An Optimal Semi-Partitioned Scheduler for Uniform Heterogeneous Multiprocessors *

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

Abstract

A semi-partitioned scheduler called EDF-tu is presented that is the first such scheduler to be optimal on uniform heterogeneous multiprocessors. EDF-tu utilizes an adjustable allocation parameter called a frame *to schedule tasks that migrate. The frame size F must divide all task periods to ensure hard real-time optimality, but for any choice of F, maximum deadline tardiness is at most F. Thus, the proper selection of F hinges on runtime overheads (which are higher when F is smaller) and the strength of the real-time guarantee desired. When determining which tasks must migrate, new issues specific to heterogeneous platforms arise that have not been explored before. It is shown via counterexamples that resolving such issues differently from EDF-tu can render feasible task systems unschedulable.*

1 Introduction

Heterogeneous computing platforms have processing elements that may differ with respect to execution speed or functionality. Such differences can be exploited in the design of real-time systems to achieve a variety of goals. For example, ARM's big.LITTLE multicore architecture [6] enables goals related to performance/energy tradeoffs to be flexibly addressed by providing two categories of cores: relatively slower, low-power ones and faster, high-power ones.

There is currently great interest in industry in exploiting such flexibility in fielded commercial products where real-time constraints exist. Unfortunately, while significant progress has been made in work on real-time resource-allocation techniques for homogeneous multiprocessors, the same is not true of heterogeneous ones. This is not surprising: on a heterogeneous platform, *choices* must be made when selecting the hardware component(s) upon which a task will execute. The need to resolve such choices can greatly complicate resource allocation.

In this paper, we expand upon the known body of work directed at real-time heterogeneous platforms by presenting the first semi-partitioned scheduler that is optimal[1] on an important category of such platforms, namely uniform ones. On a *uniform* platform, processors differ only with respect to speed. *Semi-partitioned* schedulers [1] are a hybrid between pure global and partitioned schedulers wherein migration is allowed for only a limited number of tasks, with the remaining ones being fixed to processors. A semi-partitioned scheduler can enable a wider range of systems to be supported than is possible under pure partitioning, but without incurring excessive migration costs. We describe the new scheduler proposed herein in greater detail below, after first presenting a brief overview of prior related work.

Related work. The first proposed semi-partitioned scheduler, EDF-fm [1], was designed for soft real-time (SRT) systems, where the SRT constraint of interest is that deadline tardiness is bounded (all references to SRT in this paper are with respect to this definition). EDF-fm requires utilization constraints that render it non-optimal for SRT systems. Such constraints were eliminated in the recently proposed EDF-os [2], which is the first semi-partitioned scheduler that is SRT-optimal. In work on hard real-time (HRT) systems, a variety of semi-partitioned schedulers have been proposed [3, 4, 5, 7, 8, 9, 10, 11, 14, 15, 17, 18, 19, 22]. Of these, only EKG [4] is HRT-optimal, and only for periodic task systems. EKG is optimal only when a configurable parameter is reduced in a way that increases preemption frequency. The same is true of the scheduler proposed herein.

All of the work cited above pertains to homogeneous multiprocessors. Funk and colleagues were the first to study the real-time scheduling of sporadic task systems on heterogeneous multiprocessors, as summarized in her dissertation [12]. However, this work pre-dated the advent of work on semi-partitioned schedulers. Variants of EDF-fm and EDF-os have been were proposed for SRT heterogeneous systems [20, 23], but these variants are not SRT-optimal. As explained later, heterogeneous platforms require a more complicated condition for identifying feasible task systems than homogeneous ones, and the analysis in these papers does not incorporate this more complicated condition.

Contributions. We develop an earliest-deadline-first-based semi-partitioned scheduler, EDF-tu ("tu" stands for "tunable scheduler for uniform platforms"), to schedule implicit-deadline sporadic task systems on uniform heterogeneous multiprocessors. EDF-tu uses an allocation parameter called a *frame* in managing migrating tasks. The frame size F is a tunable parameter. If F divides all task periods, then HRT optimality is ensured, but this comes at the expense of preemption frequencies that could be high for some systems. On the other hand, if only SRT schedulability is required, then F can be set to any value, and the deadline tardiness of any invocation of any task will be at most F. Thus, EDF-tu flexibly enables

*Work supported by NSF grants CNS 1115284, CNS 1218693, CPS 1239135, CNS 1409175, and CPS 1446631, AFOSR grant FA9550-14-1-0161, ARO grant W911NF-14-1-0499, and a grant from General Motors.

[1]The meaning of the term "optimal" is explained later in Sec. 2.

tradeoffs between preemption overheads and timeliness to be explored; we examine such tradeoffs both analytically and experimentally herein.

To the best of our knowledge, this is the first paper to consider such tradeoffs with respect to uniform heterogeneous platforms. Additionally, EDF-tu is only the second optimal algorithm proposed for scheduling sporadic task systems on such platforms—the only prior one is an algorithm proposed by Funk *et al.* [13] as a byproduct towards establishing a feasibility condition for such systems. That prior algorithm is not a semi-partitioned algorithm, and thus can have many more migrating tasks than EDF-tu. Thus, EDF-tu is a step forward with respect to practicality. Finally, this paper is the first to explore a number of issues pertaining to task assignments in semi-partitioned algorithms that are unique to heterogeneous platforms. In particular, we present counterexamples that show that assignment strategies that radically differ from that of EDF-tu can render feasible task systems unschedulable.

Organization. In the rest of this paper, we provide needed background (Sec. 2), develop conditions for obtaining feasible task assignments (Sec. 3), describe EDF-tu in detail (Sec. 4), present our optimality analysis (Sec. 5), examine alternative task-assignment strategies (Sec. 6), present an experimental evaluation (Sec. 7), and conclude (Sec. 8).

2 Background

We consider the scheduling of n implicit-deadline sequential sporadic tasks on m processors, where $n \geq m$ (we assume familiarity with these terms). We specify a task τ_i by (C_i, T_i), where C_i is its *worst-case execution requirement* and T_i is its *period*, and denote its *utilization* as

$$u_i = \frac{C_i}{T_i}. \tag{1}$$

On a heterogeneous platform, $u_i \leq 1$ does not necessarily hold. Needed restrictions on utilizations are given later in Sec. 2.1. We assume that time is continuous.

A *job* is an invocation of a task. If a job that has a deadline at time t_d and completes at time t_c, then its *tardiness* is defined as $\max\{0, t_c - t_d\}$. The tardiness of a *task* is the maximum tardiness of any of its jobs. A task system is *HRT-schedulable* (*SRT-schedulable*) under a given scheduling algorithm if each task can be guaranteed zero (bounded) tardiness under that algorithm. A task system is *HRT-feasible* (*SRT-feasible*) if it is HRT-schedulable (SRT-schedulable) under some scheduler. A given scheduler is *HRT-optimal* (*SRT-optimal*) if any HRT-feasible (SRT-feasible) task system is HRT-schedulable (SRT-schedulable) under it. Henceforth, references to the term "feasible" without qualification should be taken to mean "HRT-feasible."

A taxonomy of multiprocessors. The following taxonomy [12, 21] classifies multiprocessor platforms according to assumptions about processor speeds—the *speed* of a processor refers to the amount of work completed in one time unit when a job is executed on that processor.

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

On identical and uniform platforms, a processor's *capacity* is given by its speed, which can be thought of as the total available utilization that can be allocated.

Semi-partitioned scheduling. Under semi-partitioned scheduling, each task is allocated a non-zero *share* on certain processors such that the total allocated share on each processor does not exceed the processor's capacity and the total allocated share of a task matches its utilization. If a task has non-zero shares on only one (multiple) processor(s), then it is a *fixed* (*migrating*) task.

2.1 Uniform Heterogeneous Multiprocessors

In the rest of this paper, we consider a uniform heterogeneous multiprocessor system π, which has m processors. Processor p is identified by its speed s_p ($1 \leq p \leq m$, $s_p \in \mathbb{R}$). Also, we index the processors in non-increasing-speed order, *i.e.*, $\pi = \{s_1, s_2, \cdots, s_m\}$, where $s_p \geq s_{p+1}$ for $p \in \{1, 2, \cdots, m-1\}$. We consider scheduling a sporadic task set τ on π. We index the tasks in non-increasing-utilization order, *i.e.*, $\tau = \{\tau_1, \tau_2, \cdots, \tau_n\}$, where $u_i \geq u_{i+1}$ for $i \in \{1, 2, \cdots, n-1\}$.

