Tardiness Bounds for EDF Scheduling on Multi-Speed Multicore Platforms*

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

Abstract

Multicore platforms, which include several processing cores on a single chip, are being widely touted as a solution to heat and energy problems that are impediments to single-core chip designs. To accommodate both parallelizable and inherently-sequential applications on the same platform, heterogeneous multicore designs with faster and slower cores have been proposed. In this paper, we consider the problem of scheduling *soft* real-time workloads on such a platform.

1 Introduction

Given the thermal and power problems that plague single-processor chip designs, most major chip manufacturers are investing in multicore technologies to achieve higher system performance. A number of systems with a modest number of cores are currently available, and in the coming years, the number of cores per chip is expected to increase significantly. In fact, chips with hundreds of cores are envisioned. It remains to be seen, however, whether such an extensive degree of parallelism can be effectively exploited. Indeed, many applications exist that are *inherently* sequential. In light of this, one approach, which is being advocated by many chip designers, is to provide a mix of faster and slower cores on the same platform [7]. On such a platform, an inherently-sequential application would benefit by executing on a faster core, while parallizable applications could execute across many slower cores. While having only fast cores would obviously be desirable, faster cores require greater chip area, so such an approach would adversely limit the number of cores per chip.

Large multicore platforms will likely be used in settings where timing constraints are required. For example, one envisioned use of such platforms is as a central server within the home that multiplexes interactive applications that require best-effort service with multimedia applications that have real-time requirements [3]. Such applications might include streaming movies over the Internet, providing cable television, or executing custom programs such as video games. Timing constraints in these applications are typically *soft*: missed deadlines, though undesirable, are usually not disastrous. Such constraints are far more common than hard constraints in many settings [8]. Unfortunately, prior work on scheduling real-time workloads on heterogeneous multiprocessors has focused only on hard real-time systems. While such work can be applied to schedule soft real-time applications, this comes at the price of overly conservative system designs.

In this paper, we show that such conservatism can be eliminated if deadline misses are permissible. We show this by presenting an algorithm for multi-speed systems called EDF-ms, which is a variant of the global EDF (GEDF) scheduling algorithm. Like GEDF [4], deadline tardiness under EDF-ms is bounded when scheduling sporadic tasks. Further, such bounds do not require severe caps on total utilization. In contrast, even with same-speed cores, if deadlines cannot be missed, then caps that can approach 50% of the available

*Work supported by a grant from Intel Corp., by NSF grants CNS 0408996, CCF 0541056, and CNS 0615197 and by ARO grant W911NF-06-1-0425.

processing capacity are required under all known scheduling algorithms, except for Pfair algorithms [2]. Pfair algorithms, which have not been studied in the context of multi-speed systems, schedule tasks one quautum at a time, and thus preempt and migrate tasks frequently. EDF-ms preempts and migrates tasks less frequently, does not require same-speed cores, and can accommodate tasks with high execution costs for which utilization exceeds one on slower cores.

Prior work. Work on scheduling in heterogeneous multiprocessor real-time systems was initiated by Funk and colleagues, who presented a number of scheduling algorithms and associated analysis methods for systems with hard timing constraints. References concerning this work can be found in Funk's Ph.D. dissertation [6] (and are not included here due to space constraints). As noted earlier, our emphasis on *soft* real-time systems distinguishes our work from these earlier efforts.

In work that is more experimental in nature, Kumar *et al*. [7] measured throughput and job response times on a two-speed multicore system with partitioned scheduling, and presented dynamic load-balancing heuristics that maximize throughput. This work is of relevance to research on soft real-time systems, as job response times are considered, but it does not include any analysis for validating deadlines or deadline tardiness.

Our algorithm, EDF-ms, has been devised by utilizing ideas from two prior papers concerning symmetric multiprocessor systems by Devi *et al.* [1, 5]. In [1], an EDF-based algorithm called EDF-fm is presented that limits task migrations without restrictive caps on total utilization. EDF-fm is a hybrid of partitioned EDF and GEDF. In EDF-fm, tasks are categorized as either "fixed" or "migrating" (hence the suffix "fm"). A *fixed* task exclusively executes on a specific processor. On the other hand, each *migrating* task executes on two processors, with each of its invocations executing exclusively on one of its assigned processors. Individual task utilizations must be capped at 1/2, but total utilization is not restricted (other than being at most the system's total processing capacity). Invocations of each migrating task are distributed between its assigned processors so that these processors are not overloaded in the long run; however, short-term overloads are possible. Such overloads can cause fixed tasks to miss their deadlines. However, such misses are by bounded amounts only.

Because EDF-ms is a variant of GEDF, tardiness bounds established for GEDF are of relevance to our work. Such bounds were first established by Devi and Anderson in [4]. These bounds apply to any sporadic task system with total utilization at most M scheduled on M symmetric processors. Any task can have maximum tardiness. Such behavior might not be acceptable for certain applications. In [5], the analysis in [4] is extended to allow up to M "privileged" tasks to have any predefined tardiness value, including zero. The resulting variant of GEDF is called EDF-hl. (The suffix "hl" signifies that privileged tasks are given higher priority, and others lower.)

Our approach. In the problem considered herein, cores are organized into groups, where cores in the same group have the same speed. In the earlier EDF-fm algorithm, migrating tasks are prioritized over fixed tasks to ensure that the former have zero tardiness. This allows schedulability to be analyzed on each processor independently. (If a migrating task were to miss a deadline on one of its processors, then this might delay its next invocation on its other processor. As a result, the two processors could not be analyzed independently.) We desire to maintain a similar independence property across *groups* of cores. To do this, we categorize tasks as either "fixed" or "intergroup." A *fixed* task executes only on the cores in one group, while an *intergroup* task may be executed on two groups of cores. We use the term "intergroup" instead of "migrating" because a fixed task in our case may migrate (among the cores in its group). We distribute the invocations of an intergroup task between its two assigned core groups in the same way as invocations of migrating tasks are distributed in EDF-fm. Further, we treat intergroup tasks specially when scheduling, as in the earlier EDF-hl algorithm, so that they can be guaranteed zero tardiness. This enables each group of cores to be analyzed independently. However, one key difference arises in our analysis: the distribution pattern used for intergroup tasks allows short-term overloads to occur (for the same reason that such overloads occur in EDF-fm). Thus, the analysis of tardiness in [5] must be adjusted to allow privileged tasks to create short-term overloads.

Summary of contributions. The main contributions of this paper include devising EDF-ms and establishing tardiness bounds for it. In addition, we present an experimental evaluation of EDF-ms's effectiveness in limiting tardiness. To our knowledge, EDF-ms is the first algorithm proposed for multi-speed platforms that can schedule soft real-time tasks with bounded tardiness without severe utilization restrictions. Because the ideas underlying EDF-ms were originally proposed in completely different settings, new analysis for integrating these ideas had to be devised. EDF-ms, its analysis, and evaluation are presented in detail in Secs. 3–5, after first presenting our system model in Sec. 2.

2 System Model

We consider the problem of scheduling a set of sporadic tasks on $M \geq 4$ cores of $g \geq 2$ speeds. We will group cores by speed: we let m_h denote the number of cores in Group h, where $1 \leq h \leq g$, and we let s_h denote their speed. We assume $s_1 = 1$ and $s_j < s_k$ if $j < k$. We also assume that $m_h \geq 2$ holds for each h. (Given that our main focus is large multicore platforms, this is a reasonable assumption. However, we briefly consider later how to handle groups with only one core.)

We let τ denote the sporadic task system to be scheduled, and assume that it consists of n independent tasks, $T_1, \ldots, T_n$. Each task is invoked or *released* repeatedly, with each such invocation called a *job*. Associated with each task T_i are two parameters, e_i and p_i: e_i gives the maximum *execution time* of one job of T_i on a unit-speed core, while, p_i, called the *period* of T_i, gives the minimum time between two consecutive job releases of T_i. On a core with speed s_h, a job of T_i completes in e_i/s_h time units. For brevity, T_i's parameters are sometimes denoted using the notation $T_i = (e_i, p_i)$.

The k^{th} job of T_i, where $k \geq 1$, is denoted $T_{i,k}$. A task's first job may be released at any time at or after time zero. The release time of the job $T_{i,k}$ is denoted $r_{i,k}$ and its (absolute) deadline $d_{i,k}$ is defined as $r_{i,k} + p_i$. Each task is sequential, so at any time, it may execute on at most one core. When a job of a task misses its deadline, the release time of the next job of that task is unaltered. This ensures that each task receives a processor share in accordance with its utilization (defined below) in the long term. Thus, a task may release a new job when prior job(s) of that task have not been completed. Such a new job cannot commence execution until the prior jobs have completed. If a job $T_{i,j}$ with a deadline at $d_{i,j}$ completes at time t, then its *tardiness* is defined as $max(0, t - d_{i,j})$. A *task*'s tardiness is the maximum of the tardiness of any of its jobs.

