_i(\text{Dev}_i(t_b))^2 \neq K$.

Case 15.1.1. $\sum_{\tau_i \in \tau} u_i(Dev_i(t_b))^2 < K$

Let $L = K - \sum_{\tau_i \in \tau} u_i(\text{Dev}_i(t_b))^2 > 0$ and let

$$\delta' \triangleq \sqrt{t_b^2 + L/(nu_{\text{max}})} - t_b > 0. \quad (17)$$

By (16), $\exists \epsilon \in [0, \delta') : \sum_{\tau_i \in \tau} u_i(\text{Dev}_i(t_b + \epsilon))^2 > K$. Thus,

$$\begin{aligned} &\sum_{\tau_i \in \tau} u_i \left[(\text{Dev}_i(t_b + \epsilon))^2 - (\text{Dev}_i(t_b))^2 \right] \\ &> K - \sum_{\tau_i \in \tau} u_i(\text{Dev}_i(t_b))^2 = L. \end{aligned}$$

Because the maximum of a finite set of reals is at least the mean, $\exists \tau_i \in \tau : u_i \left[(\text{Dev}_i(t_b + \epsilon))^2 - (\text{Dev}_i(t_b))^2 \right] > L/n$. Dividing both sides by u_i and factoring the left-hand side of the above yields $[\text{Dev}_i(t_b + \epsilon) + \text{Dev}_i(t_b)]\,[\text{Dev}_i(t_b + \epsilon) - \text{Dev}_i(t_b)] > L/(nu_i)$. Note that because $L/(nu_i) > 0$ and by Def. 8, $\text{Dev}_i(t_b + \epsilon) + \text{Dev}_i(t_b) \geq 0$ holds, we have $\text{Dev}_i(t_b + \epsilon) - \text{Dev}_i(t_b) > 0$. Thus, by Lemma 14,

$$\begin{aligned} &[2t_b + \epsilon]\,[\text{Dev}_i(t_b + \epsilon) - \text{Dev}_i(t_b)] > L/(nu_i) \\ \Rightarrow &\{\epsilon < \delta'\} \\ &[2t_b + \delta']\,[\text{Dev}_i(t_b + \epsilon) - \text{Dev}_i(t_b)] > L/(nu_i) \\ \Rightarrow &\text{Dev}_i(t_b + \epsilon) - \text{Dev}_i(t_b) > \frac{L}{nu_i(2t_b + \delta')}. \end{aligned}$$

Because $\text{Dev}_i(t_b + \epsilon) - \text{Dev}_i(t_b) > 0$ and $\text{Dev}_i(t_b) \geq 0$ (by Def. 8), $\text{Dev}_i(t_b + \epsilon) > 0$. By Def. 8, $\text{Dev}_i(t_b + \epsilon) - \text{Dev}_i(t_b) = t_b + \epsilon - vt_i(t_b + \epsilon) - t_b + vt_i(t_b)$. Thus, $t_b + \epsilon - vt_i(t_b + \epsilon) - t_b + vt_i(t_b) > \frac{L}{nu_i(2t_b + \delta')}$. Rearrangement yields

$$
\begin{aligned}
& vt_i(t_b+\epsilon) - vt_i(t_b) \\
< {} & \epsilon - \frac{L}{nu_i(2t_b+\delta')} \\
< {} & \{\epsilon < \delta'\} \\
& \delta' - \frac{L}{nu_i(2t_b+\delta')} \\
= {} & \{t_b, \delta' > 0 \Rightarrow 2t_b + \delta' \neq 0\} \\
& \frac{1}{2t_b+\delta'}\left[\delta'(2t_b+\delta') - \frac{L}{nu_i}\right] \\
= {} & \{\text{By (17)}\} \\
& \frac{1}{2t_b+\delta'}\left[\begin{array}{l} \left(\sqrt{t_b^2 + L/(nu_{\max})} - t_b\right) \\ \times\left(\sqrt{t_b^2 + L/(nu_{\max})} + t_b\right) \\ -\dfrac{L}{nu_i} \end{array}\right] \\
= {} & \frac{1}{2t_b+\delta'}\left[t_b^2 + \frac{L}{nu_{\max}} - t_b^2 - \frac{L}{nu_i}\right] \\
\leq {} & \{u_i \leq u_{\max}\} \\
& 0.
\end{aligned}
$$

This contradicts Lemma 3.