Let $U_k = \sum_{i=1}^{k} u_i$, $S_k = \sum_{i=1}^{k} s_i$. Also, denote the total system utilization as $U_\tau = U_n$ and the total platform capacity as $S_\pi = S_m$. By leveraging the Level Algorithm [16], Funk *et al.* [13] showed that an implicit-deadline *periodic* task system τ is feasible on a uniform heterogeneous multiprocessor system π if and only if the following conditions hold.

$$U_\tau \leq S_\pi \tag{2}$$

$$U_k \leq S_k \quad \text{for} \quad k = 1, 2, \cdots, m-1 \tag{3}$$

In fact, the proof in [13] also shows that (2) and (3) are a necessary and sufficient feasibility condition (HRT or SRT) for implicit-deadline *sporadic* task systems.

2.2 Level Algorithm

The Level Algorithm was proposed by Horvath *et al.* [16] for scheduling a set of non-real-time 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 re-

maining 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 initial 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 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. 1(a) shows the resulting schedule by the Level Algorithm for this example. Fig. 1(b) shows the real schedule for "jointly executing." Fig. 1(c) shows the real schedule for the system. As seen in Fig. 1(d), when jobs start to jointly execute, we can make every processor involved in this joint execution start with its currently executing job to reduce unnecessary preemptions and migrations.

Theorem 1 (Theorem 1 in [16])**.** *Let $\mathcal{J} = \{J_1, J_2, \cdots, \mathcal{J}_n\}$ denote a set of independent non-real-time jobs to be scheduled on an m-processor uniform multiprocessor π. Let X_i denote the sum of i largest execution requirements in $\mathcal{J}$. Then the Level Algorithm constructs a minimum makespan, which is given by*

$$\max\left(\max_{1\le i\le m-1}\left\{\frac{X_i}{S_i}\right\}, \frac{X_n}{S_m}\right).$$

This is very similar to the feasibility condition given by (2) and (3), because that feasibility condition was, in fact, derived from the Level Algorithm [13].

3 Feasible Assignments

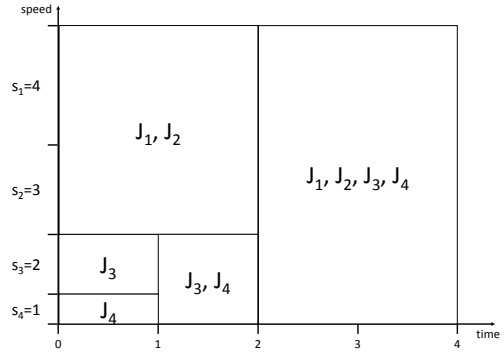
Most semi-partitioned scheduling algorithms are defined by specifying separate *assignment* and *execution* phases. In the former, per-processor shares are defined offline for each task, and fixed tasks are distinguished from migrating ones. In the latter, an actual schedule is produced at runtime, based on the task share assignments. In this section, we explore the problem of obtaining share assignments. We show that issues arise in the case of uniform heterogeneous platforms that have not been considered before.

In addressing such issues, we will need to examine situations where some number of tasks in the assignment process have been assigned as fixed. We let σ_i^f denote the sum of the utilizations of the fixed tasks on processor s_i, *i.e.*,

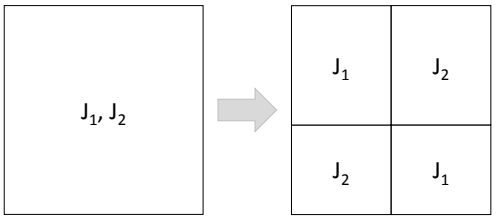
$$\sigma_i^f = \sum_{\substack{\tau_k \text{ is a fixed task} \\ \text{on processor } s_i}} u_k. \tag{4}$$

We define the *residual capacity* (*i.e.*, the currently available capacity) of processor s_i as $s_i - \sigma_i^f$.

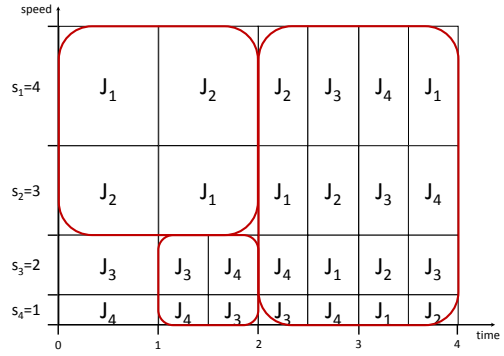
In most prior work on semi-partitioned scheduling on *identical* platforms, a greedy assignment method is used wherein the currently considered task is assigned as fixed if possible. Consider the following example.



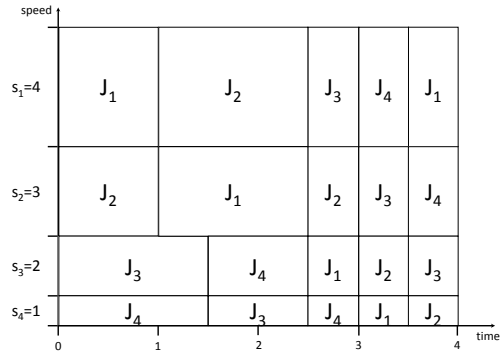
(a) level algorithm resulting schedule



(b) the real schedule for "jointly execute"

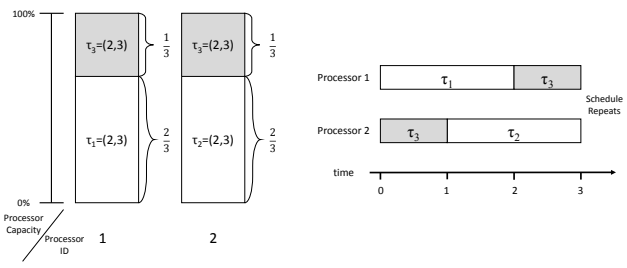


(c) real schedule



(d) final real schedule

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



(a) assignment (b) schedule

Figure 2: Assignment and schedule for Ex. 2.

Ex. 2. Three tasks $\{\tau_1 = (2,3), \tau_2 = (2,3), \tau_3 = (2,3)\}$ are to be scheduled on two unit-speed processors. We greedily assign the first two tasks as fixed, and then require the remaining one to migrate. This results in the share assignment shown in Fig. 2(a). As seen in Fig. 2(b), we can easily determine a schedule corresponding to this assignment such that all deadlines are met.

As the next example shows, a greedy assignment strategy can be problematic on a heterogeneous platform.

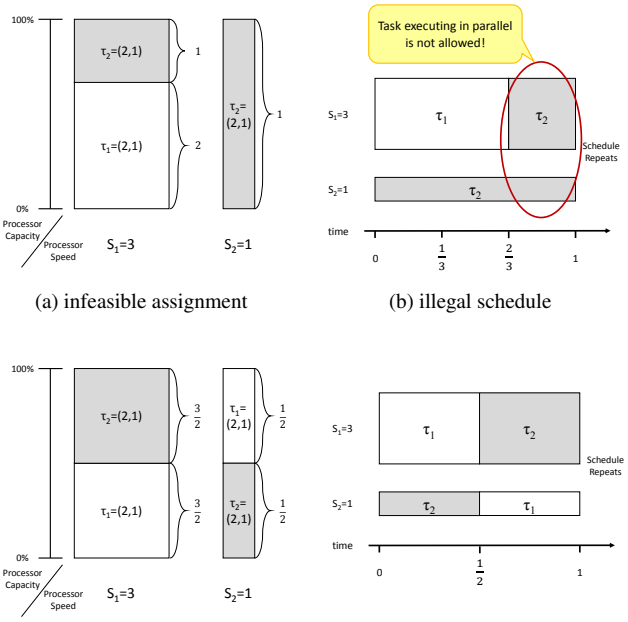
Ex. 3. Consider scheduling the two tasks $\{\tau_1 = (2,1), \tau_2 = (2,1)\}$ on the two-processor uniform heterogeneous platform $\pi = \{s_1 = 3, s_2 = 1\}$. When we first consider assigning τ_1, processor s_1 has enough capacity for it, and if we assign τ_1 there, the residual capacity of the system matches the utilization of τ_2. The task share allocations must be as shown in Fig. 3(a). The allocation to τ_2 implies that it must execute in parallel as shown in Fig. 3(b), so this assignment is infeasible. However, the original system is feasible, as seen in insets (c) and (d) of Fig. 3.

