Soft Real-Time Semi-Partitioned Scheduling with Restricted Migrations on Uniform Heterogeneous Multiprocessors *

Kecheng Yang and James H. Anderson
Department of Computer Science
University of North Carolina at Chapel Hill
{yangk,anderson}@cs.unc.edu

ABSTRACT

We present EDF-sh, which is the first soft real-time scheduling algorithm with restricted migrations for heterogeneous multiprocessors. EDF-sh does not restrict total utilization as long as the system is not overutilized. However, it requires a per-task utilization constraint, which is not too constraining but nonetheless renders EDF-sh non-optimal. We evaluate the effectiveness of EDF-sh by means of schedulability experiments. In these experiments, more than 87% of the feasible task sets that were considered were soft-real-time-schedulable under EDF-sh. Additionally, tardiness bounds for these task sets under EDF-sh were found to be quite low in almost all cases.

1. INTRODUCTION

Most multiprocessor real-time scheduling research pertains to multiprocessors where every processor is of the same speed. However, heterogeneous platforms are becoming increasingly common. For example, ARM recently proposed a new CPU technology called big.LITTLE [1], which integrates relatively slower, low-power processors with faster, high-power ones to balance performance and energy efficiency. This heterogeneous computing architecture is being used by Samsung in their new mobile SoC, Exynos 5422, which consists of four slower ARM Cortex-A7 cores and four faster ARM Cortex-A15 cores [2]. As another example, CPU frequency scaling, e.g., *cpufrequtils* in Linux, may cause a homogeneous multiprocessor to function as a heterogeneous one.

Unfortunately, schedulability analysis for heterogeneous multiprocessors is much harder than for homogeneous ones, because such analysis must consider not only *which* tasks are executing but also *where* they execute. Therefore, the existing literature pertaining to real-time scheduling on heterogeneous systems is much more limited than that pertaining to homogeneous ones. In this paper, we propose a new scheduling algorithm for such heterogeneous multiprocessors. This algorithm is designed for soft real-time (SRT) systems where some deadlines may be missed as long as deadline tardiness is bounded. In contrast, most prior work on heterogeneous multiprocessor scheduling has instead focused on hard real-time (HRT) constraints.

Prior Work. We limit our attention to uniform multiprocessor systems in this paper. That is, each processor may have a different *speed*, or clock frequency, but every processor is of the same *functionality*. See Sec. 2.1 for a more detailed taxonomy of multiprocessor platforms.

In [18], Funk et al. established a feasibility condition for scheduling periodic tasks upon uniform heterogeneous multiprocessors by leveraging the Level Algorithm [22]; this condition applies to sporadic tasks as well. Subsequently, Baruah et al. considered earliest-deadline-first (EDF) scheduling [7] and rate-monotonic (RM) scheduling [8] upon uniform heterogeneous multiprocessors. Also, Funk proposed three EDF-based schedulers for uniform heterogeneous multiprocessors with different migration constraints: f-EDF (full migration), p-EDF (partitioned, no migration), and r-EDF (restricted migration) [19]. In a restricted-migration algorithm, migrations are *boundary-limited*: a task can only migrate at job boundaries, i.e., between task invocations. All of these prior algorithms are directed at HRT scheduling, and each requires that total utilization be capped. The resulting utilization loss means that the underlying platform cannot be fully utilized.

In work on SRT systems, Devi et al. showed that under global EDF (GEDF) scheduling any feasible sporadic task system has bounded tardiness [13], i.e., GEDF is SRT-optimal. Thereafter, Erickson et al. derived better tardiness bounds for GEDF [15], and Leontyev et al. established such bounds for a class of global schedulers called *window-constrained schedulers* [27]. Finally, Erickson et al. developed the global fair-lateness (GFL) scheduling algorithm [16], and showed that it has a lower maximum tardiness bound than GEDF but a similar implementation to GEDF.

While this SRT research focused on global scheduling, global schedulers can potentially migrate tasks frequently, causing significant overheads in practice. An alternative is semi-partitioned scheduling. Under semi-partitioned scheduling, most tasks are assigned to processors, like in partitioned scheduling; however, those tasks that cannot be feasibly assigned to a single processor are allowed to migrate. EDF-fm [3] was the first semi-partitioned scheduling algorithm.

*Work supported by NSF grants CNS 1016954, CNS 1115284, CNS 1218693, and CNS 1239135, AFOSR grant FA9550-14-1-0161, and a grant from General Motors Corp.

RTNS 2014, October 8 - 10 2014, Versailles, France

http://dx.doi.org/10.1145/2659787.2659808.

Subsequently, numerous other such algorithms were proposed [5, 6, 9, 10, 11, 14, 17, 20, 21, 23, 24, 25, 30, 31], in which various migration strategies were considered. Recently, Anderson et al. proposed EDF-os [4], a SRT-optimal semi-partitioned scheduling algorithm. Migrations in EDF-os are boundary-limited. This is a desirable property because migrations within a job can be expensive in practice and also can cause problems for locking protocols [12].

All of the SRT work cited above pertains to homogeneous systems. SRT scheduling on heterogeneous multiprocessor platforms was first considered by Leontyev et al. in designing EDF-ms [26]; to the best of our knowledge, EDF-ms is the only prior algorithm proposed for SRT heterogeneous systems. EDF-ms is intended for multiprocessors with a large number of cores, where cores of the same speed are grouped together and every group has at least two cores. The semi-partitioned scheduler EDF-fm is applied at the group level, but within each group, the GEDF scheduler is used. Therefore, inter-group migrations are boundary-limited while intra-group migrations are not.

Contributions. In this paper, we propose an EDF-based semi-partitioned algorithm for scheduling SRT sporadic tasks upon uniform heterogeneous multiprocessors. We design it by generalizing EDF-os for use on heterogeneous multiprocessors. Our proposed algorithm, called EDF-sh (earliest-deadline-first-based semi-partitioned scheduling upon uniform heterogeneous multiprocessors), ensures bounded tardiness and has boundary-limited migrations, like EDF-os. Moreover, EDF-sh does not restrict total utilization as long as the system is not overutilized, though it does require a per-task utilization cap. To the best of our knowledge, this is the first SRT scheduling algorithm with boundary-limited migrations for heterogeneous systems. Also, unlike EDF-ms [26], restrictions on the hardware platform are not required.

