Optimal Semi-Partitioned Scheduling in Soft Real-Time Systems *

James H. Anderson[1], Benjamin N. Casses[1], UmaMaheswari C. Devi[2], and Jeremy P. Erickson[1]

[1]Dept. of Computer Science, University of North Carolina at Chapel Hill
[2]IBM Research - India

Abstract

Semi-partitioned real-time scheduling algorithms extend partitioned ones by allowing a (usually small) subset of tasks to migrate. The first such algorithm to be proposed was directed at soft real-time (SRT) sporadic task systems where bounded deadline tardiness is acceptable. That algorithm, called EDF-fm, is able to fully utilize the underlying hardware platform's available capacity. Moreover, it has the desirable practical property that migrations are boundary-limited, *i.e., they can only occur at job boundaries. Unfortunately, EDF-fm requires restrictions on per-task utilizations, and thus is not optimal. In this paper, a new boundary-limited, semi-partitioned algorithm is presented for SRT systems that is the first such algorithm to be optimal. This algorithm, called EDF-os, is similar to EDF-fm but utilizes several new mechanisms that obviate the need for per-task utilization restrictions. Experiments presented herein show that, not only is EDF-os provably better than EDF-fm with respect to schedulability, tardiness bounds under EDF-os are often much lower as well.*

1 Introduction

Multiprocessor real-time scheduling algorithms may follow a *partitioned* or *global* approach or some hybrid of the two. Under partitioned scheduling, tasks are statically assigned to processors, while under global scheduling, they are scheduled from a single run queue and hence may migrate. When comparing different scheduling approaches, one criterion is *optimality*, i.e., the ability to correctly schedule (without timing constraint violations) any task system for which a correct schedule exists. In the case of implicit-deadline (see Sec. 2) sporadic task systems, optimality can be achieved via global scheduling, but not partitioning; however, global scheduling entails higher runtime overheads. When designing a hybrid approach, the goal is usually to attain optimal or near-optimal behavior but with less overhead than a truly global approach.

One such hybrid approach is *semi-partitioned* scheduling, which extends partitioned scheduling by allowing those tasks that cannot be feasibly assigned to processors to migrate. Semi-partitioned scheduling was first proposed for

supporting soft real-time (SRT) sporadic task systems for which bounded deadline tardiness is allowed [1]. Subsequently, several semi-partitioned algorithms were proposed for hard real-time (HRT) systems [2, 3, 6, 7, 8, 11, 13, 14, 15, 16, 17, 18, 19, 20, 25].

The original SRT algorithm proposed in [1], called EDF-fm, is able to fully utilize the underlying hardware platform's available capacity. Moreover, it is *boundary-limited*: a migrating task may only migrate between job boundaries (i.e., between successive invocations). Unfortunately, EDF-fm requires restrictions on per-task utilizations, and thus is not optimal. In their simplest form, these restrictions preclude any task utilization from exceeding 0.5, though they can be relaxed somewhat, as discussed below.

Of the HRT algorithms cited above, two are optimal, at least in theory, namely RUN [25] and EKG [3]. However, both are optimal only for implicit deadline periodic task systems, and EKG becomes optimal (for periodic systems) only when a configurable parameter k becomes arbitrarily close to the number of processors, which unrealistically increases preemption frequency. Additionally unlike EDF-fm, neither of these algorithms is boundary-limited. Thus, they allow jobs to migrate, which can be expensive in practice if jobs maintain much cached state.

Contributions. In this paper, we present the first boundary-limited, semi-partitioned scheduling algorithm that is optimal for SRT sporadic task systems. This algorithm, which we call EDF-os (earliest-deadline-first-based optimal semi-partitioned scheduling), is based on EDF-fm.

EDF-fm was designed with implicit-deadline sporadic task systems in mind. It functions in two phases: an offline *assignment phase*, where tasks are assigned to processors and *fixed* tasks (which do not migrate) are distinguished from *migrating* ones (which do); and an online *execution phase*, where invoked tasks are scheduled via rules that extend earliest-deadline-first (EDF) scheduling to account for fixed and migrating tasks. EDF-fm's per-task utilization restrictions (the source of its non-optimality) arise for two reasons. First, the proof of bounded tardiness for EDF-fm relies crucially on the fact that migrating tasks never miss deadlines. Such misses are problematic because they create complex "couplings" of processors that are difficult to analyze: a miss by a migrating task on one processor can delay the processing of work due to it on another processor. Second, in EDF-fm, two migrating tasks may be assigned to the

same processor. EDF-fm schedules the jobs of such tasks on an EDF basis and prioritizes them over jobs of fixed tasks. To avoid misses by migrating tasks as desired, migrating tasks are simply required to have a utilization of at most 0.5 (or more generally, those sharing a processor must have a combined utilization of at most 1.0). The fact that migrating tasks do not miss deadlines then follows trivially from the optimality of EDF on one processor [23].

In EDF-os, we eliminate such utilization restrictions by modifying both phases of EDF-fm. In particular, we modify the assignment phase by considering tasks in a certain order and by allowing a migrating task to execute on any number of processors (in EDF-fm, such a task executes on at most two processors). We modify the scheduling phase by statically prioritizing the jobs of certain migrating tasks over those of others (in EDF-fm, such jobs are always scheduled on an EDF basis). As a result of our modifications, migrating tasks can miss deadlines. However, we show that the complex "coupling" noted above can be managed in our analysis by leveraging properties that follow from our modified assignment phase and the more predictable nature of the static prioritizations we introduce. This analysis shows that EDF-os is optimal: deadline tardiness is bounded provided total utilization is at most the system's capacity and per-task utilizations are at most 1.0. Moreover, this claim of optimality holds regardless of whether deadlines are implicit, constrained, or unrestricted (see Sec. 2).

Organization. We present our optimality proof (Sec. 4) after first providing needed background (Sec. 2) and describing EDF-os in detail (Sec. 3). We then discuss extensions to our main results (Sec. 5), present an experimental comparison of EDF-os and EDF-fm (Sec. 6), and conclude (Sec. 7). While EDF-os is provably superior to EDF-fm with respect to schedulability, our experiments show that tardiness bounds under EDF-os are often much lower as well.

2 Background

We consider the scheduling of a sporadic task system $\tau = \{\tau_1, \tau_2, \ldots, \tau_N\}$ on M identical processors—we assume familiarity with the sporadic task model [24]. Task τ_i is specified by (C_i, T_i), where C_i is its maximum per-job execution requirement and T_i is its period. The jth job of τ_i, denoted $\tau_{i,j}$, has release time $r_{i,j}$ and deadline $d_{i,j}$. We initially restrict attention to *implicit* deadlines ($d_{i,j} = r_{i,j} + T_i$) but later (in Sec. 5) consider both *constrained* deadlines ($d_{i,j} \leq r_{i,j} + T_i$) and *unrestricted* deadlines (no relationship between $d_{i,j}$ and $r_{i,j} + T_i$ assumed). We denote the *utilization* of τ_i by $U_i = C_i/T_i$, and the pth processor as P_p. We assume that time is discrete.

In the scheduling algorithms we consider, each task is allocated a non-zero fraction, or *share*, of the available utilization of 1.0 on certain processors. Task τ_i's share (potentially zero) on P_p is denoted $s_{i,p}$. The total share allocation on P_p is denoted $\sigma_p \triangleq \sum_{\tau_i \in \tau} s_{i,p}$. We require that $\sigma_p \leq 1.0$ and that each task's total share allocation matches its utilization: $U_i = \sum_{k=1}^{M} s_{i,k}$. If τ_i has non-zero shares on multiple (only one) processor, then it is a *migrating* (*fixed*) task.

initially $s_{i,p} = 0$ and $\sigma_p = 0$ for all i and p
$p := 1;$
for $i := 1$ **to** N **do**
    if $U_i \leq 1 - \sigma_p$ **then** /* τ_i is fixed */
        $s_{i,p} := U_i;$
        $\sigma_p := \sigma_p + U_i$
    else /* τ_i is migrating */
        $s_{i,p}, \ s_{i,p+1} := (1 - \sigma_p), \ U_i - (1 - \sigma_p);$
        $\sigma_p, \ \sigma_{p+1} := 1, \ s_{i,p+1}$
    fi;
    if $\sigma_p = 1$ **then** $p := p + 1$ **fi**
od

Figure 1: EDF-fm assignment phase.