Case 15.1.2. $\sum_{\tau_i \in \tau} u_i(Dev_i(t_b))^2 > K$

The reasoning of Case 15.1.2 is fairly similar to that of Case 15.1.1 in that we prove $vt_i(t)$ must have decreased for some task τ_i for this case to have occurred, thereby contradicting Lemma 3. We defer the reasoning for Case 15.1.2 to the appendix.

In either case, we have a contradiction. Thus, $\sum_{\tau_i \in \tau} u_i(\text{Dev}_i(t_b))^2 = K$. ■

Claim 15.2. *Unr-EDF is non-fluid.*

This claim follows from the fact that Unr-EDF only reschedules at job completions and pseudo-releases. A formal proof is provided in the appendix.

Claim 15.3. $\sum_{\tau_i \in \tau} Dev_i(t_b) \geq \sqrt{\frac{K}{u_{max}}} = \frac{2nT_{max}s_{max}+\Delta}{\ell u_{min}}$.

Proof. Consider the optimization problem in Lemma 12. Because $\text{Dev}_i(t_b) \geq 0$ (by Def. 8), (10), Claim 15.1, and (9), letting $y_i = \text{Dev}_i(t_b)$ for each task $\tau_i \in \tau$ is a solution to this optimization problem. Because an optimal solution must have a lower or equal objective function value than any other solution, $\sum_{\tau_i \in \tau} \text{Dev}_i(t_b) = \sum_{\tau_i \in \tau} y_i \leq \sqrt{K/u_{\max}}$. By (13), $\sqrt{K/u_{\max}} = (2nT_{\max}s_{\max} + \Delta)/(\ell u_{\min})$. ■

Claim 15.4. *At time t_b,*

$$\sum_{\tau_i \in \tau} Dev_i(t_b)s_i(t_b) > \Delta + \sum_{\tau_i \in \tau} Dev_i(t_b)u_i. \tag{18}$$

Proof. Consider the values of $x'/(1-\ell)$. By (12), we have $\forall \tau_i \in \tau : \sum_{\pi_j \in \pi} x'_{i,j}/(1-\ell) = 1$, $\forall \pi_j \in \pi : \sum_{\tau_i \in \tau} x'_{i,j}/(1-\ell) = 1$, and $x'/(1-\ell) \geq 0$.

Thus, $x'/(1-\ell)$ is a fractional solution (i.e., when $x_{i,j} \in \{0,1\}$ in (4) is relaxed to $x_{i,j} \geq 0$) of (4). Let x^b be the optimal solution of (4) at time t_b used to assign tasks to processors. Because (4) is an instance of the assignment problem (2) with $w_{i,j} = \Phi_i(t_b)\sum_{\pi_j \in \pi} s_{i,j}$, by Theorem 1, the optimum value obtained by x^b for (4) is at least as large as the value obtained by (fractional) solution $x'/(1-\ell)$. Thus,

$$
\begin{aligned}
& \sum_{\tau_i \in \tau} \Phi_i(t_b) \sum_{\pi_j \in \pi} s_{i,j}x^b_{i,j} \\
\geq {} & \sum_{\tau_i \in \tau} \Phi_i(t_b) \sum_{\pi_j \in \pi} s_{i,j}x'_{i,j}/(1-\ell).
\end{aligned} \tag{19}
$$

By (12), $\forall \tau_i \in \tau : \sum_{\pi_j \in \pi} s_{i,j}x'_{i,j}/(1-\ell) \geq u_i/(1-\ell)$. Because $\Phi_i(t_b) \geq 0$ (by Lemma 10), multiplying both sides by $\Phi_i(t_b)$ and summing over all tasks yields $\sum_{\tau_i \in \tau} \Phi_i(t_b) \sum_{\pi_j \in \pi} s_{i,j}x'_{i,j}/(1-\ell) \geq \sum_{\tau_i \in \tau} \Phi_i(t_b)u_i/(1-\ell)$. Thus, by (19),

$$\sum_{\tau_i \in \tau} \Phi_i(t_b) \sum_{\pi_j \in \pi} s_{i,j}x^b_{i,j} \geq \sum_{\tau_i \in \tau} \Phi_i(t_b)u_i/(1-\ell). \tag{20}$$