We evaluate EDF-sh from both schedulability and tardiness-bound perspectives. In terms of SRT schedulability, EDF-sh theoretically dominates EDF-ms. To better assess the efficiency of EDF-sh, we conducted experiments in which schedulability was checked and tardiness bounds were computed for randomly generated feasible task sets on certain heterogeneous platforms. In these experiments, more than 87% of the considered task sets were SRT-schedulable under EDF-sh. As for tardiness bounds, in most cases, EDF-sh exhibited significantly lower tardiness bounds than EDF-ms.

Organization. In Sec. 2, we provide necessary background. Then, in Sec. 3 we describe EDF-sh in detail. We present our tardiness-bound proof in Sec. 4. and experimentally evaluate EDF-sh in Sec. 5. We conclude and suggest future work in Sec. 6.

2. BACKGROUND

We consider scheduling n sporadic tasks on m processors, where $n \geq m$. We also assume implicit deadlines, i.e., each task has a relative deadline equal to its period. Thus, a task τ_i can be specified by (C_i, T_i), where C_i is its *worst-case execution requirement*[1] and T_i is its *period*. We define the *utilization* of a task τ_i as

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

Note that on heterogeneous multiprocessors, u_i may exceed 1.0. We will give needed restrictions on utilization later in Sec. 2.3.

A *job* is an invocation of a task; the j^{th} job of task τ_i is denoted $\tau_{i,j}$. $r_{i,j}$ is its release time and $d_{i,j}$ is its absolute deadline, where $d_{i,j} = r_{i,j} + T_i$. The *tardiness* of a job $\tau_{i,j}$ that completes at time t_c is defined as $\max\{0, t_c - d_{i,j}\}$, while its *lateness* is $t_c - d_{i,j}$. The two differ only if $\tau_{i,j}$ completes before its deadline, in which case its tardiness is zero but its lateness is negative. In this paper, a task system is considered to be *SRT-schedulable* under a given scheduling algorithm if each task can be guaranteed bounded tardiness under that algorithm. The *speed* of a processor refers to the amount of work completed in one time unit when a job is executed on that processor.

2.1 A Taxonomy of Multiprocessors

In terms of heterogeneity, multiprocessors can be classified as follows [19, 29]:

- **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 task 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 $s_{p,i}$.

In this paper, we focus on uniform heterogeneous multiprocessors.

2.2 EDF-os

EDF-os [4] is an EDF-based semi-partition SRT-optimal scheduling algorithm for identical multiprocessors. The term *SRT-optimal* means that, for any feasible system, bounded deadline tardiness for every job is ensured. EDF-os has two phases: an *assignment phase* and an *execution phase*. Each task has a reserved amount of capacity, or *share*, on one or more processors. $\psi_{i,p}$ denotes the share (which potentially can be zero) of task τ_i on processor p. The total share allocation on processor p is denoted $\sigma_p \stackrel{\text{def}}{=} \sum_{k=1}^{n} \psi_{k,p}$. EDF-os maintains that no processor is overutilized, i.e., $\sigma_p \leq 1.0$ holds for all p. Also, the total share allocation of a task τ_i matches its utilization, i.e., $\sum_{k=1}^{m} \psi_{i,k} = u_i$ (note that, on an identical multiprocessor, $u_i \leq 1.0$). If a task has non-zero shares on more than one processor, then it is a *migrating task*, otherwise it is a *fixed task*. We use $f_{i,p}$ to denote the long-term fraction of task τ_i's jobs that execute on processor p. $f_{i,p}$ is commensurate with the share allocated:

$$f_{i,p} = \frac{\psi_{i,p}}{u_i}. \tag{2}$$

[1]Techniques from [28] can be applied to instead provision tasks on an average-case basis; we do not consider such techniques further due to space constraints.

The set of all fixed tasks on processor p is denoted τ_p^f, and $\sigma_p^f \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau_p^f} \psi_{i,p}$.

In the assignment phase, EDF-os considers tasks in non-increasing-utilization order in the following two steps.

- First, it uses a worst-fit bin-packing heuristic to assign as many tasks as possible to be fixed.
- Second, it considers the remaining tasks to be assigned to processors in turn, and allocates these tasks on either one (in which case, the task is fixed) or more (in which case, the task is migrating) processors.

The processor with the lowest index where a migrating task is allocated is called its *first* processor.

In the execution phase, the scheduling rules are as follows on each processor.

- Jobs of migrating tasks are statically prioritized over those of fixed tasks.
- Jobs of fixed tasks are prioritized against each other on an EDF basis.
- On a migrating task's first processor, its priority is lower than other migrating tasks, but still higher than fixed ones.

To achieve the goal of boundary-limited migrations, EDF-os assigns jobs to processors and a single job executes only on the processor to which it is assigned without migration. To properly maintain (2), EDF-os uses a mechanism to fairly assign the jobs of a migrating task to guarantee Prop. 1 below. This scheme was first used by EDF-fm [3].

PROPERTY 1. *For the first z jobs of task τ_i, at least $\lfloor f_{i,p} \cdot z \rfloor$ and at most $\lceil f_{i,p} \cdot z \rceil$ of them are assigned to processor p.*

Example 1. Consider scheduling the task set $\tau = \{(5,6), (6,9), (4,6), (2,3), (2,3), (10,30), (1,6)\}$ (tasks are listed in non-increasing-utilization order) on four identical processors. Fig. 1 depicts the task assignment used by EDF-os. In the first step of the EDF-os assignment phase, the first four tasks are assigned to the four processors as fixed tasks. In the second step, the fifth task needs capacity from processors 1, 2, and 3 to be allocated, so it is a migrating task that assigns jobs to processors 1, 2, and 3. Similarly, τ_6 is a migrating task, because it has non-zero shares on both processors 3 and 4. However, the last task τ_7 is a fixed task since processor 4 is the only processor on which τ_7 has a non-zero share. For the two migrating tasks, processor 1 is the first processor of τ_5, while processor 3 is the first processor of τ_6.

2.3 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 ($1 \le p \le m$, $s_p \in \mathbb{R}$). Also, we index the processors in non-increasing-speed order, i.e., $\pi = \{s_1, s_2, \cdots, s_m\}$, where $s_p \ge s_{p+1}$ 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 $u_i \ge u_{i+1}$ for $i \in \{1, 2, \cdots, n-1\}$.