In the scheduling algorithms we consider, each job of each task τ_i executes on a specific processor. The *fraction* of τ_i's jobs (potentially zero) that execute on processor P_p is denoted $f_{i,p}$. Such fractions are commensurate with τ_i's share allocations:

$$f_{i,p} = \frac{s_{i,p}}{U_i}. \tag{1}$$

The lowest-indexed processor to which migrating task τ_i assigns jobs is called its *first processor*.

If a job $\tau_{i,j}$ completes at time t, then its *tardiness* is $max(0, t - d_{i,j})$. We seek scheduling algorithms that ensure bounded tardiness: for each task, there is a constant upper bound on the tardiness of any of its jobs. We consider only *feasible* task systems that satisfy the following conditions.

$$\forall \tau_i \in \tau, U_i \leq 1, \tag{2}$$

$$\sum_{\tau_i \in \tau} U_i \leq M. \tag{3}$$

EDF-fm. The EDF-os algorithm presented herein extends the EDF-fm algorithm proposed by Anderson et al. [1]. EDF-fm consists of *assignment* and *execution* phases. During the assignment phase, tasks are allocated shares offline via the procedure in Fig. 1. This procedure allocates processor utilization (as shares) to tasks by considering each processor and task in turn. If the currently considered processor P_p has sufficient unallocated utilization, then the currently considered task τ_i is assigned to it as a fixed task; otherwise, τ_i exhausts the remaining unallocated utilization of P_p and receives the rest of its needed allocation from P_{p+1}.

In the execution phase, released jobs are scheduled online without migration (i.e., each job executes on only one processor). The following prioritizations are used on each processor: migrating tasks are prioritized over fixed ones, and jobs of a given type (fixed or migrating) are prioritized against each other on an EDF basis. By the assignment procedure in Fig. 1, at most two migrating tasks can have non-zero shares on a given processor. It is required that for any two such tasks, their combined utilization is at most 1.0. This ensures that such tasks do not miss deadlines (which is a crucial property in the tardiness analysis of EDF-fm).

To ensure that fixed tasks have bounded tardiness, it is important that no processor be overloaded in the long run.
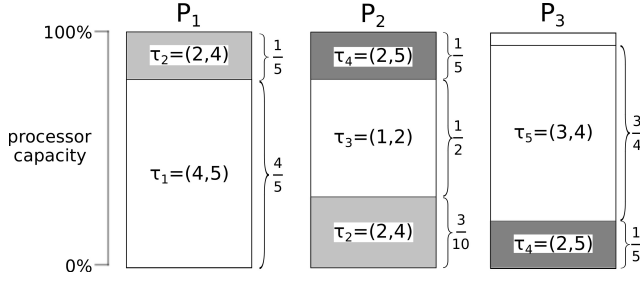
Figure 2: EDF-fm task assignment for Ex. 1. Shares of migrating tasks are shown in a darker shade. $\tau_1 = (4,5)$ is a fixed task on P_1, $\tau_2 = (2,4)$ migrates between P_1 and P_2 with with $s_{2,1} = \frac{1}{5}$ and $s_{2,2} = \frac{3}{10}$, and so on. After assignment, $\frac{1}{20}$ of P_3 is unused.

This can be ensured by employing a mechanism that ensures that, in the long run, each migrating task τ_i submits an appropriate fraction of its jobs to each of the two processors on which it executes. Such fractions are given by (1). In EDF-fm, the exact allocation of such jobs to processors is done by leveraging results from work on Pfair scheduling. We illustrate this with an example below.

Ex. 1. Consider the task system $\tau = \{(4,5), (2,4), (1,2), (2,5), (3,4)\}$, with $\sum_{\tau_i \in \tau} U_i = \frac{59}{20} \le 3$. Hence, τ is feasible on 3 processors. The assignment phase of EDF-fm will produce the share allocations in Fig. 2 when applied to the tasks in τ in the listed order.[1] Note that τ_2 has a share of $\frac{1}{5}$ on P_1 and $\frac{3}{10}$ on P_2. Thus, by (1), $f_{1,1} = (\frac{1}{5})/(\frac{1}{2}) = \frac{2}{5}$ of its jobs should execute on P_1 in the long run, and $f_{1,2} = (\frac{3}{10})/(\frac{1}{2}) = \frac{3}{5}$ of is jobs should execute on P_2. At runtime, EDF-fm determines which processor to allocate to a newly released job of such a migrating task by applying a formula that is derived by considering a Pfair schedule [4] of certain fictitious periodic tasks. To avoid confusion, we will call these fictitious tasks "processes" (instead of tasks). For the case of τ_2 in Ex. 1, two Pfair processes T and U are considered with utilizations $\frac{2}{5}$ and $\frac{3}{5}$, respectively. These utilizations match the fractions $\frac{2}{5}$ and $\frac{3}{5}$ as computed using (1) above. Job-to-processor allocations are made for τ_2 by *conceptually* maintaining a single-processor Pfair schedule of T and U assuming each is always available for execution. Whenever a new job $\tau_{2,j}$ of τ_2 is released, the Pfair schedule is extended by one quantum. If process T is scheduled during that quantum, then $\tau_{2,j}$ is scheduled on P_1 (τ_2's first processor); if process U is instead scheduled, then $\tau_{2,j}$ is scheduled on P_2. This is illustrated in Fig. 4. By the definition of a Pfair schedule, by any integral point in time t in the Pfair schedule under consideration (where each integral time unit is one quantum), process T will have received approximately $\frac{2}{5} \cdot t$ quanta, and U will have received approximately $\frac{3}{5} \cdot t$ quanta. Thus, by any point in time in the EDF-fm schedule, approximately $\frac{2}{5}$ of the released jobs of τ_2 will have been assigned to P_1 and approximately $\frac{3}{5}$ to P_2, as desired.

More specifically, in a Pfair schedule of a periodic sys-

---

[1] Strategies for selecting an assignment order are considered in [1].
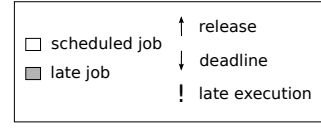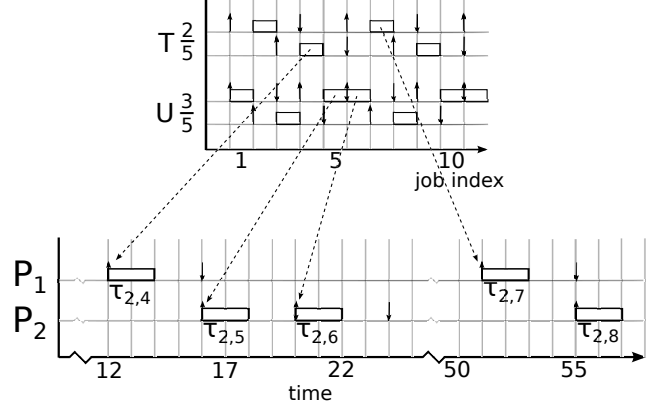

Figure 3: Key for Figures 4– 6, 9 and 11.