By (4), the definition of assignment, and the definition of $s_i(t)$, we have $\sum_{\pi_j \in \pi} s_{i,j}x^b_{i,j} = s_i(t_b)$. Because $\Phi_i(t_b) = \text{Dev}_i(t_b) + (\Phi_i(t_b) - \text{Dev}_i(t_b))$, by (20), we have

$$
\begin{aligned}
& \sum_{\tau_i \in \tau} \text{Dev}_i(t_b)s_i(t_b) \\
\geq {} & \sum_{\tau_i \in \tau} \text{Dev}_i(t_b)u_i/(1-\ell) \\
& + \sum_{\tau_i \in \tau} (\Phi_i(t_b) - \text{Dev}_i(t_b))(u_i/(1-\ell) - s_i(t_b)) \\
\geq {} & \begin{Bmatrix} \text{By Lemma 11, } \Phi_i(t_b) - \text{Dev}_i(t_b) \geq 0. \\ u_i > 0 \wedge \ell \in (0,1) \Rightarrow u_i/(1-\ell) > 0 \end{Bmatrix} \\
& \sum_{\tau_i \in \tau} \text{Dev}_i(t_b)u_i/(1-\ell) - \sum_{\tau_i \in \tau} (\Phi_i(t_b) - \text{Dev}_i(t_b))s_i(t_b) \\
= {} & \sum_{\tau_i \in \tau} \text{Dev}_i(t_b)u_i + \frac{\ell}{1-\ell}\sum_{\tau_i \in \tau} \text{Dev}_i(t_b)u_i \\
& - \sum_{\tau_i \in \tau} (\Phi_i(t_b) - \text{Dev}_i(t_b))s_i(t_b) \\
\geq {} & \{\text{By Lemma 11 and } s_i(t_b) \leq s_{\max}\} \\
& \sum_{\tau_i \in \tau} \text{Dev}_i(t_b)u_i + \frac{\ell}{1-\ell}\sum_{\tau_i \in \tau} \text{Dev}_i(t_b)u_i \\
& - \sum_{\tau_i \in \tau} 2T_{\max}s_{\max} \\
= {} & \{|\tau| = n\} \\
& \sum_{\tau_i \in \tau} \text{Dev}_i(t_b)u_i + \frac{\ell}{1-\ell}\sum_{\tau_i \in \tau} \text{Dev}_i(t_b)u_i - 2nT_{\max}s_{\max} \\
> {} & \{\text{By (12), } \ell \in (0,1) \wedge u_i \geq u_{\min}\} \\
& \sum_{\tau_i \in \tau} \text{Dev}_i(t_b)u_i + \ell u_{\min}\sum_{\tau_i \in \tau} \text{Dev}_i(t_b) - 2nT_{\max}s_{\max}
\end{aligned}
$$

$$\geq \{\text{By Claim 15.3}\}$$
$$\sum_{\tau_i \in \tau} \mathrm{Dev}_i(t_b) u_i + 2nT_{\max} s_{\max} + \Delta - 2nT_{\max} s_{\max}$$
$$= \Delta + \sum_{\tau_i \in \tau} \mathrm{Dev}_i(t_b) u_i. \qquad \blacksquare$$

By Claim 15.2, we have that Unr-EDF is non-fluid; by Claim 15.3, we have that $\sum_{\tau_i \in \tau} \mathrm{Dev}_i(t_b) > 0$; by Claim 15.4, we have (18). Thus, by Lemma 7, we have

$$\exists \delta > 0 : \forall \epsilon \in [0, \delta) :$$
$$\sum_{\tau_i \in \tau} u_i(\mathrm{Dev}_i(t_b))^2 > \sum_{\tau_i \in \tau} u_i(\mathrm{Dev}_i(t_b + \epsilon))^2.$$

However, by Claim 15.1 and (16), we have

$$\forall \delta > 0 : \exists \epsilon \in [0, \delta) :$$
$$\sum_{\tau_i \in \tau} u_i(\mathrm{Dev}_i(t_b))^2 < \sum_{\tau_i \in \tau} u_i(\mathrm{Dev}_i(t_b + \epsilon))^2.$$

Thus, the existence of the time instant t_b is a contradiction. Thus, (14) is maintained for all $t \geq 0$. □

Proving Lemma 15 completes Step 3. Step 4 (proving tardiness bounds) is a straightforward with Lemmas 5 and 15.