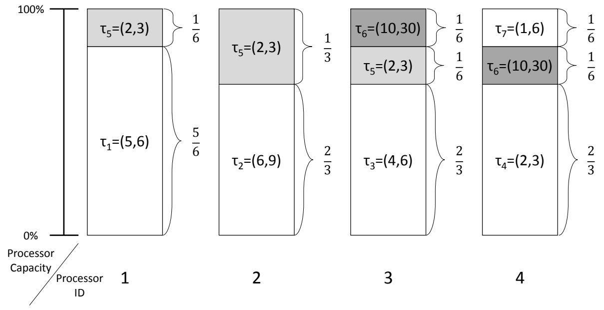


Figure 1: EDF-os task assignment for Ex. 1.

Let $U_k \stackrel{\text{def}}{=} \sum_{i=1}^{k} u_i$ and $S_k \stackrel{\text{def}}{=} \sum_{i=1}^{k} s_i$. By leveraging the Level Algorithm [22], Funk et al. [18] showed that an implicit-deadline periodic task system τ can be successfully scheduled on a uniform heterogeneous multiprocessor system π without missing any deadlines if and only if

$$U_n \le S_m \tag{3}$$

and

$$U_k \le S_k \quad \text{for} \quad k = 1, 2, \cdots, m-1. \tag{4}$$

In fact, the proof in [18] also shows (3) and (4) are a necessary and sufficient feasibility condition for implicit-deadline sporadic task systems.

In this paper, we further restrict task utilizations slightly by requiring

$$\sum_{u_i > s_k} u_i \le \sum_{s_p > s_k} s_p \quad \text{for} \quad k = 1, 2, \cdots, m. \tag{5}$$

Nevertheless, the total utilization U_n can be as large as the total speed S_m.

Note that (5) implies (4) (this can be proved by induction but we omit the proof due to space constraints). Thus, we omit (4) and hence let (3) and (5) be our task system utilization restriction in this paper.

3. ALGORITHM EDF-SH

We design EDF-sh by extending EDF-os to uniform heterogeneous multiprocessors. As a result, EDF-sh inherits most of the advantages of EDF-os, such as:

- Under EDF-sh, every job has bounded tardiness.
- Migrations are boundary-limited.
- The underlying platform can be fully utilized, i.e., U_n can be as large as S_m.

In the tardiness-bound proof for EDF-os, and for EDF-sh here, it is essential that each task executes only on processors that have a speed at least its utilization without overutilizing any processor. Unfortunately, this cannot be ensured for all feasible task systems (recall (3) and (4)). For example, a task system $\tau = \{(2,1), (2,1)\}$ to be scheduled on $\pi = \{3,1\}$ is feasible, but if we assign jobs of each task only to processors with a speed at least its utilization, then the first processor will be overutilized. Because of this difficulty, we assume (5), which is a little more restrictive than (4).

Besides this utilization restriction, we employ the following modifications:

initially $\psi_{i,p} = 0$ and $\sigma_p = 0$ for all i and p;
index tasks in a non-increasing-utilization order;
index processors in a non-increasing-speed order;
/* p is the index of the last processor to which a migrating task was assigned (or 1, if no migrating task has been assigned yet). s_p is the first processor for next migrating task if its capacity has not been exhausted yet. */
$p := 1$;
for $i := 1$ **to** n **do**
  /* If task τ_i can be fixed, then we assign it to be fixed task via worst-fit here. */
  Select k that $s_k - \sigma_k$ is maximal;
  if $s_k - \sigma_k \geq u_i$ **then**
    $\psi_{i,k} = u_i$;
    $\sigma_k = \sigma_k + u_i$;
  else
    /* If task τ_i has to migrate, then we assign its shares on processors to exhaust processor capacities in turn from the fastest one to the slowest one. */
    $remaining := u_i$;
    repeat
      $\psi_{i,p} := \min(remaining, (s_p - \sigma_p))$;
      $\sigma_p := \sigma_p + \psi_{i,p}$;
      $remaining := remaining - \psi_{i,p}$;
      if $\sigma_p = s_p$ **then**
        $p := p + 1$;
      end
    until $remaining = 0$;
  end
end

Algorithm 1: EDF-sh task assignment phase

- We have a different assignment phase, which consists of a single step instead of two. In our assignment phase, we always assign the currently considered task to be fixed if possible, regardless of whether an initial migrating task has been identified.

- Rather than statically giving a migrating task the highest priority on any processor that is not its *first* processor, we statically give it the highest priority on any processor that is not its *last* processor, which is the largest-indexed processor where it has a non-zero share.

Similarly to EDF-os, EDF-sh has two phases, an assignment phase and an execution phase. In the assignment phase, we consider tasks in non-increasing-utilization order. When considering a task, we first check the current available capacity of each processor to see if this task can be fixed. If so, we assign this task to some processor as a fixed task via a bin-packing heuristic. The specific heuristic does not matter in terms of theoretical schedulability; we choose to use worst-fit here. The pseudo-code in Algorithm 1 defines the assignment phase of EDF-sh.

Example 2. To illustrate the difference between the assignment phases of EDF-os and EDF-sh, we revisit the system in Ex. 1. Note that any identical multiprocessor is a

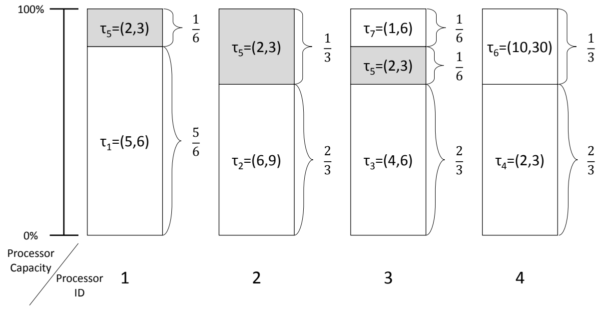


Figure 2: EDF-sh task assignment for Ex. 2. This is the same system as in Ex. 1, but EDF-sh has a different assignment from EDF-os.

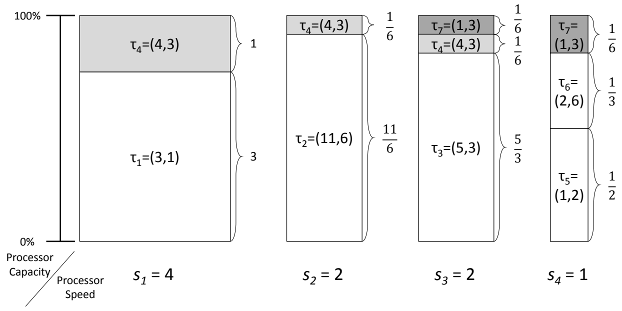


Figure 3: EDF-sh task assignment for Ex. 3. The width of each column indicates the processor speed.

uniform heterogeneous one (with $s_p = 1$ for all p), so EDF-sh also works on identical multiprocessors. The assignment of the first five tasks by EDF-sh is exactly the same as that by EDF-os. However, we will attempt to make all remaining tasks fixed as well, and this results in τ_6 being fixed on processor 4 and thereafter τ_7 being fixed on processor 3. That is, EDF-sh will have only one migrating task for this system. Fig. 2 shows the resulting assignment by EDF-sh.