The *utilization of task* T_i is defined as $u_i = e_i/p_i$, and the *utilization of the task system* τ as $U_{sum} = \sum_{T_i \in \tau} u_i$. We require $\sum_{u_i > s_j} u_i \leq \sum_{k>j} m_k \cdot s_k$ and $U_{sum} \leq \sum_{h=1}^{g} m_h \cdot s_h$. Otherwise, tardiness can grow unboundedly. Note that the first of these requirements implies $u_i \leq max(s_j)$. Note also that it is possible that $u_i > s_j$ holds for some j.

In this paper, we assume that time is continuous, but execution costs, periods, and core speeds are rational.

3 Algorithm EDF-ms

The name EDF-ms stands for EDF *multi-speed*. Like EDF-fm [1], mentioned earlier, the algorithm consists of two phases: an offline *task assignment* phase and an online *execution* phase. When the task assignment phase is applied to some task set τ, at most g groups of tasks are created. In addition, there may be up to $g - 1$ tasks that do not belong to any group. Each of these tasks may execute on the cores of two groups. For each pair of consecutive groups, say Group h and Group $h + 1$, at most one task that migrates between them may exist. We denote this task (if it exists) as $T^{h,h+1}$ and call it an *intergroup* task. All other tasks are called *fixed* tasks, as each executes within one group only. The assignment algorithm (which is not shown, due to space constraints) sorts tasks by utilization and assigns tasks to groups by exhausting the capacity of faster groups first.

The assignment algorithm returns a set of values $Z_{k,h}$, where $1 \leq k \leq n$ and $1 \leq h \leq g$. The value $Z_{k,h}$ denotes the fraction of T_k's utilization that is assigned to Group h. For any fixed task T_k assigned to Group h, $Z_{k,h} = u_k$ holds. For any intergroup task $T_k = T^{h,h+1}$, $Z_{k,h} + Z_{k,h+1} = u_k$ holds. If task T_k may not execute on Group h, then $Z_{k,h} = 0$. The assignment algorithm ensures the following.

$$\sum_h Z_{i,h} = u_i \quad \wedge \quad \sum_i Z_{i,h} \leq m_h \cdot s_h \tag{1}$$

Example 1. Fig. 1 shows the assignment determined for the task set $\tau = \{T_1, \ldots, T_{13}\}$. For conciseness, we will use the notation T_{i-j} to denote $T_i, \ldots, T_j$ in describing this task set. τ consists of $T_{1-5} = (8, 10)$, $T_{6-9} = (3, 2)$, and $T_{10-13} = (4, 2)$. Tasks T_{1-5}, T_{6-9}, and T_{10-13} have utilization 0.8, 1.5, and 2.0, respectively. The system is comprised of three groups of cores of speeds one, two, and three. That is, $m_1 = 3$, $s_1 = 1$, $m_2 = 3$, $s_2 = 2$, $m_3 = 3$, and $s_3 = 3$. In Fig. 1, each core group is depicted as a "bin" with its height proportional to $m_h \cdot s_h$, *i.e.*, the total processing capacity of the group. As seen, tasks T_{1-3} are assigned to Group 1, tasks T_{5-8} to Group 2, and tasks T_{10-13} to Group 3. Tasks $T_4 = T^{1,2}$ and $T_9 = T^{2,3}$ are intergroup tasks.

For each Group h, we can define a set of tasks τ_h with jobs to be scheduled by this group:

$$\tau_h = \{T_i : Z_{i,h} > 0\}. \tag{2}$$

As with migrating tasks in EDF-fm [1], if an intergroup task $T^{h,h+1}$ were to miss its deadline in the schedule for either

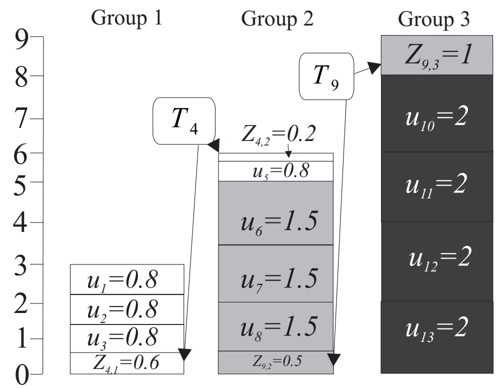


Figure 1: Mapping of the task set onto groups of cores in Example 1. Different shadings are used for tasks with utilization at most one, two, and three.

Group h or Group $h + 1$, then this would create a nontrivial linkage between these two groups that complicates scheduling analysis. This is because, if a job of $T^{h,h+1}$ misses its deadline, then the processing of the next job of $T^{h,h+1}$ may be delayed until *after* its release, and this may increase the chance that it will miss as well. Thus, missed deadlines in one group could lead to missed deadlines in another group. Thus, our scheduling policy must achieve two goals: **(i)** allow us to analyze the schedule in each group independently, and **(ii)** not overload any group in the long run (for otherwise, tardiness in such a group would grow unboundedly).

These goals are accomplished as follows. During the execution phase of EDF-ms, jobs of tasks in τ_h are scheduled on the cores in Group h using GEDF, with the jobs of intergroup tasks treated specially. As we shall see, the special treatment given to intergroup tasks ensures that their jobs always have zero tardiness. This allows us to analyze each core group as a separate (same-speed) system. The jobs of each intergroup task $T_k = T^{h,h+1}$ are distributed between its assigned Groups h and $h + 1$ using a special deterministic pattern first described in [1], which ensures that the total workload from these jobs assigned to these two groups over the long term is in accordance with the shares $Z_{k,h}$ and $Z_{k,h+1}$, respectively.

In order to describe this assignment pattern, we introduce

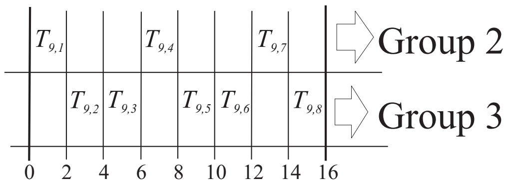


Figure 2: Assignment of jobs of the task T_9 from Example 2.

some additional notation. For each intergroup task $T_k = T^{h,h+1}$, we let $f_{k,h}$ and $f_{k,h+1}$ denote the fraction of T_k's jobs that are processed by Groups h and $h+1$, respectively. These two quantities are defined as follows.

$$f_{k,h} = Z_{k,h}/u_k \quad \wedge \quad f_{k,h+1} = Z_{k,h+1}/u_k \tag{3}$$

Note that, by (1), $f_{k,h} + f_{k,h+1} = 1$.

To explain the assignment pattern, we consider a single group, Group h. Assume that there exist two intergroup tasks $T^{h-1,h}$ and $T^{h,h+1}$. If we were to depict Group h as a "bin" as done in Fig. 1, then $T^{h-1,h}$ would be the *top* task in Group h, and $T^{h,h+1}$ would be the *bottom* task. Different (complementary) assignment rules are needed for these two cases. Let T_k denote either $T^{h-1,h}$ or $T^{h,h+1}$. Let $j \geq 1$ be the index of the latest job released by T_k at or after time t and let j_a be the number of jobs of T_k assigned to Group h before t. For the case where $T_k = T^{h-1,h}$, *i.e.*, T_k is the top task, the j^{th} job of T_k is assigned to Group h iff $j - 1 \neq \lfloor j - j_a/(1 - f_{k,h}) \rfloor$. We call this assignment rule the *top rule*. For the case where $T_k = T^{h,h+1}$, *i.e.*, T_k is the bottom task, the j^{th} job of T_k is assigned to Group h iff $j - 1 = \lfloor j_a / f_{k,h} \rfloor$. We call this assignment rule the *bottom rule*.

Example 2. Fig. 2 shows the assignment pattern for the jobs of task T_9 from Example 1, which has frequencies $f_{9,2} = Z_{9,2}/u_9 = 0.5/1.5 = 1/3$ and $f_{9,3} = Z_{9,3}/u_9 = 1/1.5 = 2/3$. Jobs of T_9 are assigned to Group 2 using the bottom rule and to Group 3 using the top rule. Here, we will focus on Group 2. Consider the time instant $t = 6$ when the fourth job of T_9 is released. When considering this fourth job in the bottom rule, $j = 4$. Prior to time t, one job of T_9 was assigned to Group 2, so $j_a = 1$. Applying these values via the bottom rule, we obtain $j - 1 = 3 = \lfloor 1 \cdot 3 \rfloor = \lfloor j_a / f_{9,2} \rfloor$, so the job is assigned to Group 2. Note that, by time 6, three jobs of T_9 are released and one of them is processed by Group 2, which is in accordance with the fraction $f_{9,2} = 1/3$.