Theorem 2. *Under Unr-EDF, if $\exists x', \ell$ such that (12) is satisfied, then the tardiness of any task τ_i is at most*

$$\sqrt{\frac{u_{max}}{u_i}} \frac{2nT_{max} s_{max}}{\ell u_{min}}. \tag{21}$$

Proof. By Lemma 15, for any time $t \geq 0$ and $\Delta > 0$, we have $\sum_{\tau_i \in \tau} u_i(\mathrm{Dev}_i(t))^2 \leq u_{\max} \left(\frac{2nT_{\max} s_{\max} + \Delta}{\ell u_{\min}}\right)^2$. Because for any task τ_i, we have $\mathrm{Dev}_i(t) \geq 0$ (by Def. 8), we have for each i that $u_i(\mathrm{Dev}_i(t))^2 \leq u_{\max} \left(\frac{2nT_{\max} s_{\max} + \Delta}{\ell u_{\min}}\right)^2$. Thus, we have $\mathrm{Dev}_i(t) \leq \sqrt{\frac{u_{\max}}{u_i}} \frac{2nT_{\max} s_{\max} + \Delta}{\ell u_{\min}}$.

By Lemma 5, the tardiness of τ_i is therefore at most $\sqrt{\frac{u_{\max}}{u_i}} \frac{2nT_{\max} s_{\max} + \Delta}{\ell u_{\min}}$. This value approaches (21) in the limit as we allow our choice of $\Delta \to 0$. □

V. Evaluation

To evaluate our tardiness bound, we simulated Unr-EDF on randomly generated task systems and multiprocessors in Python. The source code of the simulation is provided online [24]. This simulation implements the incremental algorithm discussed in Sec. III-C.

We generated task systems of sizes $n = \{20, 40, 80\}$, with $\{4, 8\}$ processors (the number of processors was increased to n by the techniques discussed in Sec. II). We also considered values of ℓ ranging from $\{1/2, 1/4, 1/8, \ldots, 1/256\}$. Processor speeds for each task were sampled uniformly from [0.0, 1.0). Utilizations were generated to match given ℓ values by solving a maximization linear program with constraints taken from (12) with decision variables x' and u. The objective function of was a linear combination of the elements of u, with coefficients sampled uniformly from [0.0, 1.0). Periods were

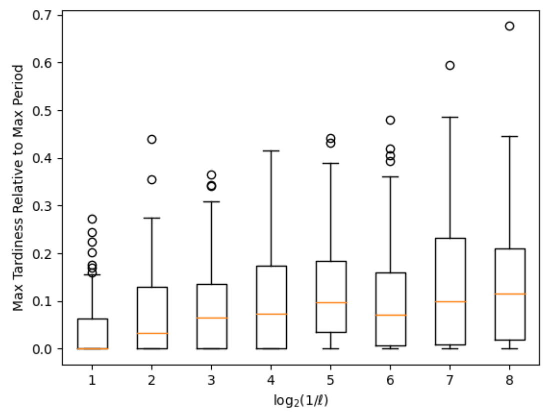


Fig. 4: Plot of tardiness against ℓ for 40 tasks and 4 processors.

then sampled uniformly from [10, 100]. 100 task systems and multiprocessors were generated for each triplet of task count, processor count, and ℓ value. For each generated system, tardiness of tasks under Unr-EDF with periodic releases was measured for 100,000 simulated time units.

For each pair of task and processor counts, we plotted the maximum tardiness relative to $T_{\max}$ of each task system against ℓ. An example graph is presented in Fig. 4 (ℓ halves at each step from left to right). Boxplots illustrate the quartiles and outliers of tardiness for each ℓ. In Fig. 4, as well as the other generated graphs, it can be observed that, while tardiness increases as $\ell \to 0$, tardiness does not scale inversely with ℓ (unlike our analytical bound in (21)). All observed task systems suffered tardiness at worst $T_{\max}$, with a majority suffering a small fraction of $T_{\max}$.