Example 3. We now give an example of the task assignment phase of EDF-sh for the case where processor speeds are different. In this example, we have a uniform heterogeneous multiprocessor system $\pi = \{4, 2, 2, 1\}$, upon which a set of sporadic tasks $\tau = \{(3,1), (11,6), (5,3), (4,3), (1,2), (2,6), (1,3)\}$ will be scheduled. Via the worst-fit heuristic, τ_1, τ_2, and τ_3 are assigned as fixed tasks to s_1, s_2, and s_3, respectively. Thereafter, no single processor has enough capacity to fix τ_4, so τ_4 must migrate. It is assigned non-zero shares on s_1, s_2, and s_3. However, next, τ_5 and τ_6 can be fixed, specifically on s_4. Finally, τ_7 must migrate between s_3 and s_4. The resulting task assignment is depicted in Fig. 3. For the two migrating tasks, s_3 is the last processor of τ_4, and s_4 is the last processor of τ_7.

The assignment phase of EDF-sh ensures the following properties.

PROPERTY 2. *There are no more than two migrating tasks on s_p. If there are exactly two migrating tasks on s_p, then s_p is the last processor for exactly one of them.*

This property follows from the assignment procedure, and it can be proved by induction.

By Prop. 2, we know that a processor s_p will have at most two migrating tasks, and if exactly two, then they must be of different priorities. Therefore, if there is only one migrating task on a given processor s_p, then we use τ_l to denote that task; if there are two, then we let τ_l (τ_h) denote the migrating task with lower (higher) priority.

Property 3. *For any processor s_p, $\sigma_p^f+\psi_{h,p}+\psi_{l,p} \le s_p$. (If for s_p, τ_h and/or τ_l do not exist, then we just consider $\psi_{h,p}$ and/or $\psi_{l,p}$ to be zero.)*

This property holds because in our assignment phase, we do not overutilize any processor, i.e., we always maintain $\sigma_p \le s_p$.

Property 4. *A task has non-zero shares only on processors that have a speed at least its utilization.*

Proof. This property clearly holds for fixed tasks. We show that it holds for migrating tasks as well by contradiction.

Suppose there is some migrating task that violates this property. Let τ_a be the first migrating task to do so, and let s_q be the first processor such that $s_q < u_a$ where τ_a is assigned a non-zero share. In Alg. 1, we do not assign a migrating task a non-zero share on a slower processor unless the capacity of every faster one has been exhausted. By the definition of τ_a, Prop. 4 holds for all previously assigned migrating tasks and hence *all* previously assigned tasks, since it trivially holds for any fixed task. Moreover, because tasks are considered in non-increasing-utilization order, every such prior task has a utilization at least u_a and therefore larger than s_q. These facts imply that all prior tasks have been assigned shares only on the first $q-1$ processors, and including τ_a, the total allocated shares of the first a tasks exceeds the capacity of the first $q-1$ processors. Thus, we have

$$\sum_{i=1}^{a} u_i > S_{q-1}. \tag{6}$$

Since the processors are indexed in non-increasing-speed order,

$$S_{q-1} \ge \sum_{s_p > s_q} s_p. \tag{7}$$

Since $u_a > s_q$ and the tasks are indexed in non-increasing-utilization order, $u_i \ge u_a > s_q$ holds for all $i \le a$. That is,

$$\sum_{u_i > s_q} u_i \ge \sum_{i=1}^{a} u_i. \tag{8}$$

By (6), (7), and (8), we have

$$\sum_{u_i > s_q} u_i > \sum_{s_p > s_q} s_p,$$

which contradicts (5). Thus, no such τ_a exists and therefore Prop. 4 holds. $\square$

Property 5. *When we assign a migrating task a non-zero share on a processor, there must be at least one fixed task on that processor.*

Proof. Suppose this property is violated for the first time when migrating task τ_i is assigned a non-zero share on processor s_p, i.e., there is no fixed task on s_p. Since τ_i is the first migrating task that violates Prop. 5, no other migrating task is assigned a non-zero share on s_p either. Because no prior task (fixed or migrating) is assigned a non-zero share on s_p and $s_p \ge u_i$ (by Prop. 4), τ_i would be assigned as fixed on s_p, which contradicts our assumption that it is a migrating task. Thus, no such τ_i exists and hence this property holds. $\square$

Property 6. *If there are exactly two migrating tasks on s_p, i.e., $\psi_{h,p} > 0$ and $\psi_{l,p} > 0$, then $\psi_{h,p} + u_l < s_p$.*

Proof. By Prop. 5, there must be at least one fixed task τ_a that was assigned to s_p before the two migrating ones are assigned shares on s_p. Since the tasks are considered in non-increasing-utilization order, we have $u_l \le u_a$. Also, by the definition of σ_p^f, $\sigma_p^f \ge u_a$, so $u_l \le \sigma_p^f$. Therefore, $u_l + \psi_{h,p} + \psi_{l,p} \le \sigma_p^f + \psi_{h,p} + \psi_{l,p} \overset{\{\text{by Prop. 3}\}}{\le} s_p$. Because $\psi_{l,p} > 0$ here, we get $\psi_{h,p} + u_l < s_p$. $\square$

In the execution phase, every fixed task will only release jobs on the processor to which it assigned, whereas jobs of migrating tasks will be assigned in accordance with Prop. 1. Therefore, the following property holds as well.