We now determine conditions for ensuring that a task assignment is feasible. After some tasks have been fixed, let $\{z_i\}$ denote the residual capacities of the processors on platform π, indexed in non-increasing order. Note that the indexing of $\{z_i\}$ may differ from that of $\{s_i\}$, *i.e.*, z_i does not necessarily correspond to the residual capacity on s_i. Let $p(i)$ denote the index of the processor with the remaining capacity z_i, *i.e.*, z_i is the remaining capacity of the processor of speed $s_{p(i)}$: $z_i + \sigma^f_{p(i)} = s_{p(i)}$. Also, let $Z_k = \sum_{i=1}^{k} z_i$.

Theorem 2. *Any task set that is feasible on the fully available platform $\pi' = \{s'_1 = z_1, s'_2 = z_2, \ldots, s'_m = z_m\}$ can also be correctly scheduled using the residual capacities $\{z_i\}$ of platform π. (In a correct schedule, all deadlines are met and all requirements of the sporadic model are respected.)*

Proof. We prove this theorem by transforming an arbitrary schedule $\mathcal{I}'$ on π' to a corresponding schedule $\mathcal{I}$ on π such that if $\mathcal{I}'$ is correct, then $\mathcal{I}$ is also correct. Moreover, in $\mathcal{I}$, only a capacity of z_i is utilized on $s_{p(i)}$ for each i.

We split the time line of $\mathcal{I}'$ into slices of width Δ such that, on *any* processor of π', *all* preemptions, migrations, job releases, and job deadlines occur on slice boundaries. This requirement can be met by choosing Δ small enough.



(a) infeasible assignment (b) illegal schedule

(c) feasible assignment (d) correct schedule

Figure 3: Assignment and schedule for Ex. 3. The width of each rectangle represents the speed of its corresponding processor.

We construct $\mathcal{I}$ on a per-slice basis: within each slice, we schedule in $\mathcal{I}$ exactly the same jobs as scheduled in $\mathcal{I}'$ and on exactly the same processors. However, in $\mathcal{I}$, the job (if any) that executes on processor s'_i in $\mathcal{I}'$ is scheduled within the first $\frac{z_i}{s_{p(i)}} \cdot \Delta$ time units of the slice. It is easy to see that the resulting schedule $\mathcal{I}$ is correct if $\mathcal{I}'$ is. In particular, all deadlines will be met and all requirements of the sporadic model are respected (including no intra-task parallelism). Also, it is straightforward to see that only a capacity of z_i is utilized on $s_{p(i)}$ for each i. □

By Theorem 2, after some tasks have been assigned as fixed, the platform defined by the resulting residual capacities can be viewed as a fully available platform as far as the feasibility of the remaining, unassigned tasks is concerned.

In the rest of this section, let τ denote the set of the remaining, unassigned tasks, and let π denote the platform defined by the residual processor capacities. Also, let $\{u_i\}$ denote the utilizations of the remaining tasks, and assume they are indexed in non-increasing order. Let $U_k = \sum_{i=1}^{k} u_i$. Define the total utilization of the remaining tasks as U_τ and the total residual capacity of the platform as Z_π.

Theorem 3. *τ is feasible on π if*[2] *the following conditions hold.*

$$U_\tau \leq Z_\pi \tag{5}$$

$$U_k \leq Z_k \quad \text{for} \quad k = 1, 2, \cdots, m-1 \tag{6}$$

Proof. Follows from Thm. 2 and the feasibility condition for uniform platforms, (2) and (3). □

[2] A counterexample is given in Appendix C that shows that "only if" cannot be asserted here.

By Thm. 3, we have a sufficient test to check if a task assignment is guaranteed to preserve the feasibility of the system. Next, we show that the following scheme can guarantee that any feasible system can have at most m migrating tasks and still be feasible.

- Consider tasks from lightest to heaviest by utilization.
- Use the best-fit bin-packing heuristic to assign as many tasks as possible as fixed.

The guarantee mentioned above follows from the following lemma. (Note that the task τ_n mentioned in the lemma definitely can be assigned as fixed if (5) and (6) hold.)

Lemma 1. *Let n denote the number of tasks in τ and assume $n \geq m+1$. If τ and the residual processor capacities π satisfy* (5) *and* (6)*, then after using the best-fit heuristic to assign the lightest task in τ (i.e., τ_n) as fixed, the remaining task set τ' and residual processor capacities π' must satisfy* (5) *and* (6) *as well.*

Proof. We prove this lemma by contradiction by showing that any violation of (5) or (6) for τ' and π' implies a violation of (5) or (6) for τ and π.

Since we consider tasks from lightest to heaviest, the order of tasks in τ' is the same as that in τ except the absence of the lightest one, τ_n. Thus,

$$u'_i = u_i \quad \text{for } 1 \leq i \leq n-1. \tag{7}$$

Case 1: τ' and π' violate (5). Since the only change is that τ_n is assigned, the total remaining utilization decreases by u_n and total residual capacity decreases by u_n too, *i.e.*, $U'_{\tau'} = U_\tau - u_n$ and $Z'_{\pi'} = Z_\pi - u_n$. Thus, τ' and π' violating (5) implies τ and π violate (5) as well.

Case 2: τ' and π' violate (6). In this case, let z_γ be the processor on which τ_n is fixed. Since $\{z_i\}$ is indexed non-increasingly, without loss of generality, we can assume that the best-fit bin-packing heuristic always chooses the highest-indexed z_i among those with equal values (if it does not, we can re-index them, which will not change either $\{z_i\}$ or the resulting $\{z'_i\}$), *i.e.*, $z_\gamma > z_{\gamma+1}$ if $\gamma < m$. Thus, as a result of the best-fit heuristic, we have

$$u_n > z_i \quad \text{for any } i > \gamma. \tag{8}$$

Moreover, the assignment of τ_n will not alter the indices of the largest $\gamma - 1$ capacities in π, *i.e.*,

$$z'_i = z_i \quad \text{for any } i \leq \gamma - 1. \tag{9}$$

Also, other than z_γ, the relative ordering in $\{z_i\}$ is preserved in $\{z'_i\}$ as well. That is, letting ϕ denote the new index of z_γ in π', *i.e.*, $z'_\phi = z_\gamma - u_n$, where $\gamma \leq \phi \leq m$, $\{z'_i\}$ is

$$\{z_1, \ldots, z_{\gamma-1}, z_{\gamma+1}, \ldots, z_\phi, z_\gamma - u_n, z_{\phi+1}, \ldots, z_m\}. \tag{10}$$

Note that, if $\phi = \gamma$, then the sequence $z_{\gamma+1}, \ldots, z_\phi$ is empty; similarly, $\phi = m$ implies that the sequence $z_{\phi+1}, \ldots, z_m$ is empty.

From (10),

$$z'_i = \begin{cases} z_i & \text{if } 1 \leq i \leq \gamma - 1 \text{ or } i \geq \phi + 1, \\ z_{i+1} & \text{if } \gamma \leq i \leq \phi - 1, \\ z_\gamma - u_n & \text{if } i = \phi. \end{cases} \tag{11}$$

Case 2.1: τ' and π' violate (6) at k such that $k \leq \gamma - 1$. By (7) and (11), $U'_k = U_k$ and $Z'_k = Z_k$. Thus, τ and π violate (6) at k as well.