As shown in [1], this assignment strategy ensures that the maximum number of jobs of an intergroup task T_k released during an interval of length t and assigned to Group h is at most $\lceil f_{k,h} \lceil \frac{t}{p_k} \rceil \rceil$. Thus, the maximum demand due to jobs of T_k that must be processed by Group h during an interval of length t is at most

$$\left\lceil f_{k,h} \left\lceil \frac{t}{p_k} \right\rceil \right\rceil e_k, \tag{4}$$

which is approximately $Z_{k,h} \cdot t$. Because the demand of each fixed task $T_k \in \tau_h$ during the interval $[0, t)$ is at most $u_k \cdot t$, and the demand of each intergroup task T_k assigned to Group h is approximately $Z_{k,h} \cdot t$, Group h will not be overloaded in the long run. For example, for Group 2 in Fig. 1, these values sum to $6t$, which matches the group's overall computing capacity within $[0, t)$, as given by $m_2 \cdot s_2 \cdot t$.

Because no group is overloaded in the long term, the scheduling policy we give below for each group will ensure that the jobs of intergroup tasks never miss their deadlines. As such, we no longer need to consider multiple groups, but can concentrate our analysis efforts on just one, say Group h. Furthermore, Group h's per-core speed of s_h is no longer an issue, since all cores in the group have the same speed. We therefore assume that all cores in Group h have a speed of one and that all execution costs, utilizations, and $Z_{k,h}$ values of tasks executing on the cores of Group h have been scaled by dividing them by s_h.

We further simplify the problem notationally by assuming that we have have m (unit-speed) processors upon which we must schedule a set of $n+2$ sporadic tasks, $\tau = \{T_0, T_1, \ldots, T_{n+1}\}$. T_0 and T_{n+1} represent, respectively, the top and bottom intergroup tasks for this core group. (Later, we explain how to adjust our results if either of these tasks does not exist.) $T_1, \ldots, T_n$ are the fixed tasks for the group. Our scheduling policy treats T_0 and T_{n+1} specially so that their jobs do not miss their deadlines, so we call them *privileged* tasks. Jobs of the privileged tasks are assigned to the system using the top and bottom rules discussed earlier. We let $Z_0 < u_0$ and $Z_{n+1} < u_{n+1}$ denote the part of the utilization of tasks T_0 and T_{n+1}, respectively, that must be processed by the system. If T_0 and T_{n+1} both exist, as assumed here, then $Z_0 + Z_{n+1} + \sum_{i=1}^{n} u_i = m$. More generally, $Z_0 + Z_{n+1} + \sum_{i=1}^{n} u_i \leq m$. We let $f_0 = Z_0/u_0$ and $f_{n+1} = Z_{n+1}/u_{n+1}$.

Jobs of privileged tasks are treated specially in scheduling by using an approach presented in [5]. In this approach, the concept of slack is used: if job $T_{k,j}$ executes for $\delta_{k,j}$ time prior to time $t \leq d_{k,j}$, then its *slack* at t, given by $d_{k,j} - t - (e_k - \delta_{k,j})$, represents the maximum amount of time that $T_{k,j}$ can remain idle (*i.e.*, not execute) and still meet its deadline. This concept is used in scheduling tasks in the following way: all jobs are scheduled using GEDF, with the exception that, if a job of a privileged task has zero slack, then it is executed continuously until its deadline. This policy clearly ensures that privileged tasks do not miss their deadlines. (Recall that each core group consists of at least two cores.)

4 Tardiness Bounds

In this section, we derive a tardiness bound for EDF-ms. We begin by digressing to consider a related bound, proved for EDF-hl in [5]. Several properties established there for EDF-hl hold for EDF-ms as well.

4.1 Comparison of EDF-ms and EDF-hl

EDF-hl is the same as EDF-ms except for three differences. First, there may be up to m privileged tasks in EDF-hl, instead of just two (where m is the number of processors). Second, *all* jobs of each privileged task are scheduled by the system.

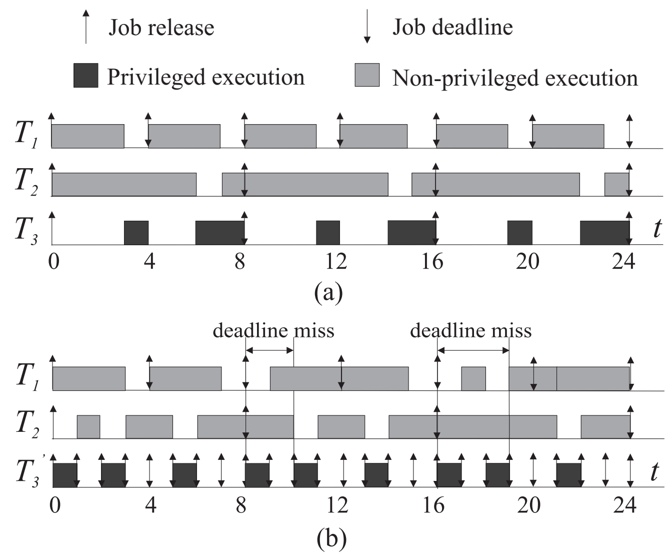


Figure 3: Schedules **(a)** without and **(b)** with temporary overloads.

In contrast, in EDF-ms, only jobs assigned to the system by the top and bottom rules are considered. Third, each privileged task T_k has a specified tardiness bound $\Delta_k \geq 0$ that must be ensured. In our case, $\Delta_k = 0$.

Of these difference, the second is the most significant. In the analysis of EDF-hl, it is assumed that $\sum u_i \leq m$ holds. For EDF-ms, we have instead $Z_0 + Z_{n+1} + \sum_{i=1}^{n} u_i \leq m$. Thus, under both schemes, long-term overloads cannot happen. However, under EDF-ms, short-term overloads *can* occur. This is not possible under EDF-hl. Thus, the analysis associated with EDF-hl must be adjusted to deal with short-term overloads. We illustrate this issue with an example.

Example 3. We consider two similar task sets τ_1 and τ_2 to be scheduled by EDF-hl and EDF-ms, respectively. Both have two non-privileged tasks $T_1 = (3, 4)$ and $T_2 = (7, 8)$. τ_1 has an addition privileged task $T_3 = (3, 8)$, while τ_2 has an additional privileged task $T_3' = (1, 1)$ for which $Z_3' = 3/8$. In τ_1, total utilization is two, so this system can be scheduled on two processors. In τ_2, the expression $Z_3' + \sum_{i=1}^{2} u_i$ is also two. However, while T_3 in τ_1 submits jobs at a steady rate according to its utilization, T_3' submits jobs at an unsteady rate, which leads to temporary overloads. To see this, consider the two schedules for τ_1 (a) and τ_2 (b) in Fig. 3. Because $e'_3 = p'_3$ holds for T'_3, its jobs that are assigned to the system by the bottom rule commence execution immediately after being released. The resulting temporary overloads cause some deadlines of non-privileged tasks to be missed. While such overloads can occur under EDF-ms, in both schedules, the amount of computation required by privileged tasks every eight time units is the same.

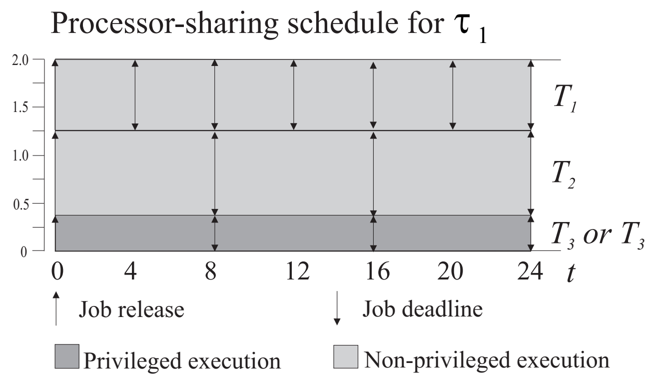


Figure 4: PS schedule for τ_1 and τ_2 in Example 3.

4.2 Tardiness Bound for EDF-hl

Because several of the properties established for EDF-hl by Devi and Anderson [5] are used in our analysis of EDF-ms, an overview of the EDF-hl analysis is in order. We begin by stating a number of definitions that are used in the analysis.

4.2.1 Definitions

The system start time is assumed to be zero. For any time $t > 0$, t^- denotes the time $t - \epsilon$ in the limit $\epsilon \to 0+$.

Definition 1 (active tasks and active jobs): A task T_i is *active* at time t if there exists a job $T_{i,j}$ (called T_i's *active job* at t) such that $r_{i,j} \leq t < d_{i,j}$. By our task model, every task has at most one active job at any time.

Definition 2 (pending jobs): $T_{i,j}$ is *pending* at t in a schedule $\mathcal{S}$ if $r_{i,j} \leq t$ and $T_{i,j}$ has not completed execution by t in $\mathcal{S}$. Note that a job with a deadline at or before t is not considered to be active at t even if it is pending at t.