Property 7. *For any k consecutive jobs of a migrating task τ_i, at most $f_{i,p} \cdot k + 2$ of them are assigned to processor s_p.*

Proof. In the first z jobs of task τ_i, let $\Gamma_{i,p}(z)$ be the number of jobs assigned to processor s_p. By Prop. 1, $\lfloor f_{i,p} \cdot z \rfloor \le \Gamma_{i,p}(z) \le \lceil f_{i,p} \cdot z \rceil$. For any k consecutive jobs of task τ_i, $\tau_{i,j}$ through $\tau_{i,j+k-1}$, the number of jobs assigned to processor s_p is

$$\begin{aligned}
&\Gamma_{i,p}(j+k-1) - \Gamma_{i,p}(j-1)\\
\le\ &\{\text{since by Prop. 1, } \lfloor f_{i,p} \cdot z \rfloor \le \Gamma_{i,p}(z) \le \lceil f_{i,p} \cdot z \rceil\}\\
&\lceil f_{i,p} \cdot (j+k-1) \rceil - \lfloor f_{i,p} \cdot (j-1) \rfloor\\
<\ &\{\text{since } \lceil x \rceil < x+1 \text{ and } \lfloor x \rfloor > x-1\}\\
&(f_{i,p} \cdot (j+k-1) + 1) - (f_{i,p} \cdot (j-1) - 1)\\
=\ &\{\text{simplifying}\}\\
&f_{i,p} \cdot k + 2.
\end{aligned}$$

$\square$

4. TARDINESS BOUNDS

In this section, we prove tardiness bounds for EDF-sh. We consider migrating tasks and fixed task separately in Secs. 4.1 and 4.2. Moreover, for migrating tasks, rather than tardiness, we upper bound lateness for each task.

4.1 Migrating Tasks

In this subsection, we derive lateness bounds for migrating tasks. Since migrating tasks are statically prioritized over fixed ones, we can ignore all fixed tasks when considering migrating ones.

Lemma 1. *Let $t_0 \ge 0$ and $t_c > t_0$. If the lateness of jobs of task τ_i is upper bounded by Δ_i, then in the time interval $[t_0, t_c)$, the demand from τ_i on processor s_p is less than*

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

PROOF. Because lateness is upper bounded by Δ_i, the jobs of τ_i released before $t_0 - (T_i + \Delta_i)$ complete their execution by t_0. Therefore, in the time interval $[t_0, t_c)$, the demand from τ_i can only come from its jobs released in $[t_0 - T_i - \Delta_i, t_c)$. τ_i can release at most $\lceil \frac{t_c-(t_0-T_i-\Delta_i)}{T_i} \rceil$ jobs in $[t_0-T_i-\Delta_i, t_c)$. By Prop. 7, at most $f_{i,p} \cdot \lceil \frac{t_c-(t_0-T_i-\Delta_i)}{T_i} \rceil + 2$ of them are assigned to processor s_p. Thus, in the time interval $[t_0, t_c)$, the demand from τ_i on processor s_p is at most

$$\left(f_{i,p} \cdot \left\lceil \frac{t_c - (t_0 - T_i - \Delta_i)}{T_i} \right\rceil + 2\right) \cdot C_i$$

$$< \{\text{since } \lceil x \rceil < x + 1\}$$
$$\left(f_{i,p} \cdot \left(\frac{t_c - (t_0 - T_i - \Delta_i)}{T_i} + 1\right) + 2\right) \cdot C_i$$

$$= \{\text{simplifying}\}$$
$$\left(f_{i,p} \cdot \frac{t_c - t_0 + 2T_i + \Delta_i}{T_i} + 2\right) \cdot C_i$$

$$= \{\text{by (1) and (2)}\}$$
$$\psi_{i,p} \cdot (t_c - t_0) + \psi_{i,p} \cdot (2T_i + \Delta_i) + 2C_i.$$

□

According to the following lemma, if we assume that all the jobs of a migrating task are moved to its last processor, then the lateness of these jobs under this assumption upper bounds their lateness in the actual schedule. This analysis device was first used by Anderson et al. [4], but assuming such jobs are moved to the task's *first* processor. We must instead consider the *last* processor, because in our case processors may have different speeds. Thus, we must conservatively assume such moves are with respect to a task's slowest processor.

LEMMA 2. *If we execute all jobs of a migrating task on its last processor rather than the processors where these jobs are actually assigned, then no job of this task will complete its execution earlier. Moreover such job moves do not impact the other migrating task on this processor (if one exits).*

PROOF. A migrating task has the highest priority on any processor that is not its last processor. Also, on any non-last processor, a migrating task is executed at a speed at least that of its last processor, since the processors are indexed from fastest to slowest. Thus, the first part of Lemma 2 holds.

The second part of Lemma 2 follows because on its last processor, a migrating task has statically lower priority than any other migrating task. □

Prop. 4 and Prop. 6 ensure that, with respect to migrating tasks, such job moves will not overutilize the last processor of the considered task.

We now compute a lateness bound for each migrating task assuming its jobs are moved as described above. If a task is the only migrating task on its last processor, then its lateness bound can be computed directly; if a migrating task τ_l shares its last processor with another migrating task τ_h, then its lateness bound depends on the lateness bound of τ_h, which can be computed inductively by the formula in Theorem 1. Lemma 3 below ensures that the base case exists.

LEMMA 3. *The migrating task with the largest index does not share its last processor with any other migrating task.*

PROOF. Follows directly from Alg. 1. □

THEOREM 1. *Consider a migrating task τ_l that has s_p as its last processor. If it shares s_p with some other migrating task τ_h, then τ_h is the only such task (by Prop. 2). Let Δ_h be the lateness bound of τ_h. Then τ_l has lateness at most*

$$\Delta_l \stackrel{def}{=} \begin{cases} \frac{\psi_{h,p}\cdot(2T_h+\Delta_h)+2C_h+C_l}{s_p-\psi_{h,p}} - T_l & \text{if } \tau_h \text{ exists,} \\ \frac{C_l}{s_p} - T_l & \text{otherwise.} \end{cases} \tag{9}$$

PROOF. By Lemma 2, we can upper bound the lateness of all jobs of τ_l by assuming that all such jobs execute on s_p. We make that assumption here.

If τ_l does not share its last processor s_p with any other migrating task, i.e., τ_h does not exist, then τ_l has the highest priority on s_p, and by Prop. 4, $s_p \geq u_l$. Therefore, every job of τ_l completes its execution within $\frac{C_l}{s_p}$ time units of its release. Thus, lateness is upper bounded by $\frac{C_l}{s_p} - T_l$.

In the remainder of the proof, we consider the case where τ_h does exist. In this case, we prove Theorem 1 by contradiction.

Interval $[t_0, t_c)$**.** Let $\tau_{l,j}$ be the first job of τ_l that has lateness exceeding Δ_l and define $t_c \stackrel{\text{def}}{=} d_{l,j} + \Delta_l$. Let t_0 be the latest time instant before t_c such that s_p is idle for migrating tasks, i.e., all jobs of τ_l or τ_h released before t_0 have completed execution by t_0 and a job of τ_l and/or τ_h is released at t_0. t_0 is well defined because if no such time instant exists within $(0, t_c)$, then time 0 must be such a time instant.