Figure 4: Using a fictitious Pfair schedule to assign jobs of τ_2 from Ex. 1. The upper part of the figure shows a Pfair schedule for two processes $T = \frac{2}{5}$ and $U = \frac{3}{5}$. (In such a schedule, each quantum of a task's execution has a release time and deadline.) The lower part shows how released jobs of τ_2 are distributed between P_1 and P_2 based on the Pfair schedule, starting with the fourth job of τ_2 (for clarity, no jobs of other tasks are depicted in the lower part). Since job releases are sporadic, consecutive jobs of τ_2 may be separated by more than T_2, as seen for jobs $\tau_{2,6}$ and $\tau_{2,7}$.

tem of processes that do not over-utilize the assumed processor platform, by the integral time instant t, a process with utilization u will have received between $\lfloor u \cdot t \rfloor$ and $\lceil u \cdot t \rceil$ quanta. In our case, a single-processor platform is assumed and the Pfair processes under consideration will always have a total utilization of 1.0. Based on this property of Pfair schedules, the following is shown in [1].

Property 1. *In any EDF-fm schedule, out of the first n jobs of a migrating task τ_i, the number of jobs assigned to some processor P_p is between $\lfloor f_{i,p} \cdot n \rfloor$ and $\lceil f_{i,p} \cdot n \rceil$*

EDF-fm does not really maintain a fictitious Pfair schedule per migrating task to assign its jobs; rather, a formula derived from Pfair scheduling principles is used that maps the index of a newly-released job to a processor. For example, for task τ_2 in Ex. 1, its fourth job would be mapped to P_1, its fifth job to P_2, and so on (see Fig. 4). Fig. 5 shows the EDF-fm schedule for the task system in Ex. 1 up to time 36, assuming each job is released as soon as possible.

Ex. 2. We now give an example task system that shows that if the task utilization restriction of EDF-fm is violated, then migrating tasks may miss deadlines. Such misses invalidate the tardiness analysis given in [1]. Consider the system $\tau = \{(4,6), (2,3), (5,6), (2,3), (1,2), (2,3)\}$. Be-
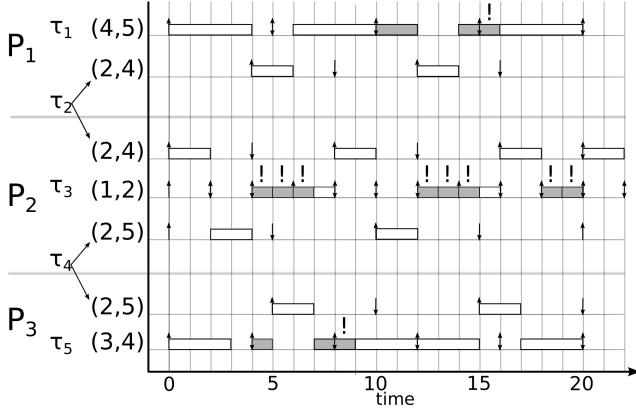
Figure 5: An EDF-fm schedule for the task system in Ex. 1. $f_{2,1} = \frac{2}{5}$ and $f_{2,2} = \frac{3}{5}$. $f_{4,2} = f_{4,3} = \frac{1}{2}$.
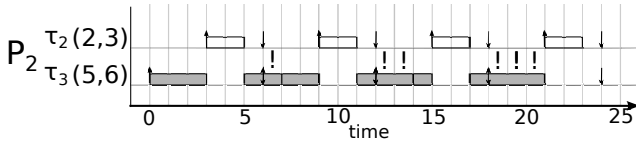

Figure 6: An EDF-fm schedule for the task system in Ex. 2 showing execution on P_2. Jobs from $\tau_3 = (5,6)$ complete late.

cause $\sum_{\tau_i \in \tau} U_i = 4 \leq 3$, τ is feasible on 4 processors. Because all task utilizations exceed $\frac{1}{2}$, EDF-fm's utilization restriction will be violated regardless of the order in which tasks are considered for assignment. For the listed order, we show that deadlines may be missed by migrating tasks on P_2. Because $\tau_1 = (4,6)$ will be assigned as a fixed task on P_1, τ_2 will migrate between P_1 and P_2 with $s_{2,1} = \frac{1}{3}$ and $s_{2,2} = \frac{1}{3}$. Since only $\frac{2}{3}$ of the available utilization of P_2 remains after assigning τ_2, $\tau_3 = (5,6)$ will migrate between P_2 and P_3 and its share on P_2 will be $s_{3,2} = \frac{2}{3}$. Note that $f_{2,1} = \frac{1}{3} \cdot \frac{3}{2} = \frac{1}{2}$. The Pfair-based mapping formula will assign odd-indexed jobs of τ_2 to P_1 and even-indexed jobs of τ_2 to P_2. For τ_3, $f_{3,2} = \frac{2}{3} \cdot \frac{6}{5} = \frac{4}{5}$. The mapping formula will assign the first four jobs in each consecutive group of five jobs of τ_3 to P_2. Fig. 6 shows the first 25 time units of execution on P_2 assuming deadline ties are broken in favor of τ_2. Note that each of the first four jobs of τ_3 misses its deadline.

3 EDF-os

EDF-os consists of assignment and execution phases, like EDF-fm. Its assignment phase is described by the procedure in Fig. 7. An assignment is produced in two steps: first, as many tasks as possible are assigned as fixed tasks, using a worst-fit decreasing bin-packing heuristic. Then, all remaining tasks are assigned (in decreasing utilization order) as migrating tasks by considering each processor and remaining task in turn. Each such task is allocated non-zero shares from a succession of processors until the sum of its shares equals its utilization. Like the assignment procedure

for EDF-fm, this procedure ensures that there are at most two migrating tasks with non-zero shares on any processor. However, a migrating task can now have non-zero shares on more than two processors. Note that we are no longer imposing any restrictions on task utilizations (other than that they be at most 1.0), so it is now possible that migrating tasks may be tardy. Note also that, because tasks are considered in decreasing utilization order, the utilization of any migrating task is no more than that of any fixed task.

The same assignment scheme has been used by Sarkar et al. [26] in work on frame-based fair scheduling for rate-based task systems. Specifically, this scheme is used in their work to determine the number of units of execution to allocate for each task in a frame (specified interval of time).

In the execution phase, EDF-os works as follows. As in EDF-fm, each job executes on only one processor. On any processor, migrating tasks are statically prioritized over fixed ones, and fixed tasks are prioritized against each other using EDF (like in EDF-fm). Also, if a processor has two migrating tasks τ_i and τ_{i+1}, then τ_i is statically prioritized over τ_{i+1} (this differs from EDF-fm). That is, a migrating task executes with highest priority on any processor that is not its first processor (recall Sec. 2). Informally, this rule ensures that tardiness is "created" for a migrating task only on its first processor; on its other processors, one of its jobs will be tardy only if its predecessor job was also tardy. In fact, any such job assigned to a non-first processor will be scheduled as soon as it is eligible (i.e., as soon as it has been released and its predecessor has finished). As we shall see in the tardiness bound proof in Sec. 4, this very predictable execution behavior for "non-first-processor" jobs can be leveraged to derive a tardiness bound for all migrating tasks, and in turn all fixed tasks.

Because a migrating task may execute on more than two processors in EDF-os, we must use a slightly altered version of EDF-fm's Pfair-based job assignment policy. In particular, if a migrating task executes on n processors, then we conceptually manage n Pfair processes with utilizations that sum to 1.0, where each Pfair process corresponds to a processor, as before. If, in a schedule of these n processes on a single processor, the kth process is allocated time slot t, then the tth job of the migrating task is assigned to processor P_k. With this generalized assignment policy, Prop. 1 continues to hold.