A task system is *concrete* if the release times of all jobs are specified, and *non-concrete*, otherwise. The tardiness bound established for EDF-hl is derived by comparing the allocations to a concrete task system τ in an ideal processor-sharing (PS) schedule to those in an EDF-hl schedule. In a *PS schedule*, each job of a task T_i is executed at a constant rate of u_i between its release and deadline. As an example, consider Fig. 4, which shows the PS schedule for the task systems in Example 3. Note that, in a PS schedule, each job completes exactly at its deadline. Thus, if a job misses its deadline, then it is "lagging behind" the PS schedule—this concept of "lag" is instrumental in the analysis and is formalized below.

So that we may compare allocations in different schedules, let $\mathsf{A}(T_{i,j}, t_1, t_2, \mathcal{S})$ denote the total allocation to the job $T_{i,j}$ in an arbitrary schedule $\mathcal{S}$ in $[t_1, t_2)$. Similarly, let $\mathsf{A}(T_i, t_1, t_2, \mathcal{S}) = \sum_{j \geq 1} \mathsf{A}(T_{i,j}, t_1, t_2, \mathcal{S})$ denote the total time allocated to all jobs of T_i in $[t_1, t_2)$ in $\mathcal{S}$.

The difference between the allocations to a job $T_{i,j}$ up to time t in a PS schedule and an arbitrary schedule $\mathcal{S}$, termed the *lag of job $T_{i,j}$ at time t in schedule $\mathcal{S}$*, is given by

$$\mathsf{lag}(T_{i,j}, t, \mathcal{S}) = \mathsf{A}(T_{i,j}, 0, t, \mathsf{PS}) - \mathsf{A}(T_{i,j}, 0, t, \mathcal{S}).$$

The *lag of a task T_k at time t in schedule $\mathcal{S}$* is defined by

$$\begin{aligned} \mathsf{lag}(T_i, t, \mathcal{S}) &= \sum_{j \geq 1} \mathsf{lag}(T_{i,j}, t, \mathcal{S}) \\ &= \mathsf{A}(T_i, 0, t, \mathsf{PS}) - \mathsf{A}(T_i, 0, t, \mathcal{S}). \end{aligned}$$

Task lags and job lags are related as follows.

$$\mathsf{lag}(T_i, t, \mathcal{S}) = \sum_{\{T_{i,j} \text{ is pending or active at } t^-\}} \mathsf{lag}(T_{i,j}, t, \mathcal{S})$$

Finally, the *lag for a finite job set Ψ at time t in the schedule*

$\mathcal{S}$ is defined by

$$\mathsf{LAG}(\Psi, t, \mathcal{S}) = \sum_{T_{i,j} \in \Psi} \mathsf{lag}(T_{i,j}, t, \mathcal{S}).$$

The concept of lag is important because, if it can be shown that lags remain bounded, then tardiness is bounded as well.

Definition 3 (busy and non-busy intervals): A time interval $[t_1, t_2)$, where $t_2 > t_1$, is said to be *busy* for any job set Ψ if all m processors are executing some job in Ψ at each instant in the interval, *i.e.*, no processor is ever idle in the interval or executes a job not in Ψ. An interval $[t_1, t_2)$ that is not busy for Ψ is said to be *non-busy* for Ψ.

We are interested in non-busy intervals (for a job set) because total lag (for that job set) can increase only across such intervals. Such increases can lead to deadline misses. We illustrate this point with an example.

Example 4. The intervals $[3, 4)$ and $[7, 8)$ in the schedule for τ_2 in Fig. 3 are non-busy for the set of jobs with deadlines at most 8. The total lag for these jobs increases by one across each of these intervals. In effect, two units of processing capacity are "lost" during these intervals. As a result, the first job of T_2 misses its deadline at time 8 by two time units.

4.2.2 Tardiness-Bound Derivation

We describe the tardiness-bound derivation for EDF-hl for the case where each privileged task is ensured zero tardiness, as in EDF-ms. In this case, given an arbitrary non-concrete task system τ^N, we want to determine the maximum tardiness of any job of any non-privileged task in any concrete instantiation of τ^N. Let τ be a concrete instantiation of τ^N. Let $\tau_H \subseteq \tau$ be the set of at most m privileged tasks in τ, and let τ_L denote the remaining tasks. We remind the reader that all tasks are sporadic and each job of every privileged task is processed by the system. Let $T_{\ell,j}$ be a job of a non-privileged task in τ, let $t_d = d_{\ell,j}$, and let $\mathcal{S}$ be an EDF-hl schedule for τ with the following property.

(P) The tardiness of every job of every non-privileged task T_k in τ with deadline less than t_d is at most $x + e_k$ in $\mathcal{S}$, where $x \geq 0$.

Our goal is to determine the smallest x, independent of the parameters of T_ℓ, such that the tardiness of $T_{\ell,j}$ remains at most $x + e_\ell$. Such a result would by induction imply a tardiness of at most $x + e_k$ for all jobs of every non-privileged task $T_k \in \tau$. Because τ is arbitrary, the tardiness bound will hold for every concrete instantiation of τ^N.

Assume that $T_{\ell,j}$ misses its deadline (for otherwise, its tardiness is zero). The completion time of $T_{\ell,j}$ then depends on the amount of work that can compete with $T_{\ell,j}$ after t_d. Hence, a value for x can be determined via the following steps.

(S1) Compute an upper bound (UB) on the work pending for tasks in τ (including that due to $T_{\ell,j}$) that can compete with $T_{\ell,j}$ after t_d.

(S2) Determine a lower bound (LB) on the amount of such work required for the tardiness of $T_{\ell,j}$ to exceed $x + e_\ell$.

(S3) Determine the smallest x such that the tardiness of $T_{\ell,j}$ is at most $x + e_\ell$ using UB and LB.

With the exception of some jobs of privileged tasks, jobs with deadlines beyond t_d cannot affect $T_{\ell,j}$. Thus, our analysis focuses mostly on the following set of jobs.

$$\Psi \stackrel{\text{def}}{=} \text{set of all jobs of tasks in } \tau \text{ with deadlines at most } t_d$$

So that we can analyze the impact of jobs of privileged tasks, let the *carry-in* job $T_{k,j}$ of a privileged task T_k be defined as the job, if any, for which $r_{k,j} \leq t_d < d_{k,j}$. At most one such job could exist for each privileged task T_k. Similarly, let the job $T_{k,j'}$ of a privileged task T_k, if any, for which $r_{k,j'} \leq t_d + x + e_\ell < d_{k,j'}$, be defined as the *carry-out* job of T_k.

The competing work for $T_{\ell,j}$ beyond t_d is given by the sum of **(i)** the amount of work pending at t_d for jobs in Ψ, plus **(ii)** the amount of work demanded by jobs of privileged tasks that are not in Ψ but can compete with jobs in Ψ during $[t_d, t_d+x+e_\ell)$. By upper bounding these two components and summing them, we can obtain an upper bound UB.

Let $\mathsf{D}(T_k, t_d, \mathcal{S})$ be an upper bound (to be determined) on the work considered in (ii) generated by one privileged task T_k. Then, an upper bound on all the work considered in (ii) is given by

$$\mathsf{D}(t_d, \mathcal{S}) = \sum_{T_k \in \tau_H} \mathsf{D}(T_k, t_d, \mathcal{S}). \quad (5)$$

Turning now to the pending work mentioned in (i), because jobs from Ψ have deadlines at most t_d, they do not execute in the PS schedule beyond t_d. Thus, this pending work is given by $\mathsf{LAG}(\Psi, t_d, \mathcal{S})$. Let δ_k denote the amount of time the carry-in job (if one exists) of some task T_k has executed before t_d. The presence of carry-in jobs can cause $\mathsf{LAG}(\Psi, t_d, \mathcal{S})$ to be higher than it otherwise would have been by an additive factor of at most $\sum_{T_i \in \tau_H} \delta_i(1-u_i)$. This is because carry-in jobs have deadlines beyond t_d, and thus when they execute prior to t_d, they deprive the jobs in Ψ of processor time that may otherwise have been available to them. If a carry-in job executes prior to t_d in the actual schedule $\mathcal{S}$ for δ_i time, then while it is executing, it receives an allocation of $u_i \cdot \delta_i$ in the PS schedule. This means its lag changes by $u_i \cdot \delta_i - \delta_i$, which is a decrease. This lag decrease for carry-in jobs translates into a corresponding increased lag for the jobs in Ψ. It remains to understand how lags change in the absence of carry-in jobs. In this case, $\mathsf{LAG}(\Psi, t_d, \mathcal{S})$ is at most $\mathsf{LAG}(\Psi, t', \mathcal{S})$, where t' is the end of the latest non-busy interval for Ψ before t_d. In particular, LAG for Ψ cannot increase between t' and t_d, because all processors are busy in $[t', t_d)$ in the actual schedule $\mathcal{S}$. Combining these ideas, we have the following upper bound.

$$\mathsf{LAG}(\Psi, t_d, \mathcal{S}) \le \mathsf{LAG}(\Psi, t', \mathcal{S}) + \sum_{T_i \in \tau_H} \delta_i(1-u_i) \quad (6)$$

This upper bound is formally established in Lemmas 6 and 7 in [5]. It is important to note that the proofs of these lemmas only depend on Property (P) and the definition of a carry-in job (particularly, the term δ_i). In particular, the proofs do not depend on the exact manner in which the jobs of privileged tasks are scheduled. Note that Property (P) makes a very strong assumption about the execution of jobs from Ψ prior to t_d.