While this suggests that our analysis is fundamentally pessimistic and Unr-EDF may actually be SRT-optimal, this is not conclusive evidence. It has always been the case, even for standard EDF on identical multiprocessors [5], that the tardiness of randomly generated task systems tends to be lower than the worst-case tardiness of hand-crafted task systems. Unfortunately, the complexity of Unr-EDF and, more generally, of tracking remaining execution requirements of jobs under unrelated multiprocessors seem to make computing schedules by hand intractable. For now, this has left simulation as our only approach for counterexample searching.

VI. Conclusion

In this work, we have designed a new EDF variant Unr-EDF for unrelated multiprocessors. We have proven that existing SRT-optimal EDF variants are special cases of Unr-EDF. We have proven that Unr-EDF is at least nearly SRT-optimal and have shown in simulation that tardiness under Unr-EDF for randomly generated task systems is reasonable.

Topics of future work include refining the analysis of this work to either prove full SRT-optimality with improved tardiness bounds or demonstrate the existence of counterexamples with unbounded tardiness. Additionally, we will investigate how Unr-EDF (or special cases of the algorithm) might be practically implemented for unrelated multiprocessors.

REFERENCES

[1] S. K. Baruah, V. Bonifaci, R. Bruni, and A. Marchetti-Spaccamela, "ILP-based approaches to partitioning recurrent workloads upon heterogeneous multiprocessors," in *Euromicro Conference on Real-Time Systems (ECRTS)*, 2016, pp. 215–225.

[2] S. K. Baruah, "Feasibility analysis of preemptive real-time systems upon heterogeneous multiprocessor platforms," in *IEEE Real-Time Systems Symposium (RTSS)*, 2004, pp. 37–46.

[3] H. S. Chwa, J. Seo, J. Lee, and I. Shin, "Optimal real-time scheduling on two-type heterogeneous multicore platforms," in *IEEE Real-Time Systems Symposium (RTSS)*, 2015, pp. 119–129.

[4] B. B. Brandenburg, "Scheduling and locking in multiprocessor real-time operating systems," Ph.D. dissertation, University of North Carolina, Chapel Hill, NC, 2011.

[5] U. M. C. Devi and J. H. Anderson, "Tardiness bounds under global EDF scheduling on a multiprocessor," in *IEEE Real-Time Systems Symposium (RTSS)*, 2005, pp. 12 pp.–341.

[6] J. Lelli, C. Scordino, L. Abeni, and D. Faggioli, "Deadline scheduling in the Linux kernel," *Softw. Pract. Exper.*, vol. 46, no. 6, p. 821–839, Jun. 2016.

[7] "Deadline task scheduling," https://github.com/torvalds/linux/blob/master/Documentation/scheduler/sched-deadline.rst, 2018, online; accessed 03 June 2020.

[8] K. Yang and J. H. Anderson, "On the soft real-time optimality of global EDF on uniform multiprocessors," in *IEEE Real-Time Systems Symposium (RTSS)*, 2017, pp. 319–330.

[9] F. Cerqueira, A. Gujarati, and B. B. Brandenburg, "Linux's processor affinity API, refined: Shifting real-time tasks towards higher schedulability," in *IEEE Real-Time Systems Symposium (RTSS)*, 2014, pp. 249–259.

[10] S. Tang, S. Voronov, and J. H. Anderson, "GEDF tardiness: Open problems involving uniform multiprocessors and affinity masks resolved," in *Euromicro Conference on Real-Time Systems (ECRTS)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), S. Quinton, Ed., vol. 133. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, pp. 13:1–13:21.

[11] I. Toroslu and G. Üçoluk, "Incremental assignment problem," *Information Sciences*, vol. 177, pp. 1523–1529, 03 2007.

[12] S. Tang and J. H. Anderson, "Towards practical multiprocessor EDF with affinities," in *IEEE Real-Time Systems Symposium (RTSS)*. Los Alamitos, CA, USA: IEEE Computer Society, dec 2020, pp. 89–101.