Ex. 1 (revisited). We now discuss how EDF-os would schedule the task system from Ex. 1 in Sec. 2. For convenience, we list here the tasks in decreasing utilization order: $\tau = \{(4,5),(3,4),(2,4),(1,2),(2,5)\}$. Fig. 8 shows how EDF-os assigns these tasks to processors. Note that now there is only one migrating task, τ_5, which executes on P_1 and P_2. Because its shares are the same on these two processors, we have $f_{5,1} = \frac{1}{2}$ and $f_{5,2} = \frac{1}{2}$. Our Pfair-based job assignment policy will assign odd-numbered jobs of τ_5 to P_1 and the even numbered ones to P_2. Fig. 9 shows an example schedule of this system. In this schedule, only fixed tasks miss deadlines. Note also that jobs of τ_5 alternate between P_1 and P_2.

initially $s_{i,p} = 0$ and $\sigma_p = 0$ for all i and p
$/*$ assign fixed tasks via a worst-fit decreasing packing $*/$
Index tasks in the order of heaviest utilization to lightest;
for $i := 1$ **to** N **do**
    Select p such that σ_p is minimal;
    if $U_i > 1 - \sigma_p$ **then**
        break $/*$ can't assign any more fixed tasks $*/$
    fi;
    $s_{i,p},\ \sigma_p,\ last := U_i,\ \sigma_p + U_i,\ i$
    fi
od;
$/*$ now assign migrating tasks $*/$
$p := 1;$
for $i := last + 1$ **to** N **do**
    $remaining := U_i$
    repeat
        $s_{i,p} := min(remaining,\ (1 - \sigma_p));$
        $\sigma_p,\ remaining := \sigma_p + s_{i,p},\ remaining - s_{i,p};$
        if $\sigma_p = 1$ **then** $p := p + 1$ **fi**
    until $remaining = 0$
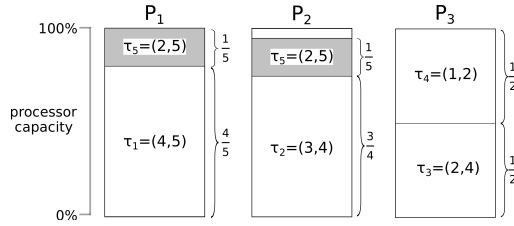**od**

Figure 7: EDF-os assignment phase.

Figure 8: EDF-os task assignment for Ex. 1: $\tau_1 = (4, 5)$, $\tau_2 = (3, 4)$, $\tau_3 = (2, 4)$, and $\tau_4 = (1, 2)$ are assigned as fixed tasks. $\tau_5 = (2, 5)$ (shaded) is the only migrating task. For it, $s_{5,1} = \frac{1}{5}$ and $s_{5,2} = \frac{1}{5}$. After assignment, $\frac{1}{20}$ of P_2 is unused.

Ex. 2 (revisited). Next, we consider how EDF-os would schedule the task system from Ex. 2 in Sec. 2. As before, we list tasks in decreasing utilization order for convenience: $\tau = \{(5, 6), (4, 6), (2, 3), (2, 3), (2, 3), (1, 2)\}$. The task assignment that EDF-os produces is shown in Fig. 10. Note that this time there are two migrating tasks, and one of them, $\tau_5 = (2, 3)$, executes on three processors, P_1, P_2, and P_3. Fig. 11 shows an example EDF-os schedule for this task system.

4 Tardiness Bounds

In this section, we derive tardiness bounds for tasks scheduled by EDF-os. We consider migrating and fixed tasks separately, in Secs. 4.1 and 4.2, respectively. In the rest of this section, we assume that the task system τ being analyzed is feasible (refer to (2) and (3)). We denote the set of all fixed tasks on processor P_p as τ_p^f, and the sum of the shares of all fixed tasks on P_p as σ_p^f.

We begin by establishing several properties that follow from the assignment procedure in Fig. 7. Recall that, as discussed in Sec. 3, Prop. 1 continues to hold for EDF-os.
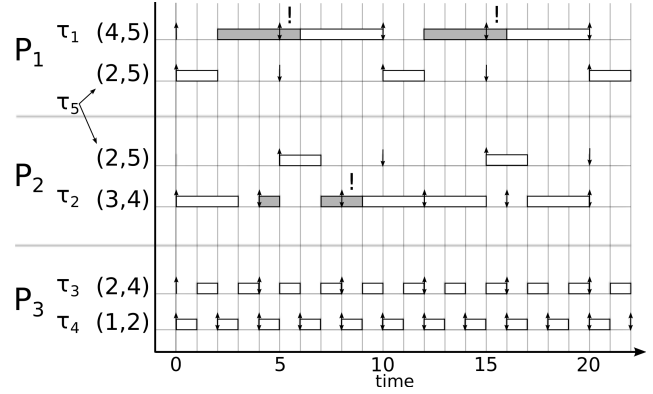
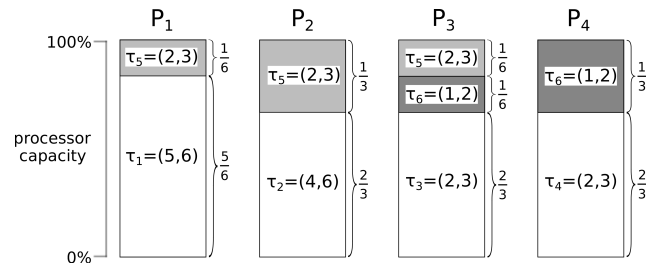Figure 9: EDF-os schedule for Ex. 1. $f_{5,1} = f_{5,2} = \frac{1}{2}$.

Figure 10: EDF-os task assignment for Ex. 2.

Property 2. *For each migrating task τ_i, $U_i < 1$.*

This property follows from the worst-fit decreasing heuristic used by our assignment procedure. Because τ is feasible, if $U_i < 1$ fails to hold, then $U_i = 1$ holds. Moreover, $i \le M$, for otherwise, total utilization would exceed M. These facts imply that τ_i would have been assigned as a fixed task to a dedicated processor.

Property 3. *There are no more than two migrating tasks that assign jobs to processor P_p. If there are two migrating tasks that assign jobs to P_p, then P_p is the first processor for exactly one of them.*

It can be shown by induction that when our assignment procedure first considers a migrating task τ_i, there can be at most one migrating task already assigned to the currently considered processor (which will be τ_i's first processor). From this, Prop. 3 follows.

Property 4. *For processor P_p with one or more migrating tasks τ_i (and possibly τ_k) that have shares $s_{i,p}$ (and $s_{k,p}$), $\sigma_p^f + s_{i,p} + s_{k,p} \le 1$.*

Our assignment procedure does not allow σ_p to exceed 1.0 (i.e., P_p cannot be over-allocated).

Property 5. *If processor P_p contains migrating tasks τ_i and τ_k and P_p is the first processor of τ_k, then $s_{i,p} + U_k < 1$.*

Because tasks are assigned in order of decreasing utilization, there must be some fixed task τ_f on P_p such that
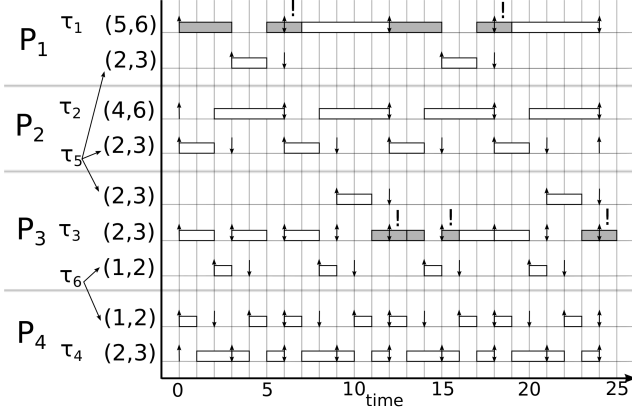
Figure 11: EDF-os schedule for Ex. 2. $f_{5,1} = \frac{1}{4}$, $f_{5,2} = \frac{1}{2}$ and $f_{5,3} = \frac{1}{4}$. $f_{6,3} = \frac{1}{3}$ and $f_{6,4} = \frac{2}{3}$.

$U_f \geq U_k$. Therefore, by Prop. 4 and because $s_{k,p} > 0$, Prop. 5 holds.