To continue, we partition the jobs in Ψ into two disjoint subsets: Ψ_H, which includes all jobs in Ψ of privileged tasks, and Ψ_L, which includes all remaining jobs in Ψ. Then, we have

$$\begin{aligned}&\mathsf{LAG}(\Psi, t', \mathcal{S}) \\ &\quad = \mathsf{LAG}(\Psi_L, t', \mathcal{S}) + \mathsf{LAG}(\Psi_H, t', \mathcal{S}). \quad (7)\end{aligned}$$

Let $U(\tau, y)$ denote the set of at most y tasks of highest utilization from the task set τ. Let $E(\tau, y)$ denote the set of at most y tasks with the highest execution costs from τ. Let $E_L = \sum_{u_i \in E(\tau_L, m-1)} e_i$ and $U_L = \sum_{e_i \in U(\tau_L, min(m-2, |\tau_L|))} u_i$. Using this notation, Lemma 6 from [5] establishes the following upper bound on $\mathsf{LAG}(\Psi_L, t', t_d)$.

$$\mathsf{LAG}(\Psi_L, t', \mathcal{S}) \le E_L + xU_L \quad (8)$$

As before, only Property (P) is used in establishing this bound. The exact manner in which privileged tasks are scheduled does not arise.

Let $\mathsf{L}(\tau_H, t', \mathcal{S})$ be an upper bound (to be determined) on the other LAG term in (7), $\mathsf{LAG}(\Psi_H, t', \mathcal{S})$. Then, from (7) and (8), we have the following.

$$\mathsf{LAG}(\Psi, t', \mathcal{S}) \le E_L + xU_L + \mathsf{L}(\tau_H, t', \mathcal{S}) \quad (9)$$

Combining (5), (6), and (9), the desired upper bound UB is

$$\begin{aligned}&\mathsf{LAG}(\Psi, t_d, \mathcal{S}) + \mathsf{D}(t_d, \mathcal{S}) \\ &\quad \le \quad E_L + xU_L + \sum_{T_i \in \tau_H} \delta_i(1-u_i) + \mathsf{L}(\tau_H, t', \mathcal{S}) +\end{aligned}$$

$\mathsf{D}(t_d, \mathcal{S}).$ (10)

Lemma 11 from [5] shows that the tardiness of $T_{\ell,j}$ is at most $x + e_\ell$ if one of the following conditions holds.

- $|\tau_H| < m$ and $\mathsf{LAG}(\Psi, t_d, \mathcal{S}) \leq (m - |\tau_H|)x + e_\ell$.
- $|\tau_H| > 0$ and $\mathsf{LAG}(\Psi, t_d, \mathcal{S}) + \mathsf{D}(t_d, \mathcal{S}) \leq (m - max(|\tau_H| - 1, 0)u_\ell)x + e_\ell$.

As before, these results only depend on Property (P). Thus, for tardiness to *exceed* $x + e_\ell$, we must have $\mathsf{LAG}(\Psi, t_d, \mathcal{S}) > (m - |\tau_H|)x + e_\ell$ for the case $|\tau_H| < m$, and $\mathsf{LAG}(\Psi, t_d, \mathcal{S}) + \mathsf{D}(t_d, \mathcal{S}) > (m - max(|\tau_H| - 1, 0)u_\ell)x + e_\ell$ for the case $|\tau_H| > 0$. These expressions give us the desired lower bound LB (for two different cases).

Now, if we set the upper bound on either $\mathsf{LAG}(\Psi, t_d, \mathcal{S}) + \mathsf{D}(t_d, \mathcal{S})$ or $\mathsf{LAG}(\Psi, t_d, \mathcal{S})$ implied by (10) to be at most its corresponding lower bound above, then the tardiness of $T_{\ell,j}$ will be at most $x + e_\ell$. (An upper bound on $\mathsf{LAG}(\Psi, t_d, \mathcal{S})$ alone is obtained from (10) by canceling $\mathsf{D}(t_d, \mathcal{S})$ from both sides.) By solving for the minimum x that satisfies both resulting inequalities, we obtain a value of x that is sufficient for ensuring a tardiness of at most $x + e_\ell$; we explain later how to obtain a value of x that is independent of the parameters of $T_{\ell,j}$ when we consider $\mathsf{EDF\text{-}ms}$. The two inequalities are as follows.

$$\begin{aligned}
&\mathsf{L}(\tau_H, t', \mathcal{S}) \\
&\quad \leq \quad (m - |\tau_H| - U_L)x - E_L \\
&\qquad - \sum_{T_i \in \tau_H} \delta_i (1 - u_i) + e_\ell
\end{aligned} \tag{11}$$

$$\begin{aligned}
&\mathsf{D}(t_d, \mathcal{S}) + \mathsf{L}(\tau_H, t', \mathcal{S}) \\
&\quad \leq \quad (m - max(|\tau_H| - 1, 0)u_\ell - U_L)x - E_L \\
&\qquad - \sum_{T_i \in \tau_H} \delta_i (1 - u_i) + e_\ell
\end{aligned} \tag{12}$$

In [5], it is shown that $\mathsf{L}(\tau_H, t', \mathcal{S}) = 0$ holds for $\mathsf{EDF\text{-}hl}$. This will not necessarily be the case for $\mathsf{EDF\text{-}ms}$.

4.3 Tardiness Bound for $\mathsf{EDF\text{-}ms}$

A tardiness bound for $\mathsf{EDF\text{-}ms}$ can be derived in a way that is similar to that used for $\mathsf{EDF\text{-}hl}$. Several aspects of the derivation remain the same. For example, we use the same notion of a PS schedule here. In addition, all of the reasoning concerning non-privileged tasks in the derivation for $\mathsf{EDF\text{-}hl}$ applies to $\mathsf{EDF\text{-}ms}$. This is because, as noted earlier, Property (P) allows us to reason about the scheduling of non-privileged tasks before t_d (and, in particular, their lags) without concern for exactly how the privileged tasks were scheduled prior to t_d. The only changes that are therefore required are those aspects of the derivation involving privileged tasks. Specifically, we must derive upper bounds of the two terms $\mathsf{D}(t_d, \mathcal{S})$ and $\mathsf{L}(\tau_H, t', \mathcal{S})$. These bounds are derived separately below.

4.3.1 An Upper Bound of D

The desired upper bound is provided in the following lemma.

Lemma 1. *Let T_k be one of the two privileged tasks scheduled by* $\mathsf{EDF\text{-}ms}$*. Let δ_k be as defined earlier. Then,* $\mathsf{D}(T_k, t_d, \mathcal{S}) \leq Z_k \cdot x + e_k(3 - f_k) + Z_k \cdot e_\ell - \delta_k(1 - Z_k)$.

Proof. If no job of T_k has a deadline in $[t_d, t_d + x + e_\ell)$, then $\mathsf{D}(T_k, t_d, \mathcal{S})$ is at most $e_k - \delta_k$, which proves the lemma. In the remainder of the proof, we assume that T_k has one or more job deadlines in $[t_d, t_d + x + e_\ell)$. In this case, the demand given by $\mathsf{D}(T_k, t_d, \mathcal{S})$ is comprised of three parts: **(i)** demand due to the carry-in job, which we denote $T_{k,ci}$: **(ii)** demand due to the carry-out job, which we denote $T_{k,co}$; and **(iii)** demand due to other jobs of T_k that have deadlines within $[t_d, t_d + x + e_\ell)$. Note that, in considering the demand created by these various jobs, we only have to consider those jobs assigned to the system by either the top or bottom rule (as the case may be). Also note that, if no carry-in or carry-out job exists, then the demand component mentioned in (i) or (ii), respectively, is simply zero.

Carry-in demand. If the carry-in job $T_{k,ci}$ has a deadline at $t_d + \xi$, then demand due to it is at most $min(e_k - \delta_k, \xi)$. In the rest of the proof, we assume that $t_d + \xi$ is so defined if the carry-in job exists, and if it does not exist, then $\xi = 0$.