[13] K. Albers and F. Slomka, "An event stream driven approximation for the analysis of real-time systems," in *Euromicro Conference on Real-Time Systems (ECRTS)*, 2004, pp. 187–195.

[14] N. Fisher and S. K. Baruah, "A fully polynomial-time approximation scheme for feasibility analysis in static-priority systems with arbitrary relative deadlines," in *Euromicro Conference on Real-Time Systems (ECRTS)*, 2005, pp. 117–126.

[15] V. Bonifaci, B. B. Brandenburg, G. DAngelo, and A. Marchetti-Spaccamela, "Multiprocessor real-time scheduling with hierarchical processor affinities," in *Euromicro Conference on Real-Time Systems (ECRTS)*, July 2016, pp. 237–247.

[16] S. Tang, J. H. Anderson, and L. Abeni, "On the defectiveness of sched_deadline w.r.t. tardiness and affinities, and a partial fix," in *International Conference on Real-Time Networks and Systems (RTNS)*, 2021.

[17] K. Yang and J. H. Anderson, "On the soft real-time optimality of global EDF on multiprocessors: From identical to uniform heterogeneous," in *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2015, pp. 1–10.

[18] G. Tong and C. Liu, "Supporting soft real-time sporadic task systems on uniform heterogeneous multiprocessors with no utilization loss," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 9, pp. 2740–2752, 2016.

[19] J. Erickson, U. M. C. Devi, and S. K. Baruah, "Improved tardiness bounds for global EDF," in *Euromicro Conference on Real-Time Systems (ECRTS)*, 2010, pp. 14–23.

[20] J. Erickson, J. Anderson, and B. Ward, "Fair lateness scheduling: Reducing maximum lateness in G-EDF-like scheduling," *Real-Time Systems*, vol. 50, 07 2014.

[21] P. Valente, "Using a lag-balance property to tighten tardiness bounds for global EDF," *Real-Time Systems*, vol. 52, 08 2015.

[22] J. Matoušek and B. Gärtner, *Understanding and Using Linear Programming*. Springer, 01 2007.

[23] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *J. ACM*, vol. 19, no. 2, p. 248–264, Apr. 1972.

[24] S. Tang, S. Voronov, and J. H. Anderson, "Extending EDF for soft real-time scheduling on unrelated multiprocessors," Full version of this paper, available at http://jamesanderson.web.unc.edu/papers/, 2021.

APPENDIX

OMITTED PROOFS

Proof of Lemma 3.

Proof. There are two cases: the same job of τ_i is current at t and $t+\epsilon$ or the current job of τ_i at t is not current at $t+\epsilon$.

Case 3.1. *The same job of τ_i is current at t and $t+\epsilon$.*

Call this job $\tau_{i,j}$. By Def. 7, $vt_i(t) = r_{i,j} + T_i\frac{C_{i,j}-c_i(t)}{C_{i,j}}$ and $vt_i(t+\epsilon) = r_{i,j} + T_i\frac{C_{i,j}-c_i(t+\epsilon)}{C_{i,j}}$. Thus, $vt_i(t+\epsilon) - vt_i(t) = T_i\frac{C_{i,j}-c_i(t+\epsilon)-C_{i,j}+c_i(t)}{C_{i,j}} = T_i(c_i(t)-c_i(t+\epsilon))/C_{i,j}$. Because the remaining execution required by $\tau_{i,j}$ cannot increase with time, we have $c_i(t)-c_i(t+\epsilon) \geq 0$. Thus, $vt_i(t+\epsilon)-vt_i(t) \geq 0$.

Case 3.2. *The current job of τ_i at t is not current by $t+\epsilon$.*

Let $\tau_{i,j}$ and $\tau_{i,j'}$ with $j' > j$ denote the jobs current at t and $t+\epsilon$, respectively. By Def. 1, $c_i(t) > 0$. Thus, by Def. 7, we have $vt_i(t) < r_{i,j} + T_i$; likewise, because $c_i(t+\epsilon) \leq C_i(t+\epsilon)$, we have $r_{i,j'} \leq vt_i(t+\epsilon)$. Because $j' > j$, we have $r_{i,j'} \geq r_{i,j} + T_i$, and thus, $vt_i(t+\epsilon) > vt_i(t)$.