Property 6. *Out of any c consecutive jobs of some migrating task τ_i, the number of jobs released on P_p is at most $f_{i,p}(c-1) + 2$.*

By Prop. 1, if τ_i executes jobs on P_p, then out of its first n jobs, the number assigned to P_p is between $\lfloor f_{i,p} \cdot n \rfloor$ and $\lceil f_{i,p} \cdot n \rceil$. Thus, out of any c consecutive jobs of τ_i, where the index of the first such job is j, the number of jobs assigned to P_p is at most

$$\lceil f_{i,p} \cdot (j + c - 1) \rceil - \lfloor f_{i,p} \cdot (j) \rfloor$$
$$\leq \{\text{Since } \lceil x + y \rceil \leq \lceil x \rceil + \lceil y \rceil\}$$
$$\lceil f_{i,p} \cdot (j) \rceil + \lceil f_{i,p} \cdot (c - 1) \rceil - \lfloor f_{i,p} \cdot (j) \rfloor$$
$$\leq \{\text{Since } \lceil x \rceil - \lfloor x \rfloor \leq 1\}$$
$$\lceil f_{i,p} \cdot (c - 1) \rceil + 1$$
$$< \{\text{Since } \lceil x \rceil < x + 1\}$$
$$f_{i,p} \cdot (c - 1) + 2.$$

4.1 Tardiness Bounds for Migrating Tasks

In this subsection, we derive a tardiness bound for migrating tasks. Because such tasks are statically prioritized over fixed ones, we need not consider fixed tasks in this derivation. Thus, all referenced tasks in this subsection are assumed to be migrating.

First, we provide a bound on the work from a migrating task that competes with an arbitrary task. This result will be used both here and in the next subsection.

Lemma 1. *Consider a migrating task τ_i that releases jobs on processor P_p. Let $t_0 \geq 0$ and $t_c > t_0$. If no job of τ_i has tardiness exceeding Δ_i, then the demand from τ_i in the interval $[t_0, t_c)$ on P_p is less than*

$$(s_{i,p})(t_c - t_0) + (s_{i,p})(\Delta_i + T_i) + 2C_i.$$

Proof. Since we assume that the maximum tardiness of τ_i is at most Δ_i, we know that any job released by τ_i will take no more than $T_i + \Delta_i$ time units to complete, so jobs of τ_i released before $t_0 - (\Delta_i + T_i)$ cannot create demand in $[t_0, t_c)$. Thus, competing demand for execution from jobs of τ_i in the interval $[t_0, t_c)$ comes from jobs of τ_i released in $[t_0 - \Delta_i - T_i, t_c)$. Since the minimum inter-release time between jobs of τ_i is T_i, there are at most $\left\lceil \frac{t_c - (t_0 - \Delta_i - T_i)}{T_i} \right\rceil$ such jobs released in this interval. Since τ_i is a migrating task, the number of jobs executed on P_p out of any number of consecutive jobs of τ_i is limited by Prop. 6. Thus, the demand from τ_i in the interval $[t_0, t_c)$ on P_p is at most

$$\left(f_{i,p} \cdot \left(\left\lceil \frac{t_c - (t_0 - \Delta_i - T_i)}{T_i} \right\rceil - 1 \right) + 2 \right) C_i$$
$$< \{\text{Since } \lceil x \rceil < x + 1\}$$
$$\left(f_{i,p} \cdot \left(\left(\frac{t_c - (t_0 - \Delta_i - T_i)}{T_i} + 1 \right) - 1 \right) + 2 \right) C_i$$
$$= \{\text{Simplifying}\}$$
$$\left(f_{i,p} \cdot \left(\frac{t_c - t_0 + \Delta_i + T_i}{T_i} \right) + 2 \right) C_i$$
$$= \{\text{By (1)}\}$$
$$(s_{i,p})(t_c - t_0) + (s_{i,p})(\Delta_i + T_i) + 2C_i. \qquad \square$$

We now show that we can upper-bound the tardiness of a migrating task τ_ℓ by considering an *alternate job allocation in which all of its jobs execute on its first processor, P_p.* (For ease of understanding, we will be using the indices "ℓ" and "h" in the rest of this subsection to reflect *lower* and *higher* static priorities, respectively.) Note that Prop. 5 ensures that, when ignoring fixed tasks (as we do in this subsection), P_p has sufficient capacity to accommodate any jobs of τ_ℓ we may move to it from other processors. This is because there must exist a fixed task on P_p with utilization at least that of τ_ℓ. (Our usage of a worst-fit decreasing assignment strategy is crucially exploited here.)

Lemma 2. *If every job of migrating task τ_ℓ that executes on a non-first processor of τ_ℓ is moved to its first processor P_p, then no job of τ_ℓ will complete earlier. Moreover, if another migrating task τ_h executes on P_p, then such job moves do not affect it.*

Proof. If τ_ℓ shares P_p with another migrating task τ_h, then by the prioritization rules of EDF-os, τ_h is not impacted by moving jobs of τ_ℓ to P_p, since τ_h has higher priority than τ_ℓ (we are not changing the static prioritization of these tasks).

We now show that moving a single job $\tau_{\ell,k}$ of τ_ℓ to P_p cannot lessen the completion time of any job of τ_ℓ. By inducting over all such moves, the lemma follows.

Because job $\tau_{\ell,k}$ is being moved, it was originally executing on a non-first processor of τ_ℓ. Hence, $\tau_{\ell,k}$ was of highest priority on that processor and executed immediately to completion as soon as it was eligible (i.e., by the later of its release time and the completion time of its predecessor $\tau_{\ell,k-1}$, if any). After the move, its execution may be delayed by jobs of τ_h, which have higher priority than those of τ_ℓ on

P_p. Thus, after the move, $\tau_{\ell,k}$ cannot complete earlier, and may complete later. If it completes later, then this cannot cause subsequent jobs of τ_ℓ to complete earlier (earlier jobs of τ_ℓ are clearly not impacted). $\qquad\square$

Thm. 1 below provides tardiness bounds for migrating tasks. If a migrating task τ_ℓ shares its first processor with another migrating task τ_h, then the bound for τ_ℓ depends on that of τ_h. The bounds for all migrating tasks can be computed inductively, with the following lemma providing the base case.

Lemma 3. *The migrating task τ_h with the lowest-indexed first processor P_p does not share P_p with another migrating task.*

Proof. By the assignment procedure of EDF-os, no migrating task other than τ_h executes on P_p. $\qquad\square$

Theorem 1. *If τ_ℓ is the only task on its first processor P_p, then it will have no tardiness. Otherwise, it shares its first processor with exactly one task τ_h, and it has tardiness no larger than*

$$\Delta_\ell \triangleq \frac{(s_{h,p})(\Delta_h + T_h) + 2C_h + C_\ell}{1 - s_{h,p}} - T_\ell, \qquad (4)$$

where Δ_h is the maximum tardiness of τ_h.

Proof. Denote as P_p the first processor of τ_ℓ. By Lem. 2, we can establish the desired tardiness bound by assuming that all jobs of τ_ℓ run on P_p. We make this assumption in the remainder of the proof.

If τ_ℓ is the only migrating task on P_p, then its jobs will be of highest priority on P_p. Thus, by Prop. 2 and Lem. 2, τ_ℓ will never be tardy.

In the rest of the proof, we assume that τ_ℓ shares P_p with another migrating task. By Prop. 3, there is a unique such task τ_h, as stated in the theorem. By the prioritization rules used by EDF-os, τ_h has higher priority than τ_ℓ.

Consider job $\tau_{\ell,j}$ with release time $r_{\ell,j}$ and deadline $d_{\ell,j}$. For purposes of contradiction, assume that $\tau_{\ell,j}$'s tardiness exceeds Δ_ℓ. According to the prioritization rules used by EDF-os, $\tau_{\ell,j}$'s execution may be impacted only by jobs from τ_h and by jobs from τ_ℓ with deadlines before $d_{\ell,j}$. We now upper bound the processor demand impacting $\tau_{\ell,j}$ by considering a certain time interval, as defined next.