Demand from τ_h**.** By Lemma 1, in the time interval $[t_0, t_c)$, the demand from τ_h on s_p is less than

$$\psi_{h,p} \cdot (t_c - t_0) + \psi_{h,p} \cdot (2T_h + \Delta_h) + 2C_h. \tag{10}$$

Demand from τ_l**.** The demand from τ_l on s_p comes from jobs of τ_l released before $r_{l,j}$ and $\tau_{l,j}$ itself. By the definition of t_0, the number of such jobs released before $r_{l,j}$ is at most $\lfloor \frac{r_{l,j}-t_0}{T_l} \rfloor$. Including $\tau_{l,j}$ itself, at most $\lfloor \frac{r_{l,j}-t_0}{T_l} \rfloor + 1$ jobs of τ_l create demand in the interval. Thus, in the time interval $[t_0, t_c)$, the demand due to τ_l on processor s_p is at most

$$\begin{aligned} &\left(\left\lfloor \frac{r_{l,j} - t_0}{T_l} \right\rfloor + 1\right) C_l \\ \leq &\left(\frac{r_{l,j} - t_0}{T_l} + 1\right) C_l \\ = &\{\text{by (1)}\} \\ &u_l(r_{l,j} - t_0) + C_l. \end{aligned} \tag{11}$$

For the purpose of minimizing redundancy in expressions, we define

$$K \stackrel{\text{def}}{=} \psi_{h,p} \cdot (2T_h + \Delta_h) + 2C_h + C_l. \tag{12}$$

Using this definition, by (10), (11), and (12), the total demand within $[t_0, t_c)$ due to migrating tasks is less than

$$K + \psi_{h,p} \cdot (t_c - t_0) + u_l(r_{l,j} - t_0)$$

$$= \{\text{rearranging}\}$$
$$K + \psi_{h,p} \cdot (t_c - r_{l,j}) + (\psi_{h,p} + u_l)(r_{l,j} - t_0)$$

$$< \{\text{by Prop. 6}\}$$
$$K + \psi_{h,p} \cdot (t_c - r_{l,j}) + s_p(r_{l,j} - t_0).$$

Since for contradiction, we assumed that $\tau_{l,j}$ has lateness exceeding Δ_l, i.e., $\tau_{l,j}$ completes execution after t_c, the total demand in the time interval $[t_0, t_c)$ is greater than the total supply in this interval, which is $s_p(t_c - t_0)$. This implies

$$K + \psi_{h,p} \cdot (t_c - r_{l,j}) + s_p(r_{l,j} - t_0) > s_p(t_c - t_0). \quad (13)$$

By simplifying (13), we have

$$K > (t_c - r_{l,j})(s_p - \psi_{h,p}). \quad (14)$$

By Prop. 6, we have $\psi_{h,p} + u_l < s_p$. Because $u_l > 0$, this implies $\psi_{h,p} < s_p$, i.e.,

$$s_p - \psi_{h,p} > 0. \quad (15)$$

By (14) and (15),

$$t_c - r_{l,j} < \frac{K}{s_p - \psi_{h,p}}. \quad (16)$$

Replacing the right-hand side of (16) by the definition of K in (12) and the definition of Δ_l in (9), we have

$$t_c - r_{l,j} < \Delta_l + T_l. \quad (17)$$

Since $T_l = d_{l,j} - r_{l,j}$, (17) implies $t_c < d_{l,j} + \Delta_l$, which contradicts the definition of t_c Thus, such an assumed job $\tau_{l,j}$ with lateness exceeding Δ_l does not exist. Hence, Theorem 1 holds. $\square$

4.2 Fixed Tasks

In this subsection, instead of lateness, we consider tardiness directly.

To begin with, note that if no migrating task assigns jobs to a processor, then all of the fixed tasks on that processor have a tardiness bound of zero, since EDF is optimal for uniprocessor scheduling and by Prop. 3 we do not overutilize any single processor.

Theorem 2 below provides a tardiness bound for a fixed task that executes on a processor where migrating task(s) also execute. In this case, the tardiness bound for the fixed task depends on the lateness bound(s) for the migrating task(s) on the same processor, which can be computed by Theorem 1. By Prop. 2, at most two migrating tasks have non-zero shares on a processor.

THEOREM 2. *Suppose that one or two migrating tasks have non-zero shares on processor s_p. If two, let τ_l (τ_h) be the one with lower (higher) priority; if only one, let τ_l denote that task and consider τ_h to be a "null" task with $C_h = 0$, $\psi_{h,p} = 0$, and $T_h = 1$. Then, a fixed task τ_i on s_p has tardiness at most*

$$\Delta_i \stackrel{\text{def}}{=} \frac{\psi_{l,p} \cdot (2T_l + \Delta_l) + 2C_l + \psi_{h,p} \cdot (2T_h + \Delta_h) + 2C_h}{s_p - \psi_{l,p} - \psi_{h,p}}. \quad (18)$$

PROOF. This proof is similar to that of Theorem 1.

Interval $[t_0, t_c)$. Let $\tau_{i,j}$ be the first job of any fixed task on s_p that has tardiness exceeding the bound in (18) and define $t_c \stackrel{\text{def}}{=} d_{i,j} + \Delta_i$. Let t_0 be the latest idle time instant before t_c, i.e., at time instant t_0, the processor s_p is either idle or executing some job with a priority lower than $\tau_{i,j}$'s priority and at least one job with a priority at least $\tau_{i,j}$'s priority is released. t_0 is well-defined because if no such time instant exists within $(0, t_c)$, then time 0 must be a such time instant.