In either case, we have $vt_i(t+\epsilon) \geq vt_i(t)$. □

Proof of Claim 15.2.

Proof. Unr-EDF only reschedules when the decision variables $x_{i,j}$ of (4) change value. Because we assume τ, π, and $s_{i,j}$ are fixed values, these $x_{i,j}$ variables only change value when $d_i(t)$ or $D_i(t)$ changes, which can only occur at job completions and pseudo-releases.

For any time t, there are finitely many job completions and pseudo-releases in the interval $[0, t')$ for any $t' \in (t, \infty)$. Thus, there are finitely many time instants in which a reschedule occurs in $[0, t')$. Suppose there are k such distinct time instants and denote them $t_1, t_2, \ldots, t_k$ such that $t_1 < t_2 < \cdots < t_k$. The intervals $[0, t_1)$, $[t_1, t_2)$, $[t_2, t_3)$, $\ldots$, $[t_k - 1, t_k)$, $[t_k, t')$ partition $[0, t')$ and rescheduling does not occur within any such interval. Because $t \in [0, t')$, time t is within one of these intervals. By Def. 9, Unr-EDF is non-fluid. □

Proof of Case 15.1.2.

The reasoning of this case is fairly similar to that of Case 15.1.1. Let $L = \sum_{\tau_i \in \tau} u_i(\text{Dev}_i(t_b))^2 - K > 0$ and let

$$\epsilon \in (0, \min\{t_b, L/(2nu_{\max}t_b)\}). \tag{22}$$

By (15), $\sum_{\tau_i \in \tau} u_i(\text{Dev}_i(t_b - \epsilon))^2 \leq K$. Thus,

$$\begin{aligned}&\sum_{\tau_i \in \tau} u_i(\text{Dev}_i(t_b))^2 - \sum_{\tau_i \in \tau} u_i(\text{Dev}_i(t_b - \epsilon))^2 \\ \geq &\sum_{\tau_i \in \tau} u_i(\text{Dev}_i(t_b))^2 - K = L.\end{aligned}$$

Because the maximum of a finite set of reals is at least the mean, $\exists \tau_i \in \tau : u_i\left[(\text{Dev}_i(t_b))^2 - (\text{Dev}_i(t_b - \epsilon))^2\right] \geq L/n$. Dividing both sides by u_i and factoring the left hand side yields $[\text{Dev}_i(t_b) + \text{Dev}_i(t_b - \epsilon)]\,[\text{Dev}_i(t_b) - \text{Dev}_i(t_b - \epsilon)] \geq L/(nu_i)$. Because $L/(nu_i) > 0$ and $\text{Dev}_i(t_b) + \text{Dev}_i(t_b - \epsilon) \geq 0$ by Def. 8, $\text{Dev}_i(t_b) - \text{Dev}_i(t_b - \epsilon) > 0$. Thus, by Lemma 14,

$$\begin{aligned}&[2t_b - \epsilon]\,[\text{Dev}_i(t_b) - \text{Dev}_i(t_b - \epsilon)] \geq L/(nu_i) \\ \Rightarrow &\{\text{By (22), } 0 < \epsilon < t_b\} \\ &\text{Dev}_i(t_b) - \text{Dev}_i(t_b - \epsilon) > \frac{L}{2nu_it_b} \\ \Rightarrow &\; t_b - vt_i(t_b) - t_b + \epsilon + vt_i(t_b - \epsilon) > \frac{L}{2nu_it_b} \\ \Rightarrow &\; vt_i(t_b) - vt_i(t_b - \epsilon) < \epsilon - \frac{L}{2nu_it_b}\end{aligned}$$