Carry-out demand. Assume that the carry-out job $T_{k,co}$ is released at time $r_{k,co}$ and its (absolute) deadline is at time $d_{k,co} = r_{k,co} + p_k$. Since $T_{k,co}$ is a carry-out job, $d_{k,co} \geq t_d + x + e_\ell$. Since the carry-out job has a deadline after t_d, it cannot be prioritized over jobs in Ψ unless its slack is zero. The earliest time this can happen is $d_{k,co} - e_k = r_{k,co} + p_k - e_k$. If $d_{k,co} - e_k \geq t_d + x + e_\ell$, then the carry-out job is never prioritized over $T_{\ell,j}$ in $[t_d, t_d + x + e_\ell)$, *i.e.*, the carry-out demand is zero. If $d_{k,co} - e_k < t_d + x + e_\ell$, then this demand is

$$t_d + x + e_\ell - d_{k,co} + e_k = t_d + x + e_\ell - r_{k,co} - p_k + e_k. \quad (13)$$

Let $T_{k,co-1}$ be the job of T_k that precedes the carry-out job and let $d_{k,co-1} \geq t_d$ be its deadline—note that $T_{k,co-1}$ may or may not have been assigned to the system by the top or bottom rule. Let $\phi = t_d + x + e_\ell - d_{k,co-1}$. Then, because $d_{k,co-1} \leq r_{k,co}$, we have $\phi \geq t_d + x + e_\ell - r_{k,co}$. Thus, the demand in (13) is at most $\phi - p_k + e_k$. Combining these cases together, the carry-out demand is at most $max(0, e_k - (p_k - \phi))$.

Remaining demand. Jobs of T_k that are released and assigned to the system between $t_d + \xi$ and $t_d + x + e_\ell - \phi$ create the remaining demand. By (4), this demand is at most

$$\left\lceil f_k \left\lceil \frac{x + e_\ell - \phi - \xi - p_k}{p_k} \right\rceil \right\rceil e_k.$$

Having upper bounded the three relevant sources of demand, we now show that their sum is upper bounded by the expression given in the lemma. If $x + e_\ell - \phi - \xi - p_k \leq 0$, then the third demand component is zero, and we have $max(0, e_k - (p_k - \phi)) + min(e_k - \delta_k, \xi) \leq 2e_k - \delta_k$. It can be shown that this last expression is at most $e_k(3 - f_k) + Z_k \cdot x + Z_k \cdot e_\ell - \delta_k(1 - Z_k)$.

The remaining possibility is that $x + e_\ell - \phi - \xi - p_k > 0$. In this case, we have the following.

$$\begin{aligned}
&\mathsf{D}(T_k, t_d, \mathcal{S}) \\
&\leq min(e_k - \delta_k, \xi) \\
&\quad + \left\lceil f_k \left\lceil \frac{x + e_\ell - \phi - \xi - p_k}{p_k} \right\rceil \right\rceil e_k \\
&\quad + max(0, e_k - (p_k - \phi)) \\
&\leq min(e_k - \delta_k, \xi) + e_k + e_k f_k \left(\frac{x + e_\ell - \phi - \xi}{p_k} \right) \\
&\quad + max(0, e_k - (p_k - \phi)) \\
&= min(e_k - \delta_k, \xi) + e_k + Z_k(x + e_\ell - \phi - \xi) \\
&\quad + max(0, e_k - (p_k - \phi)) \\
&\quad \{\text{because, by (3), } f_k e_k / p_k = Z_k\} \\
&= min(e_k - \delta_k, \xi) + e_k \\
&\quad + max(Z_k(x + e_\ell - \phi - \xi), \\
&\quad\quad Z_k(x + e_\ell - \phi - \xi) + e_k - (p_k - \phi))
\end{aligned}$$

Before continuing the derivation, note that $Z_k(x + e_\ell - \phi - \xi) + e_k - (p_k - \phi) = Z_k(x + e_\ell - \xi) - Z_k \cdot \phi + e_k - (p_k - \phi) = Z_k(x + e_\ell - \xi) + (\phi(1 - Z_k) + e_k - p_k)$. Because $0 \leq \phi \leq p_k$, this expression is maximized when $\phi = p_k$. Thus, $Z_k(x + e_\ell - \xi) + (\phi(1 - Z_k) + e_k - p_k) \leq Z_k(x + e_\ell - \xi) + e_k - Z_k \cdot p_k = Z_k(x + e_\ell - \xi) + e_k(1 - f_k)$ ((3) is used in the last step). Because $Z_k(x + e_\ell - \xi - \phi) \leq Z_k(x + e_\ell - \xi)$, we can continue the derivation as follows to conclude the proof.

$$\begin{aligned}
&\mathsf{D}(T_k, t_d, \mathcal{S}) \\
&\leq min(e_k - \delta_k, \xi) + e_k + Z_k(x + e_\ell - \xi) + \\
&\quad e_k(1 - f_k) \\
&\leq e_k - \delta_k + e_k + Z_k(x + e_\ell - (e_k - \delta_k)) \\
&\quad + e_k(1 - f_k) \\
&\quad \{\text{because } \xi \geq e_k - \delta_k\} \\
&\leq e_k(3 - f_k) - \delta_k + Z_k \cdot x + Z_k \cdot e_\ell - Z_k(e_k - \delta_k) \\
&= e_k(3 - f_k) + Z_k \cdot x + Z_k(e_\ell - e_k) - \delta_k(1 - Z_k) \\
&\leq e_k(3 - f_k) + Z_k \cdot x + Z_k \cdot e_\ell - \delta_k(1 - Z_k)
\end{aligned}$$

■

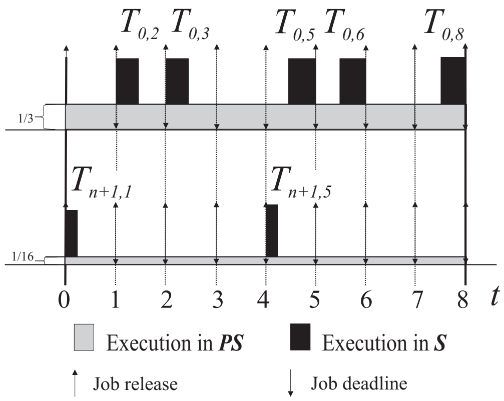


Figure 5: Schedules for tasks T_0 and T_{n+1} in Example 5.

4.3.2 An Upper Bound of L

The remaining issue is to determine an upper bound of $\mathsf{L}(\tau_H, t', \mathcal{S})$, which is simply the sum of the lags of the two privileged tasks. In fact, we will show that the lag of each of these tasks is bounded by a constant. This follows because the top and bottom rules assign jobs of these tasks to the system in a way that prevents long-term overloads, and also because such jobs are scheduled in a way that ensures that they do not miss their deadlines. The needed bounds are quite simple and are given in the following lemma, which we state without proof.

Lemma 2. *Let T_0, T_{n+1}, and τ be as defined above. Then, for any $t \geq 0$, $\mathsf{lag}(T_0, t, \mathcal{S}) \leq e_0 + Z_0(p_0 - e_0)$ and $\mathsf{lag}(T_{n+1}, t, \mathcal{S}) \leq Z_{n+1}(p_{n+1} - e_{n+1})$.*

Example 5. We illustrate the lemma by considering a system with a top privileged task T_0 with $e_0 = 1/2$, $p_0 = 1$, and $Z_0 = 1/3$, and a bottom privileged task T_{n+1} with $e_{n+1} = 1/4$, $p_{n+1} = 1$, and $Z_{n+1} = 1/16$. The corresponding frequencies are $f_0 = Z_0/u_0 = 2/3$ and $f_{n+1} = Z_{n+1}/u_{n+1} = 1/4$. Fig. 5 depicts both actual and PS schedules for each task. Let us determine $\mathsf{lag}(T_0, 4.5, \mathcal{S})$. Up to time 4.5, T_0 is allocated $Z_0 \cdot t = 4.5 \cdot 1/3 = 1.5$ time units in the PS schedule, and $2 \cdot 1/2 = 1$ time unit in the actual schedule. Therefore, $\mathsf{lag}(T_0, 4.5, \mathcal{S}) = 0.5$. The upper bound in Lemma 2, applied to T_0, is 2/3.

From Lemma 2, we have the desired upper bound.

Corollary 1. $\mathsf{L}(\tau_H, t', \mathcal{S}) \leq e_0 + \sum_{k \in \{0, n+1\}} Z_k(p_k - e_k)$.