Case 2.2: τ' and π' violate (6) at k such that $k \geq \gamma$.That is,

$$U'_k > Z'_k. \tag{12}$$

First, we show the following inequality holds in Case 2.2 by considering two sub-cases.

$$Z'_k \geq \left(\sum_{i=1}^{k} z_i\right) - u_n. \tag{13}$$

Case 2.2.1: $\gamma \leq k \leq \phi - 1$.

$$\begin{aligned} Z'_k &= \left(\sum_{i=1}^{\gamma-1} z'_i\right) + \left(\sum_{i=\gamma}^{k-1} z'_i\right) + z'_k \\ &\geq \{\text{since } k \leq \phi - 1 \text{ and } \{z'_i\} \text{ is in non-increasing order}\} \\ &\quad \left(\sum_{i=1}^{\gamma-1} z'_i\right) + \left(\sum_{i=\gamma}^{k-1} z'_i\right) + z'_\phi \\ &= \{\text{by (11)}\} \\ &\quad \left(\sum_{i=1}^{\gamma-1} z_i\right) + \left(\sum_{i=\gamma}^{k-1} z_{i+1}\right) + (z_\gamma - u_n) \\ &= \{\text{simplifying}\} \\ &\quad \left(\sum_{i=1}^{k} z_i\right) - u_n. \end{aligned}$$

Case 2.2.2: $k \geq \phi$.

$$\begin{aligned} Z'_k &= \left(\sum_{i=1}^{\gamma-1} z'_i\right) + \left(\sum_{i=\gamma}^{\phi-1} z'_i\right) + z'_\phi + \left(\sum_{i=\phi+1}^{k} z'_i\right) \\ &= \{\text{by (11)}\} \\ &\quad \left(\sum_{i=1}^{\gamma-1} z_i\right) + \left(\sum_{i=\gamma}^{\phi-1} z_{i+1}\right) + (z_\gamma - u_n) + \left(\sum_{i=\phi+1}^{k} z_i\right) \\ &= \{\text{simplifying}\} \\ &\quad \left(\sum_{i=1}^{k} z_i\right) - u_n. \end{aligned}$$

From these sub-cases, we can conclude that (13) holds.

By (12) and (13), we have

$$U'_k + u_n > \left(\sum_{i=1}^{k} z_i \right). \tag{14}$$

By the condition of Case 2.2, $k \geq \gamma$, and by (8), we have $u_n > z_i$ for any $i \geq k+1 > \gamma$, which implies

$$(m-k)u_n > \left(\sum_{i=k+1}^{m} z_i \right). \tag{15}$$

By (14), (15), and the definition of Z_π,

$$U'_k + (m-k+1)u_n > Z_\pi. \tag{16}$$

Finally, we have

$$\begin{aligned} U_\tau = \; & U_k + \sum_{i=k+1}^{n} u_i \\ = \; & \{\text{by (7)}\} \\ & U'_k + \sum_{i=k+1}^{n} u_i \\ \geq \; & \{\text{since } \{u_i\} \text{ is in non-increasing order}\} \\ & U'_k + (n-k)u_n \\ \geq \; & \{\text{since } n \geq m+1\} \\ & U'_k + (m-k+1)u_n. \end{aligned} \tag{17}$$

By (16) and (17), $U_\tau > Z_\pi$ holds, *i.e.*, τ and π violate (5). □

In the remainder of the paper, we say that an assignment of a task as fixed to a processor is *legal* if and only if (5) and (6) hold for the remaining, unassigned tasks.

Theorem 4. *For any feasible task system, if we continue to assign tasks as fixed as long as legal assignments can be made using the best-fit heuristic, with tasks considered from lightest to heaviest by utilization, then at most m tasks will remain as unassigned.*

Proof. By Thm. 3 and Lem. 1, we can continue to make legal assignments at least until the number of unassigned tasks is m. □

4 Algorithm EDF-tu

We now describe our new scheduling algorithm EDF-tu by considering its assignment and execution phases separately.

Assignment phase. The assignment phase must not only distinguish fixed tasks from migrating ones, but also determine the per-processor share allocations for each migrating task. As for determining which tasks should be fixed, Thm. 4 suggests the way forward: we simply consider tasks from lightest to heaviest by utilization, and keep assigning tasks as fixed via the best-fit heuristic until all of them are assigned or we encounter a task that cannot be so assigned legally. The remaining m' unassigned tasks will be migrating tasks. By Thm. 4, $m' \leq m$. Also, by Thms. 2 and 4, the set of migrating tasks is feasible on the resulting platform as defined by the residual processor capacities. In fact, this set of tasks is feasible on the sub-platform comprised of the m' processors with the largest residual capacities.

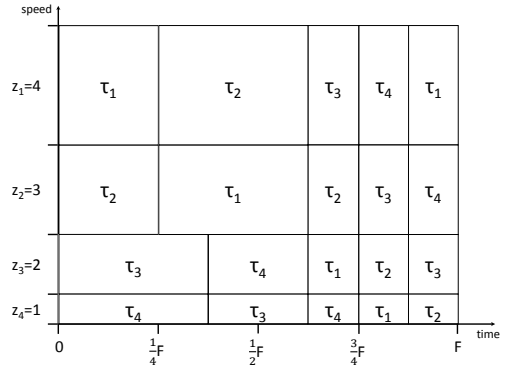
In order to determine per-processor share allocations for migrating tasks, and how such tasks are scheduled alongside fixed ones, we construct a *processor allocation table*. This table indicates which task may execute on which processor within an interval of time, or *frame*, of length F. As shown later in Sec. 5, if HRT-schedulability is the goal, then the frame size F must meet a certain constraint, but this constraint is not required if only SRT-schedulability is required.

We construct the processor allocation table $\mathcal{A}$ via a two-step process (which is illustrated via an example below). In the first step, we construct a processor allocation table $\mathcal{A}'$ for the m' migrating tasks on a hypothetical platform $\pi' = \{s'_1 = z_1, s'_2 = z_2, \ldots, s'_{m'} = z_{m'}\}$ by applying the Level Algorithm to schedule the job set $\mathcal{J}$ with execution costs $\{u_1 \cdot F, u_2 \cdot F, \ldots, u_{m'} \cdot F\}$ on π'. We obtain the table $\mathcal{A}'$ by allocating processor s'_i to task τ_k in each sub-interval where the corresponding job of cost $u_k \cdot F$ executes on processor s'_i. The Level Algorithm ensures that the schedule for $\mathcal{J}$ is free of intra-job parallelism. This implies that task allocations in the table $\mathcal{A}'$ are free of intra-task parallelism. Also, by Thms. 1, 3, and 4, the makespan of the schedule for $\mathcal{J}$ is at most F. This implies that $\mathcal{A}'$ gives task allocations over an interval of length at most F as well. The total allocation recorded for each migrating task τ_k in $\mathcal{A}'$ is $u_k \cdot F$.

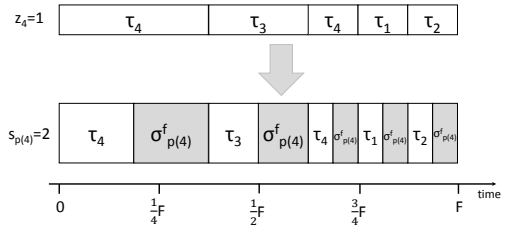
In the second step, we obtain the final table $\mathcal{A}$ by integrating allocations for fixed tasks into $\mathcal{A}'$. Examining the task allocations recorded in $\mathcal{A}'$, we say that the sub-interval $[t_1, t_2)$ is a *maximal non-preemptive sub-interval on processor* s'_i if s'_i is allocated to the same migrating task throughout $[t_1, t_2)$ and s'_i is not allocated to that task either immediately before t_1 or at t_2. We construct the processor allocation table $\mathcal{A}$ for the real physical platform π from $\mathcal{A}'$ by examining all such maximal non-preemptive sub-intervals. In particular, if the migrating task τ_k is allocated in $\mathcal{A}'$ to processor s'_i throughout the maximal non-preemptive sub-interval $[t_1, t_2)$, then we allocate processor $s_{p(i)}$ to τ_k in $\mathcal{A}$ throughout the first $\frac{z_i}{s_{p(i)}}$ of the maximal non-preemptive sub-interval, *i.e.*, $\left[t_1, t_1 + \frac{z_i}{s_{p(i)}} \cdot (t_2 - t_1)\right)$. We allocate the remainder of the maximal non-preemptive sub-interval, *i.e.*, $\left[t_1 + \frac{z_i}{s_{p(i)}} \cdot (t_2 - t_1), t_2\right)$, to fixed tasks on $s_{p(i)}$. We denote this in the table by indicating that the sub-interval $\left[t_1 + \frac{z_i}{s_{p(i)}} \cdot (t_2 - t_1), t_2\right)$ is allocated to $\sigma^f_{p(i)}$. If $m' < m$, then $\mathcal{A}$ is extended to incorporate all processors by fully allocating the processors with residual capacities $z_{m'+1}, \ldots, z_m$ to the fixed tasks assigned to those processors.