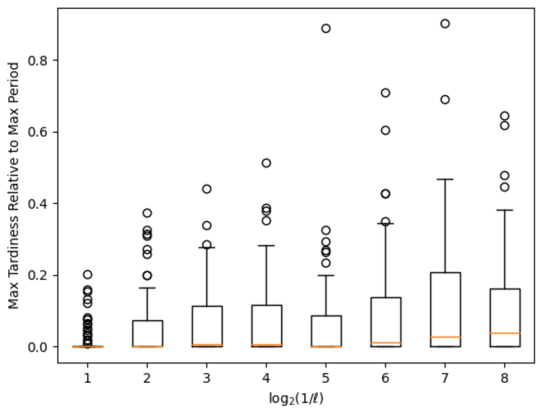
By (22), $\epsilon < L/(2nu_{\max}t_b)$. Thus,

$$\begin{aligned}&vt_i(t_b) - vt_i(t_b - \epsilon) \\ < &\frac{L}{2nu_{\max}t_b} - \frac{L}{2nu_it_b} \\ \leq &\{u_{\max} > u_i\} \\ &0.\end{aligned}$$

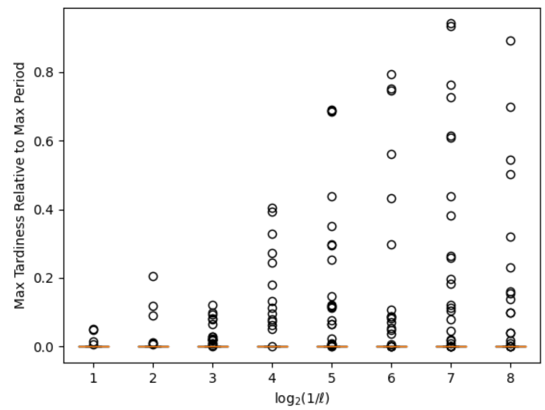
This contradicts Lemma 3.

ADDITIONAL PLOTS

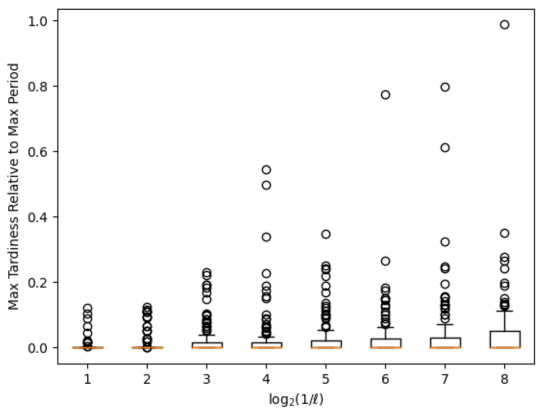
Additional plots of simulated tardiness are presented in Fig. 5.



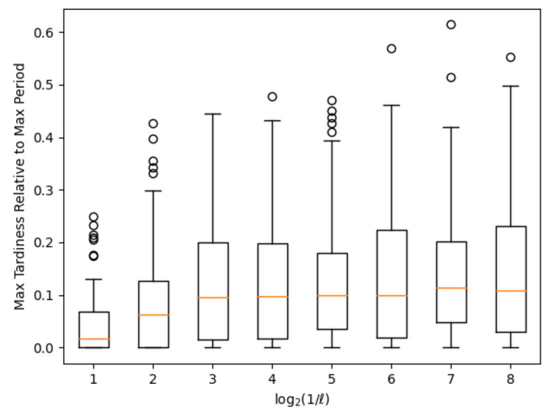
(a) 20 tasks and 4 processors.



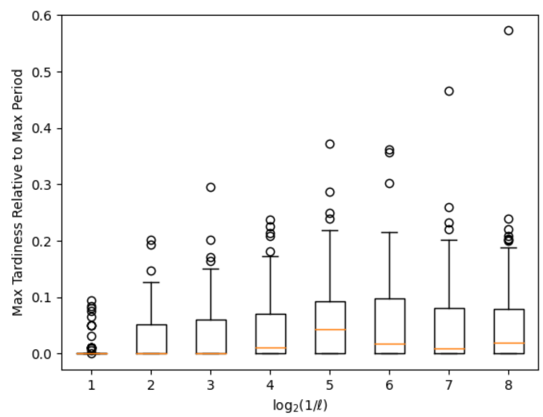
(b) 20 tasks and 8 processors.



(c) 40 tasks and 8 processors.



(d) 80 tasks and 4 processors.



(e) 80 tasks and 8 processors.

Fig. 5: Omitted tardiness plots.