4.3.3 Tardiness Bound Derivation

Applying the upper bound in Corollary 1 to (11), we get

$$\begin{aligned} & e_0 + \sum_{k \in \{0,n+1\}} Z_k(p_k - e_k) \\ \leq\ & (m - |\tau_H| - U_L)x \\ & -E_L - \sum_{k \in \{0,n+1\}} \delta_k(1 - Z_k) + e_\ell. \end{aligned}$$

Note that, in the last summation, we have used Z_k instead of u_k because under $\mathsf{EDF\text{-}ms}$, Z_k represents T_k's actual utilization in the system. If we upper bound δ_k by e_k and solve for x, we get

$$x \geq \frac{E_L + e_0 + \sum_{k \in \{0,n+1\}} e_k(1 + f_k - 2Z_k) - e_\ell}{m - |\tau_H| - U_L} \tag{14}$$

Applying the upper bounds in Lemma 1 and Corollary 1 to (12), we get

$$\begin{aligned} & \sum_{k \in \{0,n+1\}} (e_k(3 - f_k) + Z_k \cdot x + Z_k \cdot e_\ell \\ & \quad -\delta_k(1 - Z_k)) + e_0 + \sum_{k \in \{0,n+1\}} Z_k(p_k - e_k) \\ \leq\ & (m - max(|\tau_H| - 1, 0)u_\ell - U_L)x \\ & \quad -E_L - \sum_{k \in \{0,n+1\}} \delta_k(1 - Z_k) + e_\ell. \end{aligned}$$

As before, we have used Z_k instead of u_k in the last summation. Solving for x, we get

$$x \geq \frac{e_0 + E_L + \sum_{k \in \{0,n+1\}} (e_k(3 - Z_k) + Z_k \cdot e_\ell) - e_\ell}{m - max(|\tau_H| - 1, 0)u_\ell - U_L - Z_0 - Z_{n+1}}. \tag{15}$$

If x is the smaller of the two values on the right-hand sides of (14) and (15), then the tardiness of $T_{\ell,j}$ will not exceed $x + e_\ell$. Let $e_{max} = max(e_i)$ and $e_{min} = min(e_i)$. Then, a value for x that is independent of the parameters of T_ℓ can be obtained by replacing u_ℓ with $max_{T_i \in \tau_L} u_i$, e_ℓ with e_{min} in (14), and the expression $(Z_0 + Z_{n+1} - 1)e_\ell$ with EL in (15), where

$$EL = \begin{cases} (Z_0 + Z_{n+1} - 1)e_{min} & \text{if } Z_0 + Z_{n+1} \leq 1 \\ (Z_0 + Z_{n+1} - 1)e_{max} & \text{otherwise.} \end{cases}$$

Theorem 1. *With x as defined as above, tardiness for a non-privileged task T_k scheduled under* EDF-ms *is at most $x + e_k$.*

Note that, for tardiness to be bounded under EDF-ms, the denominators in the right-hand-side expressions in (14) and (15) must not be zero. Upon substituting $max_{T_i \in \tau_L} u_i$ for u_ℓ, this gives us two requirements, $m - |\tau_H| - U_L > 0$, and $m - max(|\tau_H| - 1, 0)(max_{T_i \in \tau_L} u_i) - Z_0 - Z_{n+1} - U_L > 0$. Thus, to ensure bounded tardiness, some slight restrictions on task utilizations are required. (The impact of these restrictions is assessed in the next section.)

We have assumed above that there are two privileged tasks. If only one such task exists, then we can assume there are two, with the execution cost, utilization, and Z and f values for one of them being zero. Of course, if there are no privileged tasks, then tardiness can be analyzed using the results from [5].

We have also required that there be at least two cores per group. A group with one core can be handled in three ways.

- We can use the same approach described above, but limit the group to have at most one privileged task. In this case, the group's lone core may not be fully utilized.

- We can schedule the tasks within the group like EDF-fm schedules tasks on one processor. Note that EDF-fm requires that if two privileged tasks exist, then they have a combined share on the processor of at most one. On the other hand, the processor can be fully utilized.

- We can combine the lone core in the group with slower cores to create a group of at least two cores. This comes at the expense of not utilizing the full processing capacity of the core added to the slower group.

The best approach will depend on the workload to be scheduled.

5 Experimental Evaluation

In this section, we present an experimental evaluation of EDF-ms. We performed two sets of experiments. In the first, we assessed tardiness within a single group. In the second, we assessed the impact of several variants of the task-assignment method discussed in Sec. 2 on overall tardiness.

5.1 Tardiness Bounds for a Single Group

In this set of experiments, we computed per-task tardiness bounds for random task sets on $m = 2$, 4, 8, and 16 unit-speed processors in the presence of one or two privileged tasks (top or bottom). Each task set consisted of at least $m + 1$ tasks. Tasks within each set were generated with utilizations uniformly distributed in $[0, u_{max})$, where u_{max} ranged from 0.1 to 1 in steps of 0.05. For each value of u_{max}, 1,000 task sets were generated. Task execution costs were uniformly distributed over $[10, 20)$. Tasks were added to the generated task set until total utilization exceeded m. The shares of the privileged tasks were then defined so that $Z_0 + Z_{n+1} + \sum_{i=1,\ldots n} u_i = m$. The top (bottom) task (if either existed) was taken to be the task with the smallest (largest) utilization.

Fig. 6 shows the average *maximum* task tardiness plotted against the average task utilization, u_{avg}, for different values of m. Note that tardiness grows as u_{avg} grows, with the exception of the case of two processors and one privileged task, shown in inset (a). In this case, if we apply Theorem 1 for $m = 2$, then $U_L = 0$, and only one of Z_0 and Z_{n+1} is non-zero and it is at most one. In this case, the denominators of (14) and (15) are independent of the utilizations of non-privileged tasks.

The situation for $m = 2$ changes drastically if there are two privileged tasks. During some time intervals, both processors in the group must execute jobs of privileged tasks. During such intervals, non-privileged tasks cannot execute at all. If the number of non-privileged tasks is small, and they have high utilizations, then these tasks recover slowly from this shortage of processing capacity, as demand due to privileged tasks lessens. This situation is depicted in the right part of inset (a), where non-privileged tasks have high utilizations.

As seen in insets (b)–(d), this effect eases as the number of processors grows. This is because, with more than two processors, at least one processor is always available to execute non-privileged tasks. As the number of processors grows, more processing capacity is available for executing the jobs of non-privileged tasks. Hence, tardiness decreases. This suggests that EDF-ms may be very effective in large multicore systems, the main focus of our work.

Note that the curves in insets (a) and (b) for the case of two privileged tasks do not continue to increase at the right. This is because, when m is only two or four but two privileged tasks exist, the number of samples with high u_{avg} is small.

As remarked earlier, bounded tardiness is guaranteed under EDF-ms only if the two conditions $m - max(|\tau_H| - 1, 0)max_{T_i \in \tau_L}(u_i) - Z_0 - Z_{n+1} - U_L > 0$ and $m - |\tau_H| - U_L > 0$ hold. These conditions are not very restrictive. As evidence of this, no task set generated in this set of experiments had to be rejected because of these conditions.

5.2 Task Assignments, Revisited

Because tardiness within a group depends on the parameters of the privileged tasks in that group, it might be possible to lower overall tardiness by using a task-assignment policy that lessens the impact of privileged tasks on other tasks. To see if this is so, we considered two such policies and compared them to the one described in Sec. 2. In that which follows, we refer to original policy described earlier as SIMPLE.

According to Theorem 1, if the privileged tasks within a group either require large shares within the group or have high execution costs, then tardiness within the group may be high. This suggests two alternative assignment policies, one that seeks to minimize the shares of privileged tasks, and a second that seeks to minimize their execution costs. Both policies function in a similar way: after running SIMPLE, consider the