The pseudo-code for the assignment process is given in Appendix A. The following example provides an illustration.

Ex. 4. Suppose that after all fixed tasks have been identified, we are left with four migrating tasks with utilizations



(a) example for deriving $\mathcal{A}'$



(b) example on one processor for constructing $\mathcal{A}$ from $\mathcal{A}'$

Figure 4: EDF-tu execution phase illustration for Ex. 4.

$\{3, 3, 2.125, 1.875\}$ to be scheduled on four processors with residual capacities $\{4, 3, 2, 1\}$. Further, suppose we are using a frame size of $F = 4$. Then the schedule resulting from the Level Algorithm within a frame is identical to Ex.1. Fig. 4(a) shows the resulting table $\mathcal{A}'$, which provides allocations only for migrating tasks. To obtain the final table $\mathcal{A}$ for these four processors, we must integrate fixed tasks. To illustrate this, suppose that the processor with residual capacity $z_4 = 1$ corresponds to a physical processor with speed $s_{p(4)} = 2$, *i.e.*, half of this processor's capacity is reserved for fixed tasks. Then the allocations on this processor within each frame will be as depicted in Fig. 4(b).

Execution phase. In the execution phase, the processor allocation table $\mathcal{A}$ is consulted on a frame-by-frame basis at runtime to determine which task may execute on which processor at any given time. In particular, the following rules are applied at any time $t \in [k \cdot F, (k+1) \cdot F)$, where $k \in \mathbb{Z}^+$.

- If processor s'_i is allocated to the migrating task τ_k at time t **mod** F in $\mathcal{A}$, and if τ_k has an unfinished job at time t, then the earliest-released such job is scheduled on processor s'_i.
- If no such job exists, or if processor s'_i is either unallocated or allocated to σ_i^f at time t **mod** F in $\mathcal{A}$, then an unfinished job of a fixed task on s'_i is scheduled on processor s'_i at time t if one exists. If multiple such jobs exist, then the one with the earliest deadline is selected. If no such job exists, then processor s'_i is idled.

According to the sporadic task model, it is possible for a task to release a job within a frame, *i.e.*, at some time $k \cdot F + r$, where $k \in \mathbb{Z}^+$ and $0 < r < F$. Such a job will receive exactly the same allocation over the next F time units as it would receive had it been released at a frame boundary. As a result, EDF-tu guarantees the following two key properties.

Property 1. *Within any time interval of length F, the processor supply guaranteed to a migrating task τ_i is $u_i \cdot F$. Therefore, within any time interval of length L, the processor supply guaranteed to a migrating task τ_i is at least $\left\lfloor \frac{L}{F} \right\rfloor \cdot u_i \cdot F$.*

Property 2. *Within any time interval of length F, the supply guaranteed to the set of fixed tasks on processor s_p is $\sigma_p^f \cdot F$. Therefore, within any time interval of length L, the supply guaranteed to the set of all fixed tasks on processor s_p is at least $\left\lfloor \frac{L}{F} \right\rfloor \cdot \sigma_p^f \cdot F$.*

5 Optimality

We now show that EDF-tu is HRT-optimal, provided the frame size, F, meets a certain requirement. We also show that EDF-tu is SRT-optimal for any choice of F, with the tardiness of any job being at most F. As F decreases, preemption frequencies increase, so the choice of F is a trade-off between temporal guarantees and run-time overheads.

5.1 HRT Optimality

HRT optimality is dealt with in the following theorem.

Theorem 5. *If the frame size F divides the periods of all tasks, then all deadlines will be meet.*

Proof. Since the frame size F divides the periods of all tasks, we can represent task periods as

$$T_i = k_i \cdot F, \quad k_i \in \mathbb{Z}^+. \tag{18}$$

Migrating tasks. By Prop. 1, within any interval of length T_i, a migrating task τ_i is guaranteed supply of at least $\left\lfloor \frac{T_i}{F} \right\rfloor \cdot u_i \cdot F = k_i \cdot u_i \cdot F = T_i \cdot u_i = C_i$. This implies that no job of any migrating task will miss a deadline.

Fixed tasks. The proof of this case utilizes the following claim.

> **Claim 1.** *For real numbers $a, b > 0$ and $x \in \mathbb{Z}^+$, $\left\lfloor \frac{a}{x \cdot b} \right\rfloor \leq \frac{1}{x} \left\lfloor \frac{a}{b} \right\rfloor$.*
>
> *Proof.* Letting $y = \left\lfloor \frac{a}{b} \right\rfloor$ and $c = a - y \cdot b$, we have $a = y \cdot b + c$, where $y \in \mathbb{Z}$ and $0 \leq c < b$. The latter implies $0 \leq \frac{c}{b} < 1$. Hence, because $x, y \in \mathbb{Z}$, $\left\lfloor \frac{y + \frac{c}{b}}{x} \right\rfloor = \left\lfloor \frac{y}{x} \right\rfloor$. Thus, we have $\frac{1}{x} \left\lfloor \frac{a}{b} \right\rfloor = \frac{y}{x} \geq \left\lfloor \frac{y}{x} \right\rfloor = \left\lfloor \frac{y + \frac{c}{b}}{x} \right\rfloor = \left\lfloor \frac{y \cdot b + c}{x \cdot b} \right\rfloor = \left\lfloor \frac{a}{x \cdot b} \right\rfloor$. □

We now dispense with the case of fixed tasks by contradiction. Let t_d be the first time a job of any fixed task on processor s_p misses its deadline, and let t_0 be the latest time instant before t_d that is idle for fixed tasks on processor s_p, *i.e.*, all jobs of fixed tasks on processor s_p released earlier than t_0 have completed by t_0 and such a job is released at t_0. Within the time interval $[t_0, t_d)$, the demand due to the

set of fixed tasks on processor s_p is at most

$$\begin{aligned}
&\sum_{\substack{\tau_i \text{ is a fixed task} \\ \text{on processor } s_p}} \left\lfloor \frac{t_d - t_0}{T_i} \right\rfloor \cdot C_i \\
= \quad & \{\text{by (1)}\} \\
&\sum_{\substack{\tau_i \text{ is a fixed task} \\ \text{on processor } s_p}} \left\lfloor \frac{t_d - t_0}{T_i} \right\rfloor \cdot T_i \cdot u_i \\
= \quad & \{\text{by (18)}\} \\
&\sum_{\substack{\tau_i \text{ is a fixed task} \\ \text{on processor } s_p}} \left\lfloor \frac{t_d - t_0}{k_i \cdot F} \right\rfloor \cdot k_i \cdot F \cdot u_i \\
\leq \quad & \{\text{by Claim 1}\} \\
&\sum_{\substack{\tau_i \text{ is a fixed task} \\ \text{on processor } s_p}} \frac{1}{k_i} \cdot \left\lfloor \frac{t_d - t_0}{F} \right\rfloor \cdot k_i \cdot F \cdot u_i \\
= \quad & \{\text{simplifying}\} \\
&\left\lfloor \frac{t_d - t_0}{F} \right\rfloor \cdot F \cdot \sum_{\substack{\tau_i \text{ is a fixed task} \\ \text{on processor } s_p}} u_i \\
= \quad & \{\text{by (4)}\} \\
&\left\lfloor \frac{t_d - t_0}{F} \right\rfloor \cdot \sigma_p^f \cdot F.
\end{aligned}$$

By Prop. 2, within the time interval $[t_0, t_d)$, the supply guaranteed to the set of fixed tasks on processor s_p is at least

$$\left\lfloor \frac{t_d - t_0}{F} \right\rfloor \cdot \sigma_p^f \cdot F.$$

This implies that a deadline is not missed at time t_d as assumed. □

By Thm. 5, to guarantee HRT optimality, the frame size cannot exceed the greatest common divider (gcd) of all task periods. The gcd could be quite small for some systems (*e.g.*, if at least two periods are relatively prime), yielding high run-time overheads. However, for some systems (*e.g.*, harmonic ones), the frame could be of a reasonable size, yielding acceptable overheads.

5.2 SRT Optimality

SRT optimality is dealt with in the following theorem.