Demand from mirgrating tasks. By Lemma 1, in $[t_0, t_c)$, the demand from τ_l on s_p is less than

$$\psi_{l,p} \cdot (t_c - t_0) + \psi_{l,p} \cdot (2T_l + \Delta_l) + 2C_l, \quad (19)$$

and the demand from τ_h on s_p is less than

$$\psi_{h,p} \cdot (t_c - t_0) + \psi_{h,p} \cdot (2T_h + \Delta_h) + 2C_h. \quad (20)$$

Demand from fixed tasks. A fixed task τ_k can release at most $\lfloor \frac{d_{i,j} - t_0}{T_k} \rfloor$ jobs with a priority at least $\tau_{i,j}$'s priority in the interval $[t_0, t_c)$. Thus, the demand from fixed tasks in $[t_0, t_c)$ is at most

$$\begin{aligned}
&\sum_{\tau_k \in \tau_p^f} \left\lfloor \frac{d_{i,j} - t_0}{T_k} \right\rfloor \cdot C_k \\
\leq &(d_{i,j} - t_0) \cdot \sum_{\tau_k \in \tau_p^f} \frac{C_k}{T_k} \\
= &\{\text{by the definition of } \sigma_p^f\} \\
&(d_{i,j} - t_0) \cdot \sigma_p^f \\
\leq &\{\text{by Prop. 3}\} \\
&(d_{i,j} - t_0)(s_p - \psi_{l,p} - \psi_{h,p}).
\end{aligned} \quad (21)$$

For the purpose of minimizing redundancy in expressions, we define

$$K \stackrel{\text{def}}{=} \psi_{l,p} \cdot (2T_l + \Delta_l) + 2C_l + \psi_{h,p} \cdot (2T_h + \Delta_h) + 2C_h. \quad (22)$$

Using this definition, by (19), (20), (21), and (22), the total demand within $[t_0, t_c)$ is at most

$$K + (\psi_{l,p} + \psi_{h,p})(t_c - t_0) + (s_p - \psi_{l,p} - \psi_{h,p})(d_{i,j} - t_0).$$

Since for the purpose of contradiction, we assume $\tau_{i,j}$ has tardiness exceeding Δ_i, i.e., $\tau_{i,j}$ completes execution after t_c, the total demand in the time interval $[t_0, t_c)$ is greater than the total supply in the interval which is $s_p(t_c - t_0)$. That is,

$$\begin{aligned}
&K + (\psi_{l,p} + \psi_{h,p})(t_c - t_0) \\
&\quad + (s_p - \psi_{l,p} - \psi_{h,p})(d_{i,j} - t_0) > s_p(t_c - t_0).
\end{aligned} \quad (23)$$

By simplifying (23), we have

$$K > (t_c - d_{i,j})(s_p - \psi_{l,p} - \psi_{h,p}). \quad (24)$$

By Prop. 3, we have $\sigma_p^f + \psi_{h,p} + \psi_{l,p} \leq s_p$. Because $\sigma_p^f > 0$, this implies

$$s_p - \psi_{l,p} - \psi_{h,p} \geq \sigma_p^f > 0. \quad (25)$$

By (24) and (25),

$$t_c - d_{i,j} < \frac{K}{s_p - \psi_{l,p} - \psi_{h,p}}. \quad (26)$$

Replacing the right-hand side of (26) by the definition of K in (22) and the definition of Δ_i in (18), we have

$$t_c - d_{i,j} < \Delta_i. \quad (27)$$

(27) implies $t_c < d_{i,j} + \Delta_i$, which contradicts the definition of t_c. Thus, such an assumed job $\tau_{i,j}$ with tardiness exceeding Δ_i does not exist. Hence, Theorem 2 holds. $\square$

5. EVALUATION

To evaluate how restrictive the assumed per-task utilization constraint is and the effectiveness of EDF-sh, we conducted experiments to assess schedulability and tardiness bounds for EDF-sh.

When conducting such experiments for identical multiprocessors, the assumed platform is implicitly determined by an assumed total processor capacity, or the number of processors. However, for uniform heterogeneous multiprocessors, processor speeds must be defined. Given a total processor capacity, there are a infinite number of speed choices from which to select. Because only selected choices can be considered, no evaluation can be exhaustive. In our experiments, we considered systems of eight processors with a total processor capacity of 36. We considered four such platforms, with speeds as follows: $\pi_1 = \{6, 6, 6, 6, 3, 3, 3, 3\}$, $\pi_2 = \{8, 8, 4, 4, 4, 4, 2, 2\}$, $\pi_3 = \{8, 7, 6, 5, 4, 3, 2, 1\}$, $\pi_4 = \{15, 3, 3, 3, 3, 3, 3, 3\}$.

The process of randomly generating feasible task systems for the considered platforms also varies from that for identical ones. In the identical case, the per-task utilization bound is fixed for every task to be 1.0. However, per-task utilization bounds in the heterogeneous case must instead follow (4). As such, before generating a new task, we calculated a per-task utilization cap for it by (4), considering previously generated tasks. We then selected the utilization of that task uniformly at random between zero and the computed cap. This generation process terminates when the total utilization of all generated tasks exceeds or equals a pre-set total utilization limit. The utilization of the last generated task is then adjusted so that the total generated utilization matches the pre-set limit.

We require the number of tasks n to be at least the number of processors m. To ensure this, whenever $n < m$ held for a generated system, a task was chosen at random and replaced by two tasks with half the utilization of the original one (this process was repeated as necessary). Given a platform and a total task system utilization, having fewer (more) tasks means having higher (lower) expected per-task utilizations. To reflect these two extremes, we defined the minimum number of the tasks to be either eight (fewer but heavier tasks) or 32 (more but lighter tasks) for every considered platform. Also, we selected each task's execution requirement uniformly from $[5, 25]$ and calculated its period from its utilization and execution requirement. In all experiments in this section, we varied total utilization within $[0, 36]$ by increments of 0.5, and for each total utilization, we generated 10,000 feasible task sets.

Schedulability. EDF-sh has the same utilization restrictions (i.e., (3) and (5)) as EDF-ms, but EDF-sh can support platforms in which speed groups exist with only one processor, while EDF-ms requires each such group to have at least two processors. For this reason, EDF-sh dominates EDF-ms in terms of SRT schedulability.

Given this provable dominance over EDF-ms, our assessment of schedulability under EDF-sh focuses on determining the fraction of randomly generated feasible systems (as defined by (3) and (4)) it can successfully schedule for every given total utilization and every platform. Fig. 4 shows the results of these experiments. More than 87% of the generated systems were SRT-schedulable under EDF-sh. In general, the smaller the difference among processor speeds, the better the schedulability. This makes sense, since for identical multiprocessors, EDF-sh is SRT-optimal, like EDF-os. Furthermore, when there are many lighter tasks instead of few heavier ones, schedulability is quite close to optimal.

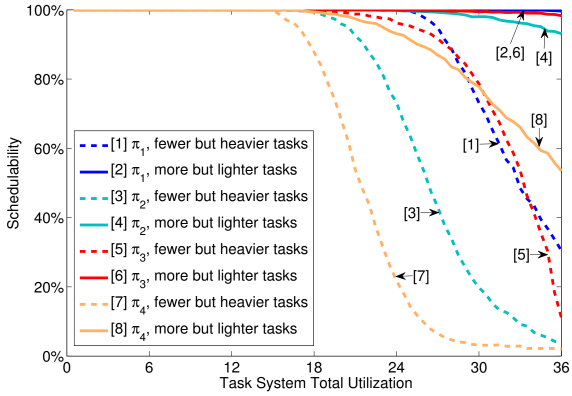


Figure 4: Schedulability under EDF-sh.