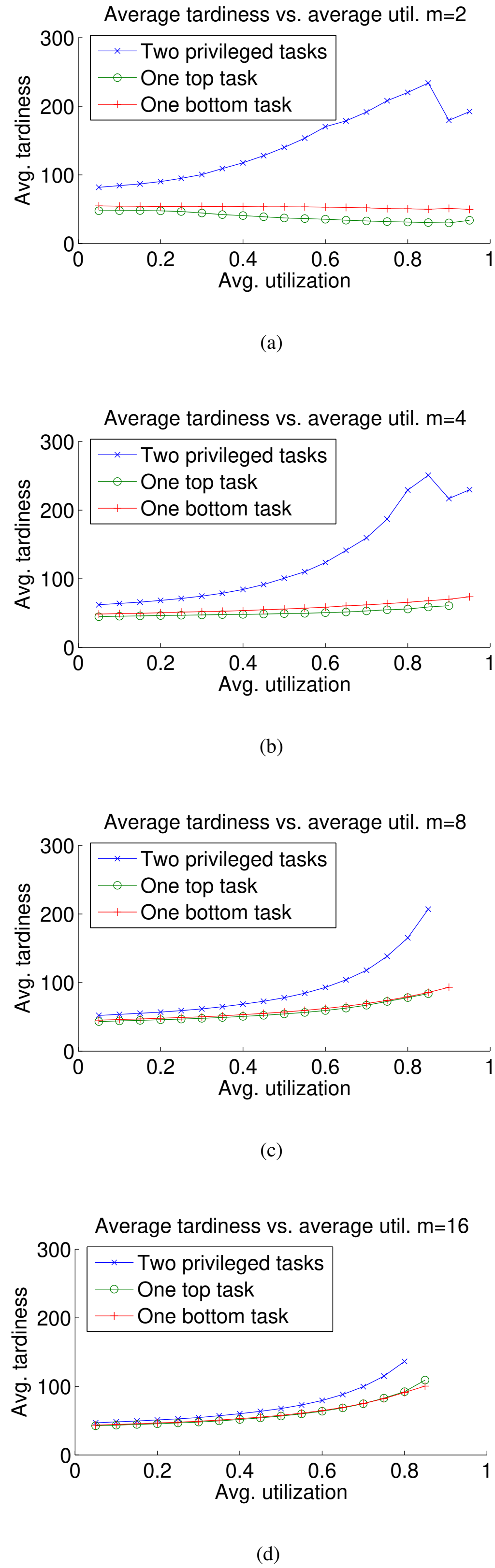
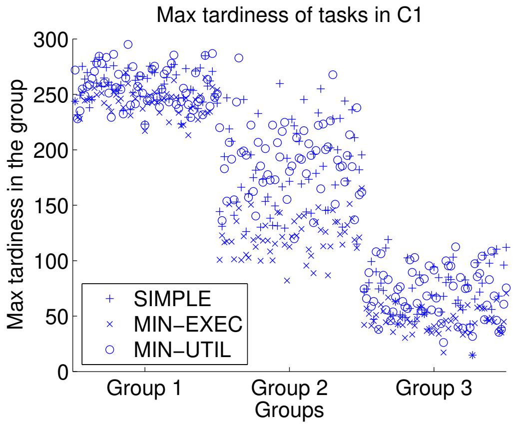
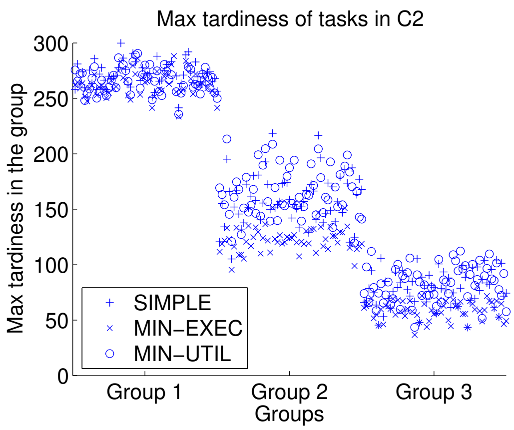


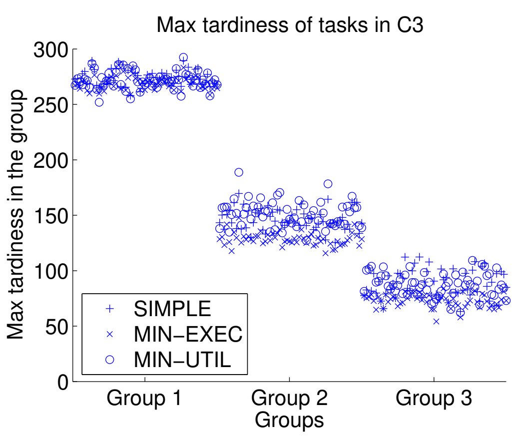
Figure 6: Tardiness bounds versus u_{avg} for **(a)** $m = 2$, **(b)** $m = 4$, **(c)** $m = 8$, and **(d)** $m = 16$.



(a)



(b)



(c)

Figure 7: Tardiness bounds under different assignment schemes for three core groups in three different configurations: **(a)** C_1, **(b)** C_2, and **(c)** C_3.

	Core groups			
	$s_1 = 1$	$s_2 = 2$	$s_3 = 3$	M
C_1	12	4	2	18
C_2	24	8	4	36
C_3	48	16	8	72

Table 1: Three evaluated configurations.

groups in decreasing index order, and for each group, select as the bottom task the task in the group with nonzero share that has the smallest utilization or execution cost. We call the former scheme MIN-UTIL, and the latter, MIN-EXEC. (If a task with a lower utilization is selected as the bottom task in Group h, then it may actually fit within Group $h + 1$. Thus, in both schemes, the process of assigning tasks to groups is in fact iterative.)

To evaluate the impact of MIN-UTIL and MIN-EXEC, we considered three system configurations C_1, C_2, and C_3, which have a small, medium, and large total number of cores M. Each configuration consists of cores with speeds one, two, and three. The number of cores of each type is shown in Table 1. For each configuration, we evaluated 60 task sets. The tasks in each set were generated as follows. First, tasks with utilizations distributed randomly in $[0, 2.1)$ were generated until the processing capacity of Group 3 would be exceeded. Then, tasks with utilizations in $[0, 1.4)$ were generated until the combined processing capacities of Groups 2 and 3 would be exceeded. Finally, the remaining tasks were generated with utilizations in $[0, 0.7)$ until the remaining capacity of the system was exhausted. All task execution costs were distributed uniformly over $[1, 100)$. For each generated task system, we used Theorem 1 to compute the maximum tardiness of the non-privileged tasks in each group under each assignment scheme.

Fig. 7 shows the maximum tardiness per group for each configuration. Each point in each group gives the maximum tardiness of one of the generated task sets. As the graphs show, MIN-EXEC results in significantly lower tardiness for Group 2 and slightly lower tardiness for the other groups. On the other hand, the use of MIN-UTIL did not result in better tardiness than SIMPLE. However, this could be an artifact of our task-generation methodology. The overall conclusion to be drawn from these results is that the significant flexibility that exists in the task-assignment process can be exploited to realize certain benefits in some systems. (In particular, this assignment process is not rigid like the bin-packing strategies used in partitioning schemes.) Other benefits beyond lowering tardiness are possible. For example, some hard real-time tasks could be supported by choosing them as intergroup tasks. Also, the response times of certain tasks could be lowered by assigning them to faster cores, as long as the resulting assignment is valid and utilization constraints are met.

6 Conclusion

We have presented a new algorithm, EDF-ms, which can be used for scheduling sporadic soft real-time task systems on asymmetric multicore platforms with cores of different speeds. To our knowledge, this paper is the first to propose a scheduling approach for such heterogeneous platforms that is suitable for soft real-time workloads that require bounded deadline tardiness. Our algorithm is capable of fully utilizing the processing capacity of the system, provided certain very slight restrictions on task utilizations hold. This property comes at the price of needing to migrate tasks, as required in global scheduling approaches such as GEDF. Note that the main cost of a migration is a loss of cache affinity. Thus, in a multicore platform, the need to migrate tasks is less of a concern than for a traditional SMP, due to the presence of shared on-chip caches. Although we have not directly included migration costs in our task model, they can be accounted for by inflating task execution costs to include the cost of migrations, as is commonly done in real-time scheduling analysis.

Several interesting avenues for further work exist. For example, it would be interesting to extend our results to include tasks with synchronization requirements. It would also be interesting to consider workloads with both soft real-time and non-real-time tasks. Finally, in this paper we have considered heterogeneous platforms where the cores only differ in speed. This is different from *functional asymmetry*, where each core has a different set of "capabilities" and tasks must be matched with cores possessing the capabilities they need. It would be interesting to extend our results to apply to such platforms.

References

[1] J. Anderson, V. Bud, and U. Devi. An EDF-based scheduling algorithm for multiprocessor soft real-time systems. In *Proc. of the 17th Euromicro Conf. on Real-Time Systems*, pp. 199–208, 2005.

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

[3] J. Calandrino, D. Baumberger, T. Li, S. Hahn, and J. Anderson. Soft real-time scheduling on performance asymmetric multicore platforms. In *Proc. of the 13th IEEE Real-Time and Embedded Technology and Applications Symp.*, pp. 101–110, 2007.

[4] U. Devi and J. Anderson. Tardiness bounds for global EDF scheduling on a multiprocessor. In *Proc. of the 26th IEEE Real-Time Systems Symp.*, pp. 330–341, 2005.

[5] U. Devi and J. Anderson. Flexible tardiness bounds for sporadic real-time task systems on multiprocessors. In *Proc. of the 20th IEEE International Parallel and Distributed Processing Symp.*, 2006.

[6] S. Funk. *Implementing Real-time Systems on Heterogeneous Multiprocessors*. Ph.D. dissertation, The University of North Carolina at Chapel Hill, 2004.

[7] R. Kumar, D. Tullsen, P. Ranganathan, N. Jouppi, and K. Farkas. Single-ISA heterogeneous multi-core architectures for multithreaded workload performance. *Proc. of the 31st International Symp. on Computer Architecture (ISCA)*, pp. 64–75, 2004.

[8] R. Rajkumar. Resource Kernels: Why Resource Reservation should be the Preferred Paradigm of Construction of Embedded Real-Time Systems. Keynote talk, 18th Euromicro Conference on Real-Time Systems, Dresden, Germany, 2006.