Theorem 6. *Given any frame size $F > 0$, no job will have tardiness exceeding F.*

Proof. As before, we consider migrating and fixed tasks separately.

Migrating tasks. Consider the j^{th} job of the migrating task τ_i, denoted $\tau_{i,j}$. Let t_d be the deadline of $\tau_{i,j}$ and let t_0 be the latest idle instant for task τ_i at or before the release of $\tau_{i,j}$. Also, let t_F be the first time instant at or after t_d such that $t_F - t_0$ is a multiple of F. Then, we have $t_F - t_d < F$.

The number of jobs with deadlines at or before time t_d that τ_i can release at or after time t_0 is at most $\left\lfloor \frac{t_d - t_0}{T_i} \right\rfloor \leq \left\lfloor \frac{t_F - t_0}{T_i} \right\rfloor$. The resulting demand is at most $\left\lfloor \frac{t_F - t_0}{T_i} \right\rfloor \cdot C_i \leq (t_F - t_0) \cdot u_i$. Because $(t_F - t_0)$ is a multiple of F, by Prop. 1, τ_i is guaranteed a supply of $(t_F - t_0) \cdot u_i$ within $[t_0, t_F)$. This implies that $\tau_{i,j}$ completes by time t_F. Thus, no job of a migrating task will have tardiness exceeding F.

Fixed tasks. Let t_d be the deadline of the j^{th} job $\tau_{i,j}$ of the fixed task τ_i and t_0 be the latest idle instant for fixed tasks on processor s_p at or before the release of $\tau_{i,j}$. Also, let t_F be the first time instant at or after t_d such that $t_F - t_0$ is a multiple of F. Then, we have $t_F - t_d < F$.

The number of jobs with deadlines at or before time t_d that a fixed task τ_k on processor s_p can release at or after time t_0 is at most $\left\lfloor \frac{t_d - t_0}{T_k} \right\rfloor \leq \left\lfloor \frac{t_F - t_0}{T_k} \right\rfloor$, so the total demand due to such jobs is at most

$$\begin{aligned}
\sum_{\substack{\tau_k \text{ is a fixed task} \\ \text{on processor } s_p}} \left\lfloor \frac{t_F - t_0}{T_k} \right\rfloor \cdot C_k \; &\leq (t_F - t_0) \cdot \sum_{\substack{\tau_k \text{ is a fixed task} \\ \text{on processor } s_p}} u_k \\
&= (t_F - t_0) \cdot \sigma_p^f.
\end{aligned}$$

Because $(t_F - t_0)$ is a multiple of F, by Prop. 2, the fixed tasks on processor p are guaranteed a supply of $(t_F - t_0) \cdot \sigma_p^f$ within $[t_0, t_F)$. This implies that $\tau_{i,j}$ completes by t_F. Thus, no job of a fixed task will have tardiness exceeding F. □

6 Alternate Assignment Strategies

Given that at most m tasks are migrating under EDF-tu, and these tasks are the heaviest by utilization, two natural questions arise.

- **Q1:** Can we guarantee that fewer than m tasks are migrating?
- **Q2:** Can we require lighter tasks, instead of heavier ones, to migrate?

In this section, we provide counterexamples that show that the answer to each question is no.

Question Q1. We show that the answer to Question Q1 is no by showing that, for any semi-partitioned scheduler, if it is guaranteed that there will be at most k migrating tasks for *any* feasible system, then k cannot be less than m. This result follows from the following counterexample, which consists of m tasks, all of which must migrate.

Ex. 5. Consider a system of m tasks, each with parameters $\tau_i = (1 + \epsilon, 1)$, where $\epsilon < 1/m$, to be scheduled on m uniform processors, where $s_1 = 1 + m \cdot \epsilon$ and $s_i = 1$ for $2 \leq i \leq m$. Conditions (2) and (3) imply that this system is feasible. Now, if we attempt to assign any single task as fixed, then it must be assigned to processor s_1. However, if we do so, in order to receive enough supply, the remaining $m-1$ tasks must fully use the remaining residual capacities, *i.e.*, they must fully utilize $m - 1$ processors (s_2 to s_m) and meanwhile also utilize the residual capacity on s_1. Because intra-task parallelism is forbidden, this is infeasible.

The above counterexample shows that we cannot *generally* guarantee that fewer than m tasks will migrate. However, if we examine a *specific* task system, then it may indeed be possible to require fewer than m tasks to migrate. In fact, in systems that can be fully partitioned, *no* task will migrate. Unfortunately, determining the minimum number of migrating tasks for a specific, concrete task system is NP-hard in the strong sense. This can be shown by transforming from the variable-sized bin-packing problem [12].

Question Q2. The following counterexample shows that the answer to Question Q2 is no as well.

Ex. 6. Consider n tasks to be scheduled on m uniform processors, where $s_1 = 1 + (m+1) \cdot \epsilon$, where $\epsilon < (m-1)/(m+1)$, and $s_i = 1$ for $2 \leq i \leq m$. The n tasks include m heavy ones with parameters $(1+\epsilon, 1)$ and $n-m$ light ones with parameters $(\epsilon, n-m)$. Conditions (2) and (3) imply that this system is feasible. Now, if any one of the m heavy tasks is assigned as fixed, then it must be fixed on processor s_1. However, if we do so, the remaining $m-1$ heavy tasks cannot all be allocated shares that match their utilizations without introducing intra-task parallelism. Thus, the remaining system is infeasible. The formal proof of this is somewhat tedious, so we defer it to Appendix B.

7 Evaluation

The frame size F used in EDF-tu is a tunable parameter. For any feasible task system, tardiness will always be at most F, and if F is set low enough, tardiness will be zero. Given this, it would not be very interesting to experimentally examine issues related to schedulability. However, for a given task system, the Level-Algorithm-induced preemption pattern within a frame is the same regardless of its size, and preemption frequencies over time are higher when F is smaller. Thus, it *is* interesting to experimentally evaluate the number of tasks that are required to migrate and the number of preemptions experienced by such tasks, as it is these tasks that give rise to preemptions induced by the Level Algorithm. In this section, we briefly discuss an experimental evaluation that focuses on these two metrics.

Experimental setup. We assessed the impact of both metrics by randomly generating feasible task systems and by determining for each generated system the number of migrating tasks and the number of preemptions experienced by such tasks per frame. In experimental studies that focus on identical platforms, choosing an overall utilization cap implicitly defines the considered multiprocessor platform. However, in the uniform case, processor speeds must be selected, and the number of such speed settings is unbounded for a given total utilization. To reasonably constrain our experiments, we considered systems of eight processors with a total processor capacity of 36. We considered four such platforms, with speeds as follows: $\pi_1 = \{6, 6, 6, 6, 3, 3, 3, 3\}$, $\pi_2 = \{8, 8, 4, 4, 4, 4, 2, 2\}$, $\pi_3 = \{8, 7, 6, 5, 4, 3, 2, 1\}$, and $\pi_4 = \{15, 3, 3, 3, 3, 3, 3, 3\}$. To randomly generate feasible task systems, we used a framework used in a prior experimental study by us [23], which we do not describe here due to space constraints (the conclusions below are understandable without knowing the precise distributions used to generate task parameters). When using this framework, two categories of task systems are generated: systems that have fewer tasks but heavier tasks (by utilization), and systems that have more tasks but lighter tasks.

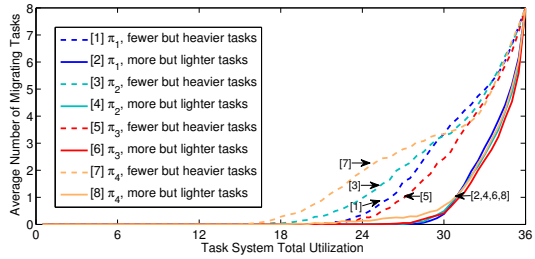


Figure 5: Number of migrating tasks.

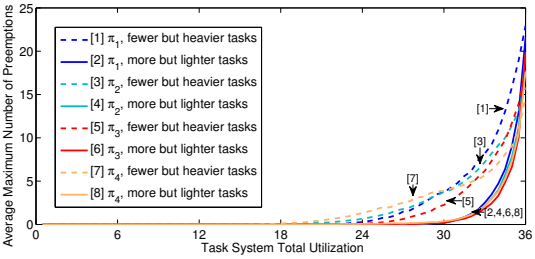


Figure 6: Maximum number of preemptions of migrating tasks per frame.