Interval $[t_0, t_c)$. Let t_0 be the latest point in time at or before $r_{\ell,j}$ such that no jobs of τ_h or τ_ℓ released on P_p before t_0 are pending; a released job is *pending* if it has not yet completed execution. (t_0 is well-defined because the stated condition holds at time 0.) Define $t_c \triangleq d_{\ell,j} + \Delta_\ell$. The assumption we seek to contradict is that $\tau_{\ell,j}$ does not complete by t_c. Since $\tau_{\ell,j}$ fails to complete by t_c, there are more than $t_c - t_0$ units of demand in the interval $[t_0, t_c)$ for the execution of jobs on P_p with priority at least that of $\tau_{\ell,j}$.

Demand from τ_h. By Lem. 1, the competing demand in $[t_0, t_c)$ due to τ_h on P_p is at most

$$(s_{h,p})(t_c - t_0) + (s_{h,p})(\Delta_h + T_h) + 2C_h. \qquad (5)$$

Demand from τ_ℓ. Additional demand can come from jobs of τ_ℓ with deadlines earlier than $d_{\ell,j}$. By the definition of t_0, all such jobs are released in $[t_0, r_{\ell,j})$. Thus, there are at most $\left\lfloor \frac{(r_{\ell,j} - t_0)}{T_\ell} \right\rfloor$ such jobs. Including job $\tau_{\ell,j}$ itself, there are at most $\left\lfloor \frac{(r_{\ell,j} - t_0)}{T_\ell} \right\rfloor + 1$ jobs of τ_ℓ released in $[t_0, t_c)$ with deadlines at most $d_{\ell,j}$. The total demand due to such jobs is $\left(\left\lfloor \frac{(r_{\ell,j} - t_0)}{T_\ell} \right\rfloor + 1 \right) C_\ell$, which by the definition of U_ℓ is at most

$$(U_\ell)(r_{\ell,j} - t_0) + C_\ell. \qquad (6)$$

Total demand. For notational convenience, let

$$K \triangleq (s_{h,p})(\Delta_h + T_h) + 2C_h + C_\ell. \qquad (7)$$

Then, by (5) and (6), the total demand on P_p due to jobs of equal or higher priority than $\tau_{\ell,j}$ in $[t_0, t_c)$ is at most

$$K + (t_c - t_0)s_{h,p} + (r_{\ell,j} - t_0)U_\ell. \qquad (8)$$

Because $\tau_{\ell,j}$ completed after time t_c (by assumption), the considered demand exceeds the length of the interval $[t_0, t_c)$, so

$$
\begin{aligned}
(t_c - t_0) &< \{\text{By (8)}\} \\
&\quad K + (t_c - t_0)s_{h,p} + (r_{\ell,j} - t_0)U_\ell \\
&= \{\text{Rearranging}\} \\
&\quad K + (t_c - r_{\ell,j})s_{h,p} + (r_{\ell,j} - t_0)(s_{h,p} + U_\ell) \\
&< \{\text{By Prop. 5}\} \\
&\quad K + (t_c - r_{\ell,j})s_{h,p} + (r_{\ell,j} - t_0). \qquad (9)
\end{aligned}
$$

Subtracting $(r_{\ell,j} - t_0)$ from both sides of (9) gives $(t_c - r_{\ell,j}) < K + (t_c - r_{\ell,j})s_{h,p}$, which implies

$$K > (t_c - r_{\ell,j})(1 - s_{h,p}). \qquad (10)$$

By Prop. 2, $U_h < 1$, and hence $s_{h,p} < 1$. Thus, by (10),

$$
\begin{aligned}
(t_c - r_{\ell,j}) &< \{\text{since } 1 - s_{h,p} \text{ is positive}\} \\
&\quad \frac{K}{1 - s_{h,p}} \\
&= \{\text{By (7)}\} \\
&\quad \frac{(s_{h,p})(\Delta_h + T_h) + 2C_h + C_\ell}{1 - s_{h,p}} \\
&= \{\text{By (4)}\} \\
&\quad \Delta_\ell + T_\ell.
\end{aligned}
$$

Because $r_{\ell,j} = d_{\ell,j} - T_\ell$, this implies $t_c - d_{\ell,j} < \Delta_\ell$, which contradicts our definition of $t_c = d_{\ell,j} + \Delta_\ell$. Thus, it cannot be the case that $\tau_{\ell,j}$ completed after time $d_{\ell,j} + \Delta_\ell$ as assumed. $\qquad\square$

4.2 Tardiness Bounds for Fixed Tasks

If no migrating tasks execute on a given processor, then the fixed tasks on that processor have zero tardiness, by the optimality of EDF on one processor. The following theorem

establishes tardiness bounds for fixed tasks that must execute together with migrating tasks.

Theorem 2. *Suppose that at least one migrating task executes on processor P_p and let τ_i be a fixed task on P_p. If P_p has two migrating tasks (refer to Prop. 3), denote them as τ_h and τ_ℓ, where τ_h has higher priority; otherwise, denote its single migrating task as τ_h, and consider τ_ℓ to be a "null" task with $T_\ell = 1$, $s_{\ell,p} = 0$, and $C_\ell = 0$. Then, τ_i has a maximum tardiness of at most*

$$\Delta_i \triangleq \frac{(s_{h,p})(\Delta_h + T_h) + 2C_h + (s_{\ell,p})(\Delta_\ell + T_\ell) + 2C_\ell}{(1 - s_{h,p} - s_{\ell,p})}. \quad (11)$$

Proof. This proof will be similar in structure to that of Thm. 1. We will upper bound demand over the following interval.

Interval $[t_0, t_c)$. For purposes of contradiction, suppose that there exists a job $\tau_{i,j}$ of τ_i that has tardiness exceeding Δ_i, i.e., $\tau_{i,j}$ has not completed by t_c, where $t_c \triangleq d_{i,j} + \Delta_i$. Define a job as a *competing* job if it is released on P_p and it is a job of τ_h or τ_ℓ, or a job of a fixed task that has a deadline at or before $d_{i,j}$. Let t_0 be the latest point in time at or before $r_{i,j}$ such that no competing jobs released before t_0 are pending. (t_0 is well-defined because the stated condition holds at time 0.) We now bound demand over $[t_0, t_c)$ due to competing jobs (including $\tau_{i,j}$ itself) by considering migrating and fixed tasks separately.

Demand from migrating tasks. By Lem. 1, demand over $[t_0, t_c)$ due to jobs of τ_h and τ_ℓ is at most

$$(s_{h,p})(t_c - t_0) + (s_{h,p})(\Delta_h + T_h) + 2C_h + \\ (s_{\ell,p})(t_c - t_0) + (s_{\ell,p})(\Delta_\ell + T_\ell) + 2C_\ell. \quad (12)$$

Demand from fixed tasks. A fixed task τ_k can release at most $\left\lfloor \frac{d_{i,j} - t_0}{T_k} \right\rfloor$ competing jobs within $[t_0, t_c)$. Thus, demand from all competing jobs of fixed tasks is at most

$$\sum_{\tau_k \in \tau_p^f} \left\lfloor \frac{d_{i,j} - t_0}{T_k} \right\rfloor C_k \leq (d_{i,j} - t_0) \sum_{\tau_k \in \tau_p^f} \frac{C_k}{T_k}. \quad (13)$$

By the definition of σ_p^f, (13) can be rewritten as

$$(d_{i,j} - t_0)(\sigma_p^f) \leq \{\text{By Prop. 4}\} \\ (d_{i,j} - t_0)(1 - s_{h,p} - s_{\ell,p}). \quad (14)$$

Total demand. For notational convenience, let

$$K \triangleq (s_{h,p})(\Delta_h + T_h) + 2C_h + (s_{\ell,p})(\Delta_\ell + T_\ell) + 2C_\ell. \quad (15)$$

Then, by (12) and (14), total competing demand is at most

$$K + s_{h,p}(t_c - t_0) + s_{\ell,p}(t_c - t_0) + \\ (d_{i,j} - t_0)(1 - s_{h,p} - s_{\ell,p}). \quad (16)$$