Tardiness Bounds. We also compared tardiness bounds under EDF-sh to those under EDF-ms. Since EDF-ms requires each speed group to have at least two processors, i.e., it does not apply to π_3 and π_4, we only computed tardiness bounds for π_1 and π_2. We compared EDF-sh and EDF-ms in terms of both maximum absolute tardiness bounds and maximum relative tardiness bounds, where the latter is defined as the ratio of a task's tardiness bound to its period. Fig. 5(a) shows absolute tardiness bounds and Fig. 5(b) shows relative tardiness bounds.

In most cases, EDF-sh exhibits significantly lower maximum tardiness bounds than EDF-ms. The only exception to this is when total utilization is close to overutilizing the platform, and even then, EDF-sh is never substantially worse. Furthermore, note that in EDF-sh, migrations are boundary-limited. If we were to take overheads into account, EDF-sh would likely outperform EDF-ms in all cases. We defer an overhead-aware comparison to future work.

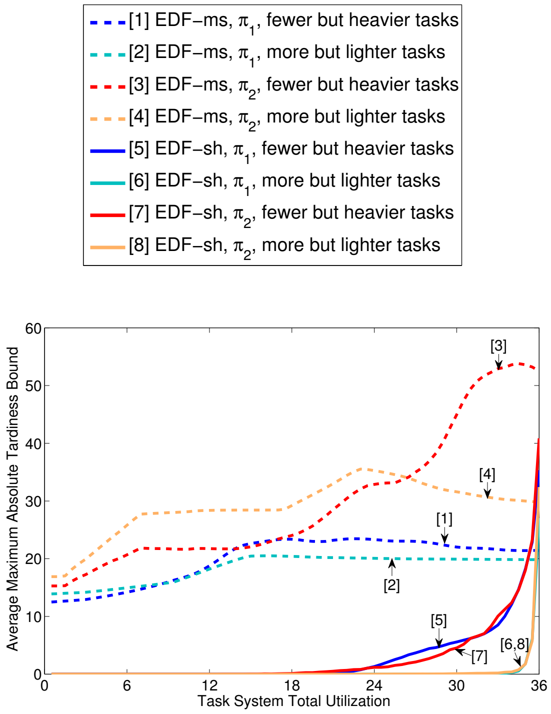
6. CONCLUSION

We have presented EDF-sh, an EDF-based semi-partitioned scheduling algorithm for SRT uniform heterogeneous systems. To the best of our knowledge, this is the first restricted-migration scheduling algorithm for SRT heterogeneous systems.

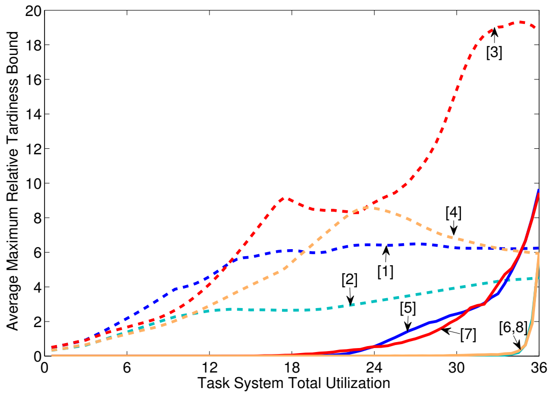
EDF-sh is an extension of EDF-os, which is a SRT-optimal semi-partitioned scheduling algorithm for identical multiprocessors. Hence, EDF-sh inherits the boundary-limited property from EDF-os. However, the optimality of EDF-os has not been retained, due to the introduction of (5). On the other hand, EDF-sh, like EDF-os, allows total task system utilization to be as high as the total processor capacity.

We also presented an experimental evaluation that shows that EDF-sh has good performance in terms of both schedulability and tardiness bounds. Furthermore, EDF-sh dominates the previously proposed EDF-ms in terms of schedulability, and in our experiments, almost always exhibited significantly lower tardiness bounds than EDF-ms.

Future work. The development of SRT-optimal global and semi-partitioned scheduling algorithms for uniform heterogeneous multiprocessors remains as an open problem.



(a) abosolute tardiness bounds



(b) relative tardiness bounds

Figure 5: Tardiness bounds of EDF-ms and EDF-sh.

Overhead-aware scheduler comparisons based on actual implementations remain as future work as well. Processors are available today that can run at degraded speeds to save energy or reduce temperature. An identical multiprocessor system that consists of such processors can potentially become a uniform heterogeneous system if each processor can degrade speed independently. Such systems warrant further attention as well.

References

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

[2] Samsung's Exynos 5422 & The Ideal big.LITTLE: Exynos 5 Hexa (5260). http://www.anandtech.com/show/7811/samsungs-exynos-5422-the-ideal-biglittle-exynos-5-hexa-5260.

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

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

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

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

[7] S. Baruah, S. Funk, and J. Goossens. Robustness results concerning EDF scheduling upon uniform multiprocessors. *IEEE Trans. on Computers*, 52(9):1185-1195, 2003.

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

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

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

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

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

[13] U. Devi and J. Anderson. Tardiness bounds under global EDF sched-uling on a multiprocessor. *Real-Time Sys.*, 38(2):133-189, 2008.

[14] 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.

[15] J. Erickson, U. Devi, and S. Baruah. Improved tardiness bounds for global EDF. In *22nd ECRTS*, 2010.

[16] J. Erickson and J. Anderson. Fair lateness scheduling: reducing maximum lateness in G-EDF-like Scheduling. In *24th ECRTS*, 2012.

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

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

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

[20] 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.

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

[22] E. Horvath, S. Lam, and R. Sethi. A level algorithm for preemptive scheduling. *Journal of the ACM*, 24(1):32-43, 1977.

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

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

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

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

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

[28] A. Mills and J. Anderson. A multiprocessor server-based scheduler for soft real-time tasks with stochastic execution demand. In *17th RTCSA*, 2011.

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

[30] M. Shekhar, A. Sarkar, H. Ramaprasad, and F. Mueller. Semipartitioned hard real-time scheduling under locked cache migration in multicore systems. In *24th ECRTS*, 2012.

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