For each platform and task generating pattern, we varied total utilization within $[0, 36]$ by increments of 0.5, and for each total utilization, we generated 1,000 feasible task systems. Fig. 5 plots the average number of migrating tasks required for each such set of 1,000 task systems. For every generated task system, we also simulated EDF-tu and recorded the maximum number of preemptions per frame of any migrating task. Fig. 6 plots the average of these maximum values for each set of 1,000 task systems. Figs. 7 and 8 show minimum and maximum values in addition to averages for the system setting with fewer but heavier tasks on platform π_1 (we omit such data for the other system settings due to space constraints). Fig. 8 also contains some additional plots, which we discuss later.

Results. As seen in Figs. 5 and 6, the number of migrating tasks is often modest, and these tasks often experience only a moderate number of preemptions per frame. With total utilization as high as 30, the number of migrating tasks (on average) typically is at most four, and the number of preemptions per frame (on average) is at most five. Even in the extreme case that the total utilization achieves the total speed of the platform, the number of preemptions per frame (on average) is still less than 25. While 25 preemptions per frame may seem somewhat high, recall that in a SRT system, we can define the frame size F to be quite large at the cost of increasing the tardiness bound.

Other comments. In work on uniform platforms, the first optimal scheduler for implicit-deadline sporadic task sys-

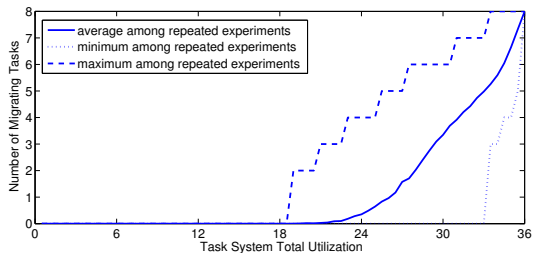


Figure 7: Range of results for the number of migrating tasks for fewer but heavier tasks on π_1.

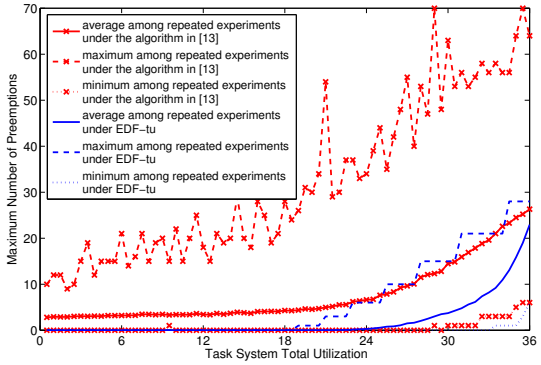


Figure 8: Range of results for the maximum number of preemptions of migrating tasks per frame under the algorithm in [13] or EDF-tu for fewer but heavier tasks on π_1.

tems was proposed by Funk *et al.* [13] as a byproduct towards establishing (2) and (3) as a feasibility condition for such systems. That algorithm also applies the Level Algorithm in a frame-based way, but can require any task to migrate. Moreover, each such task may experience $\mathcal{O}(mn)$ preemptions per frame. In contrast, under EDF-tu, at most m' tasks migrate, where $m' \leq m$, and the number of per-frame preemptions of such a task can be shown to be $\mathcal{O}(m'^2)$. Also, because the processor allocation table is produced offline, the actual number of such preemptions for a specific task system can be estimated precisely, rather than by relying on pessimistically established upper bounds. To provide a sense of the improvement offered by EDF-tu, we have included results for the algorithm in [13] in Fig. 8.

8 Conclusion

We have presented EDF-tu, a semi-partitioned scheduler for uniform heterogeneous multiprocessors that can be configured to be HRT- or SRT-optimal by appropriately sizing a frame parameter. The configurability of EDF-tu enables tradeoffs between timeliness and runtime overheads to be explored. Developing the assignment phase for EDF-tu required confronting a number of issues that do not arise on identical multiprocessors. In addressing these issues, we proposed a sufficient condition for producing a feasible task assignment, and showed via counterexamples that alternative assignment strategies can compromise schedulability. If the frame size F used in EDF-tu is defined to support HRT-optimality, then the resulting preemption overheads could be high for some systems. However, for any choice of F, deadline tardiness is at most F. Moreover, allocations to migrating tasks within a frame are determined offline, so preemption frequencies can be determine precisely, rather than via upper bounds that may be loose.

References

[1] J. Anderson, V. Bud, and U. Devi. An EDF-based scheduling algorithm for multiprocessor soft real-time systems. In *17th ECRTS*, 2005.

[2] J. Anderson, J. Erickson, U. Devi, and B. Casses. Optimal semi-partitioned scheduling in soft real-time systems. In *20th RTCSA*, 2014.

[3] B. Andersson, K. Bletsas, and S. Baruah. Scheduling arbitrary-deadline sporadic task systems on multiprocessors. In *29th RTSS*, 2008.

[4] B. Andersson and E. Tovar. Multiprocessor scheduling with few preemptions. In *12th RTCSA*, 2006.

[5] M. Bhatti, C. Belleudy, and M. Auguin. A semi-partitioned real-time scheduling approach for periodic task systems on multicore platforms. In *27th SAC*, 2012.

[6] big.LITTLE Processing. http://www.arm.com/products/processors/technologies/biglittleprocessing.php.

[7] K. Bletsas and B. Andersson. Notional processors: An approach for multiprocessor scheduling. In *15th RTAS*, 2009.

[8] K. Bletsas and B. Andersson. Preemption-light multiprocessor scheduling of sporadic tasks with high utilisation bound. *Real-Time Systems*, 47(4):319–355, 2011.

[9] A. Burns, R.I. Davis, P. Wang, and F. Zhang. Partitioned EDF scheduling for multiprocessors using a c=d scheme. In *18th RTNS*, 2010.

[10] F. Dorin, P. Yomsi, J. Goossens, and P. Richard. Semi-partitioned hard real-time scheduling with restricted migrations upon identical multiprocessor platforms. *Cornell University Library Archives*, arXiv:1006.2637 [cs.OS], 2010.

[11] M. Fan and G. Quan. Harmonic semi-partitioned scheduling for fixed-priority real-time tasks on multi-core platform. In *DATE*, 2012.

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

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

[14] J. Goossens, P. Richard, M. Lindström, I. Lupu, and F. Ridouard. Job partitioning strategies for multiprocessor scheduling of real-time periodic tasks with restricted migrations. In *20th RTNS*, 2012.

[15] N. Guan, M. Stigge, W. Yi, and G. Yu. Fixed-priority multiprocessor scheduling with Liu and Layland's utilization bound. In *16th RTAS*, 2010.

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

[17] S. Kato and N. Yamasaki. Real-time scheduling with task splitting on multiprocessors. In *13th RTCSA*, 2007.

[18] S. Kato and N. Yamasaki. Portioned EDF-based scheduling on multiprocessors. In *8th EMSOFT*, 2008.

[19] S. Kato and N. Yamasaki. Semi-partitioned fixed-priority scheduling on multiprocessors. In *15th RTAS*, 2009.

[20] H. Leontyev and J. Anderson. Tardiness bounds for EDF scheduling on multi-speed multicore platforms. In *13th RTCSA*, 2007.

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

[22] P. Sousa, P. Souto, E. Tovar, and K. Bletsas. The carousel-EDF scheduling algorithm for multiprocessor systems. In *19th RTCSA*, 2013.

[23] K. Yang and J. Anderson. Soft real-time semi-partitioned scheduling with restricted migrations on uniform heterogeneous multiprocessors. In *22nd RTNS*, 2014.