Because $\tau_{i,j}$ completed after time t_c (by assumption), the considered demand exceeds the length of the interval $[t_0, t_c)$, so

$$(t_c - t_0) < \{\text{By (16)}\} \\ \quad K + s_{h,p}(t_c - t_0) + s_{\ell,p}(t_c - t_0) + \\ \quad (d_{i,j} - t_0) - (d_{i,j} - t_0)(s_{h,p} + s_{\ell,p}) \\ = \{\text{Rearranging}\} \\ \quad K + (s_{h,p} + s_{\ell,p})(t_c - t_0) + \\ \quad (d_{i,j} - t_0) - (d_{i,j} - t_0)(s_{h,p} + s_{\ell,p}). \quad (17)$$

Subtracting $(d_{i,j} - t_0)$ from both sides of (17), we have $(t_c - d_{i,j}) < K + (s_{h,p} + s_{\ell,p})(t_c - t_0) - (d_{i,j} - t_0)(s_{h,p} + s_{\ell,p}) = K + (s_{h,p} + s_{\ell,p})(t_c - d_{i,j})$. This implies

$$K > (t_c - d_{i,j})(1 - s_{h,p} - s_{\ell,p}). \quad (18)$$

By Prop. 4 and because at least one fixed task τ_i assigned to P_k, we have $(1 - s_{h,p} - s_{\ell,p}) > 0$. Thus, by (18),

$$t_c - d_{i,j} < \frac{K}{(1 - s_{h,p} - s_{\ell,p})} \\ = \{\text{By (11) and (15)}\} \\ \Delta_i.$$

This contradicts our definition of $t_c = d_{i,j} + \Delta_i$, so it cannot be the case that $\tau_{i,j}$ has more than Δ_i units of tardiness. $\square$

5 Discussion

In this section, we discuss several issues motivated by our analysis as well as extensions to our results.

Refined assignment procedures. Our analysis suggests that, by refining EDF-os's assignment procedure, it may possible to obtain lower tardiness bounds. First, note that the inductive nature of the tardiness bound calculation for migrating tasks may cause migrating tasks assigned to later processors to have higher bounds because tardiness can cascade (though it will remain bounded). It may be possible to reduce such cascades by adjusting the assignment of migrating tasks, particularly on systems that are not fully utilized. Second, note that reducing the shares of migrating tasks executing on P_p reduces the bounds in (4) and (11). However, such a reduction would entail increasing the shares of these tasks on other processors, which could lead to tardiness bound increases on those processors. It may be possible to take such linkages among processors into account and obtain an assignment of tasks to processors that lessens the largest tardiness bound in the system. Finally, note that (4) includes $-T_\ell$ as part of τ_ℓ's tardiness bound. By biasing the task assignment procedure to prefer larger periods for migrating tasks, it might be possible to lessen the tardiness bounds that result. We leave refinements to our assignment procedure motivated by these observations as future work.

Bounds with non-implicit deadlines. We have so far assumed that all job deadlines are implicit. However, if we

maintain the prioritizations that EDF-os uses, then bounded tardiness can be easily ensured for systems with non-implicit deadlines, i.e., ones where each task τ_i has a specified relative D_i that may be less than, equal to, or greater than T_i. Maintaining the existing prioritizations for migrating tasks is straightforward, as these tasks are not scheduled by deadline (that are statically prioritized). For each job $\tau_{i,j}$ of a fixed task τ_i, we merely need to define a "scheduling deadline" equal to $r_{i,j} + T_i$ and prioritize such jobs on an EDF basis using scheduling deadlines instead of real ones. With this change, EDF-os will behave as before, but our analysis then bounds tardiness (for both migrating and fixed tasks) with respect to scheduling deadlines. However, they can be easily corrected to be expressed with respect to real deadlines: if $D_i > T_i$, then simply subtract $D_i - T_i$ from the bound; if $D_i < T_i$, then simply add $T_i - D_i$ to the bound.

Window constrained second-level schedulers In defining EDF-os, we used EDF as a secondary scheduler for fixed tasks. (For migrating tasks, our prioritization rules and the sporadic task model fully characterize the behavior.)

Optimal variants of EDF-os can be constructed in which other algorithms are used as the secondary scheduler. All that we require is that a *window-constrained* [22] scheduler be used. Such a scheduler employs a per-task priority function $\chi_i(\tau_{i,j}, t)$ such that for some constants ϕ_i and ψ_i, $r_{i,j} - \phi_i \le \chi_i(\tau_{i,j}, t) \le d_{i,j} + \psi_i$ for each job $\tau_{i,j}$. The priority of job $\tau_{i,j}$ is at least that of $\tau_{i',j'}$ at time t if $\chi_i(\tau_{i,j}, t) \le \chi_{i'}(\tau_{i',j'}, t)$ (priority functions can potentially change with time).

Our analysis can be modified to deal with this more general priority specification as follows. The bounds for migrating tasks continue to hold without modification; the proof of Thm. 1 is unchanged. By the definition of EDF-os priorities, all jobs of τ_h always have a higher priority than $\tau_{\ell,j}$, and by the sporadic task model, no job of τ_ℓ released after $r_{\ell,j}$ is eligible for execution before $\tau_{\ell,j}$ completes.

However, in our analysis of the tardiness of fixed tasks in the proof of Theorem 2, the number of competing jobs due to a fixed task τ_k is no longer $\left\lfloor \frac{d_{i,j} - t_0}{T_k} \right\rfloor$ but $\left\lfloor \frac{d_{i,j} + \psi_i + \phi_k - t_0}{T_k} \right\rfloor$. In effect, this change causes "$d_{i,j}$" to be replaced by "$d_{i,j} + \psi_i$" throughout the proof and K to be inflated by an additional $\sum_{\tau_k \in \tau_p^f} U_k \phi_k$. As a result, Δ_i must be increased by $\psi_i + \frac{\sum_{\tau_k \in \tau_p^f} U_k \phi_k}{1 - s_{h,p} - s_{\ell,p}}$. Tighter analysis may be possible if more is known about the prioritization function (as was the case with EDF).

Non-preemptivity. After designing EDF-os, we initially thought it retained its optimality if job execution is non-preemptive. However, this turns out not to be the case. In particular, with non-preemptivity, a job of a migrating task executing on a non-first processor for that task may be non-preemptively blocked when it is released. This blocking negates an important property exploited in our analysis, namely that such jobs execute immediately upon release. In an online appendix (available at http://www.cs.unc.edu/~anderson/papers.html), we give a counterexample consisting of five tasks executing on three processors where such non-preemptive blocking causes a migrating task to have unbounded tardiness.

6 Experimental Comparison

To assess the improvement of EDF-os over EDF-fm, we conducted experiments in which the two algorithms were compared both on the basis of schedulability and the tardiness bounds ensured by each. Due to space constraints, we only discuss the tardiness bound experiments here. It is self-evident that EDF-os is preferable with respect to schedulability, as it is optimal and EDF-fm is not.

In our tardiness bound experiments, we randomly generated task sets and computed for each task set the maximum tardiness bound under each of EDF-os and EDF-fm. Our parameter distributions for the generated task sets were based on prior studies, e.g., [12]. However, in order to be certain that any generated task set was schedulable by EDF-fm, we restricted the utilization of each task to be at most 0.5. We used both *uniform* and *bimodal* utilization distributions. For uniform distributions, we considered a *light* distribution where values were drawn from $[0.001, 0.1]$ and a *medium* distribution where they were drawn from $[0.1, 0.4]$. (We did not test a heavy distribution to avoid creating utilizations exceeding 0.5.) For bimodal distributions, we drew values uniformly in the range of either $[0.001, 0.1]$ or $[0.25, 0.499]$ with respective probabilities of $\frac{8}{9}$ and $\frac{1}{9}$, $\frac{6}{9}$ and $\frac{3}{9}$, and $\frac{4}{9}$ and $\frac{5}{9}$, for light, medium, and heavy distributions, respectively. We generated periods uniformly from either a *short* (3 ms to 33 ms), *medium* (10 ms to 100 ms), or *long* (50 ms to 250 ms) distribution.