Appendix A: Assignment Phase Pseudo-Code

initially $\sigma_p^f := 0$ for all p;
index tasks in non-increasing-utilization order;
index processors in non-increasing-speed order;
/* The first $n-m$ lightest tasks are guaranteed to be fixed via the best-fit heuristic. If $n = m$, then this for loop is skipped. */
for $i := n$ **downto** $m+1$ **do**
  Select k that $s_k - \sigma_k^f$ is minimal while at least u_i ;
  $\sigma_k^f := \sigma_k^f + u_i$;
end
/* Try to continue fixing tasks until the fixing step is not legal or all tasks are fixed. */
$m' := m$;
$isLegal := 1$;
repeat
  Select k that $s_k - \sigma_k^f$ is minimal while at least $u_{m'}$;
  $last\_\sigma_k^f := \sigma_k^f$;
  $\sigma_k^f = \sigma_k^f + u_{m'}$;
  for $j := 1$ **to** m' **do**
    if $\sum_{j \text{ largest}}(s_p - \sigma_p^f) < \sum_{j \text{ largest}} u_i$ **then**
      $isLegal := 0$;
    end
  end
  $last\_m' := m'$;
  $m' := m' - 1$;
until $isLegal = 0$ ***or*** $m' = 0$;
if *isLegal* = 0 **then**
  /* If the last fixing step is not legal, restore to last feasible assignment. */
  $m' := last\_m'$;
  $\sigma_k^f := last\_\sigma_k^f$;
  /* Now, we have m' migrating tasks to be scheduled on m' processors using a frame-based schedule. */
  for $j := 1$ **to** m' **do**
    $z_j :=$ the j^{th} largest $(s_p - \sigma_p^f)$;
  end
  Use the Level Algorithm to construct the processor allocation table for a frame;
else
  In this case, there is no migrating task, and a valid partitioned schedule can be generated by applying the uniprocessor EDF scheduler on each processor;
end

Algorithm 1: EDF-tu assignment phase

Appendix B: Detailed Explanation for Ex. 6

Let $\sigma_{i,p}$ denote the capacity allocated to τ_i on processor s_p, where $0 \le \sigma_{i,p} \le s_p$. Then, the *portion* of s_p that is allocated to τ_i is

$$\rho_{i,p} = \frac{\sigma_{i,p}}{s_p}, \tag{19}$$

where $0 \le \rho_{i,p} \le 1$. The portion $\rho_{i,p}$ is the needed percentage of CPU time on s_p for τ_i to receive processor supply on s_p that matches its allocated capacity $\sigma_{i,p}$ on s_p. Thus, if intra-task parallelism is forbidden, then the following condition must hold for the system to be feasible.

$$\sum_{p=1}^{m} \rho_{i,p} \le 1, \quad \text{for any } i \tag{20}$$

For illustration, consider a task that needs to utilize 70% of the CPU time on one processor and 80% of the CPU time on another processor. This is clearly infeasible if intra-task parallelism is not allowed.

Now, let us examine the specific system in Ex. 6. As shown in Ex. 6, if any one of the m heavy tasks is assigned as fixed, then it must be fixed on processor s_1. Without loss of generality, assume that the heavy task τ_1 is fixed on s_1 and the remaining $m-1$ heavy tasks are $\{\tau_2, \tau_3, \ldots, \tau_m\}$. Since τ_1 is fixed on s_1, the other heavy tasks cannot be allocated shares on s_1 exceeding its residual capacity. Thus,

$$\sum_{i=2}^{m} \sigma_{i,1} \le s_1 - u_1. \tag{21}$$

(19) and (21) imply

$$\sum_{i=2}^{m} \rho_{i,1} \le 1 - \frac{u_1}{s_1}. \tag{22}$$

Moreover, by (20), $\sum_{i=2}^{m}\sum_{p=1}^{m} \rho_{i,p} \le m-1$, *i.e.*, $\sum_{i=2}^{m} \rho_{i,1} + \sum_{i=2}^{m}\sum_{p=2}^{m} \rho_{i,p} \le m-1$. Therefore,

$$\sum_{i=2}^{m}\sum_{p=2}^{m} \rho_{i,p} \le (m-1) - \sum_{i=2}^{m} \rho_{i,1}. \tag{23}$$

Thus, the allocated shares for the remaining $m-1$ heavy tasks satisfy

$$\begin{aligned}
& \sum_{i=2}^{m}\sum_{p=1}^{m} \sigma_{i,p} \\
= \; & \{\text{by (19)}\} \\
& \sum_{i=2}^{m}\sum_{p=1}^{m} \rho_{i,p} \cdot s_p \\
= \; & \{\text{rearranging and by } s_p = 1 \text{ for } 2 \le p \le m \text{ as in Ex. 6}\} \\
& \sum_{i=2}^{m} \rho_{i,1} \cdot s_1 + \sum_{i=2}^{m}\sum_{p=2}^{m} \rho_{i,p} \cdot 1
\end{aligned}$$

$$
\begin{aligned}
&\leq \quad \{\text{by (23)}\} \\
&\quad \sum_{i=2}^{m} \rho_{i,1} \cdot s_1 + (m-1) - \sum_{i=2}^{m} \rho_{i,1} \\
&= \quad \{\text{rearranging}\} \\
&\quad (m-1) + \sum_{i=2}^{m} \rho_{i,1} \cdot (s_1 - 1) \\
&\leq \quad \{\text{by (22)}\} \\
&\quad (m-1) + (1 - \frac{u_1}{s_1}) \cdot (s_1 - 1) \\
&= \quad \{\text{by the definitions of } s_1 \text{ and } u_1 \text{ in Ex. 6}\} \\
&\quad (m-1) + \left(1 - \frac{1+\epsilon}{1+(m+1)\cdot\epsilon}\right) \cdot (1+(m+1)\cdot\epsilon - 1) \\
&= \quad \{\text{simplifying}\} \\
&\quad (m-1) + \frac{m \cdot \epsilon}{1 + (m+1) \cdot \epsilon} \cdot (m+1) \cdot \epsilon \\
&= \quad \{\text{simplifying}\} \\
&\quad (m-1) + \frac{m \cdot (m+1) \cdot \epsilon}{1 + (m+1) \cdot \epsilon} \cdot \epsilon \\
&= \quad \{\text{simplifying}\} \\
&\quad (m-1) + \frac{m}{\frac{1}{(m+1)\cdot\epsilon} + 1} \cdot \epsilon \\
&< \quad \{\text{as stated in Ex. 6, } \epsilon < (m-1)/(m+1)\} \\
&\quad (m-1) + \frac{m}{\frac{1}{(m+1)\cdot(m-1)/(m+1)} + 1} \cdot \epsilon \\
&= \quad \{\text{simplifying}\} \\
&\quad (m-1) + \frac{m}{\frac{1}{m-1} + 1} \cdot \epsilon \\
&= \quad \{\text{simplifying}\} \\
&\quad (m-1) + (m-1) \cdot \epsilon,
\end{aligned}
$$

which is the needed total share allocation of the remaining $m-1$ heavy tasks. Thus, the remaining system is not feasible if intra-task parallelism is forbidden.

Appendix C: Further Explanation of Footnote 2

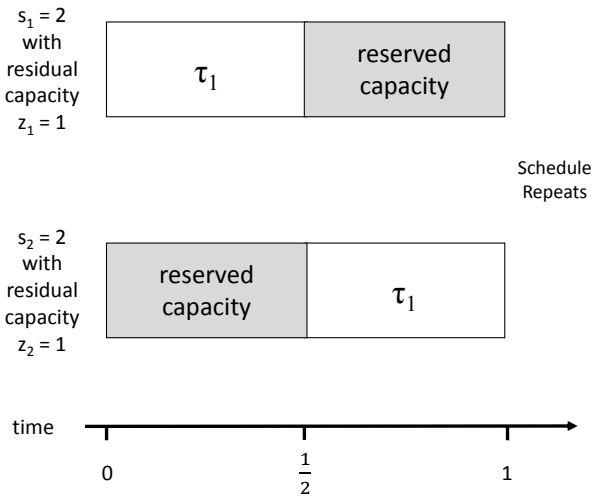


Figure 9: A correct schedule for Ex. 7.

As indicated in Footnote 2, in Thm. 3, (5) and (6) are only a sufficient condition for feasibility. This is because (6) is not necessary for feasibility, although the similar condition (3) is. We show this by the following counterexample.

Ex. 7. Consider scheduling a single task $\tau_1 = (2, 1)$ on two unit-speed processors. This system is clearly infeasible since its utilization $u_1 = 2 > 1$ holds, which violates (3).

Now, consider scheduling the same task τ_1 on two other processors, where both processors have a residual capacity of 1, *i.e.*, $z_1 = z_2 = 1$. Then, this system violates (6), but it could be feasible. For example, assuming the two processors both have an initial speed of 2, Fig. 9 is a correct schedule for this system.