For each combination of period and utilization distribution, we generated 100,000 task sets and computed the maximum tardiness bound of each task set under each of EDF-fm and EDF-os. When generating each task set, we added tasks until the task set became infeasible on eight processors, and then removed the last task.

Fig. 12 shows the percent mean reduction in the maximum tardiness bound for each experiment. Across all tests, EDF-os achieved an average tardiness reduction of 84% over EDF-fm. Therefore, it is a substantial improvement over EDF-fm even for systems that EDF-fm can schedule. One major reason for this improvement is that EDF-os can often generate a schedule with no migrating tasks (due to its bin-packing heuristic), resulting in no tardiness for any task. Although (unlike EDF-fm) EDF-os allows migrating tasks to be tardy, the first migrating task will have no tardiness, and there are few migrating tasks to "cascade" tardiness.

7 Conclusion

We have presented EDF-os, the first boundary-limited semi-partitioned scheduling algorithm that is optimal under the "bounded tardiness" definition of SRT correctness. We have also discussed optimal variants of EDF-os in which implicit deadlines are not assumed and in which algorithms other than EDF are used as the secondary scheduler. EDF-os and its analysis extend prior work on EDF-fm by intro-
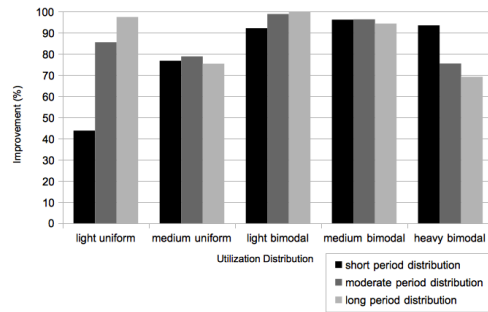
Figure 12: Percent tardiness reduction of EDF-os over EDF-fm.

ducing two new key ideas: using some static prioritizations to make the execution of migrating tasks more predictable; and exploiting properties of worst-fit decreasing task assignments to enable a migrating task to be analyzed by "pretending" that all of its jobs execute on its first processor. In experiments that we conducted, EDF-os proved to be better than EDF-fm not only with respect to schedulability, but also tardiness bounds that can be guaranteed.

The only other optimal boundary-limited scheduling algorithms for SRT systems known to us are non-preemptive global EDF (NP-G-EDF) [10] and global FIFO (G-FIFO) [21] (which is also non-preemptive). For static systems, EDF-os is likely to be preferable in practice, because the tardiness bounds we have established are lower than those known for NP-G-EDF and G-FIFO, and because semi-partitioned algorithms have lower runtime overheads than global ones [5]. On the other hand, for dynamic systems, where task timing parameters (such as execution budgets and periods) may change at runtime, NP-G-EDF is likely to be preferable, as EDF-based global scheduling tends to more amenable to runtime changes [9]. In contrast, the correctness of EDF-os relies crucially on how tasks are assigned to processors, and redefining such assignments on-the-fly does not seem easy.

In future work, we would like to implement EDF-os and conduct experiments in which it is compared to other alternatives on the basis of schedulability with measured overheads considered. We would also like to determine conditions under which bounded tardiness can be ensured under EDF-os if non-preemptive code regions exist or if locking protocols are used to access shared resources.

References

[1] J. Anderson, V. Bud, and U. Devi. An EDF-based restricted-migration scheduling algorithm for multiprocessor soft real-time systems. *Real-Time Sys.*, 38(2):85–131, 2008.

[2] B. Andersson, K. Bletsas, and S. Baruah. Scheduling arbitrary-deadline sporadic task systems on multiprocessors. In *Proc. of the 29th IEEE Real-Time Sys. Symp.*, pages 385–394, 2008.

[3] B. Andersson and E. Tovar. Multiprocessor scheduling with few preemptions. In *Proc. of the 12th IEEE Int'l Conf. on Embedded and Real-Time Computing Sys. and Applications*, pages 322–334, 2006.

[4] S. Baruah, N. Cohen, C.G. Plaxton, and D. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15(6):600–625, 1996.

[5] A. Bastoni. *Towards the Integration of Theory and Practice in Mul-*

tiprocessor Real-Time Scheduling. PhD thesis, University of Rome 'Tor Vergata', Rome, Italy, 2011.

[6] M. Bhatti, C. Belleudy, and M. Auguin. A semi-partitioned real-time scheduling approach for periodic task systems on multicore platforms. In *Proc. of the 27th Annual ACM Symp. on Applied Computing*, pages 1594–1601, 2012.

[7] K. Bletsas and B. Andersson. Notional processors: an approach for multiprocessor scheduling. In *Proc. of the 15th IEEE Real-Time and Embedded Technology and Applications Symp.*, pages 3–12, 2009.

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

[9] A. Block. *Multiprocessor Adaptive Real-Time Systems*. PhD thesis, University of North Carolina, Chapel Hill, NC, 2008.

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

[11] 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.

[12] J.P. Erickson and J.H. Anderson. Fair lateness scheduling: Reducing maximum lateness in g-edf-like scheduling. In *Real-Time Sys. (ECRTS), 2012 24th Euromicro Conf. on*, pages 3–12, 2012.

[13] M. Fan and G. Quan. Harmonic semi-partitioned scheduling for fixed-priority real-time tasks on multi-core platform. In *Proc. of the Design, Automation & Test in Europe Conf. & Exhibition (DATE)*, pages 503–508, 2012.

[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 *Proc. of the 20th Int'l Conf. on Real-Time and Network Sys.*, pages 141–150, 2012.

[15] N. Guan, M. Stigge, W. Yi, and G. Yu. Fixed-priority multiprocessor scheduling: Beyond Liu & Layland utilization bound. In *Proc. of the 31st Real-Time Sys. Symp. Work in Progress Session*, 2010.

[16] N. Guan, M. Stigge, W. Yi, and G. Yu. Fixed-priority multiprocessor scheduling with Liu and Layland's utilization bound. In *Proc. of the 16th IEEE Real-Time and Embedded Technology and Applications Symp.*, pages 165–174, 2010.

[17] S. Kato and N. Yamasaki. Real-time scheduling with task splitting on multiprocessors. In *Proc. of the 13th IEEE Int'l Conf. on Embedded and Real-Time Computing Sys. and Applications*, pages 441–450, 2007.

[18] S. Kato and N. Yamasaki. Portioned edf-based scheduling on multiprocessors. In *Proc. of the 8th ACM Int'l Conf. on Embedded software*, pages 139–148, 2008.

[19] S. Kato and N. Yamasaki. Semi-partitioning technique for multiprocessor real-time scheduling. In *Proc. of the 29th IEEE Real-Time Sys. Symp. Work in Progress Session*, 2008.

[20] S. Kato and N. Yamasaki. Semi-partitioned fixed-priority scheduling on multiprocessors. In *Proc. of the 15th IEEE Real-Time and Embedded Technology and Applications Symp.*, pages 23–32, 2009.

[21] H. Leontyev and J. Anderson. Tardiness bounds for FIFO scheduling on multiprocessors. In *Proc. of the 19th Euromicro Conf. on Real-Time Sys.*, pages 71–80, 2007.

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

[23] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[24] A. Mok. *Fundamental Design Problems of Distributed Systems for Hard Real-Time Environments*. Cambridge, Mass., 1983.

[25] P. Regnier, G. Lima, E. Massa, G. Levin, and S. Brandt. Run: Optimal multiprocessor real-time scheduling via reduction to uniprocessor. In *Proc. of the 32nd IEEE Real-Time Sys. Symp.*, pages 104–115, 2011.

[26] A. Sarkar, PP Chakrabarti, and S. Ghose. Partition oriented frame based fair scheduler. *Journal of Parallel and Distributed Computing*, 70(7):707–718, 2010.