Priority Point Exploration in EDF-Like Scheduling for Self-Suspending Tasks

Mario Günzel, Kuan-Hsun Chen, Jian-Jia Chen, and Ching-Chi Lin
TU Dortmund University, Germany
University of Twente, The Netherlands

in: Workshop on OPtimization for Embedded and ReAl-time systems (OPERA) co-located with the 44th IEEE Real-Time Systems Symposium (RTSS). See also BibTeX entry below.

BibTeX:

```
@inproceedings{9188172,
  author={Mario Günzel and Kuan-Hsun Chen and Jian-Jia Chen and Ching-Chi Lin},
  booktitle={Workshop on OPtimization for Embedded and ReAl-time systems (OPERA)
    co-located with the 44th IEEE Real-Time Systems Symposium (RTSS)},
  title={Priority Point Exploration in EDF-Like Scheduling for Self-Suspending Tasks},
  year={2023},
}
```

Priority Point Exploration in EDF-Like Scheduling for Self-Suspending Tasks

Mario Günzel*, Kuan-Hsun Chen†, Jian-Jia Chen* and Ching-Chi Lin*
*TU Dortmund University, Germany
Email: {mario.guenzel, jian-jia.chen, chingchi.lin}@tu-dortmund.de
†University of Twente, The Netherlands
Email: k.h.chen@utwente.nl

***Abstract*—The choice of a scheduler has major impact on the schedulability of a task set. While Earliest-Deadline-First (EDF) provides the best schedulability guarantee for tasks without self-suspension, the existing proof of optimality does not hold anymore when scheduling self-suspending tasks. In such a case, empirical searches can be conducted to identify a scheduler that makes a task set schedulable. However, due to the extensive amount of priority-based schedulers, finding a proper scheduler can be challenging.**

In this work, we employ EDF-Like (EL) scheduling, which allows us to describe a large amount of priority-based schedulers by setting a relative priority point for each task. We propose two approaches for finding relative priority points that describe a scheduler that ensures the schedulability of the task set.

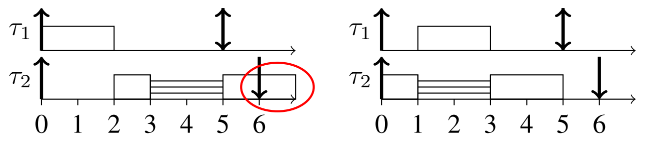
I. Introduction

Schedulability is a fundamental property of real-time systems. Given a task set, we can apply schedulability tests corresponding to a scheduling algorithm to verify whether every task meets its deadline. One typical approach is to ensure that the worst-case response time (WRCT) R_i of each task τ_i does not exceed its relative deadline D_i.

The schedulability of a task set can be affected by the choice of the underlying scheduling algorithm. In this work, we consider job-level fixed-priority preemptive scheduling algorithms, in which each job is assigned a priority and at each point in time the pending job with the highest priority is executed. One prime example for job-level fixed-priority scheduling algorithms is the Earliest-Deadline-First (EDF) scheduling algorithm, where jobs are prioritized in order of ascending absolute deadlines.

It is shown that EDF scheduling provides the best schedulability guarantees for scheduling ordinary tasks [4]. However, in the case of self-suspending tasks, i.e., tasks that voluntarily yield the processor before completing their executions, the existing proofs of optimality may no longer hold [1]. For instance, elevating the priority of the first job of τ_2 over the first job of τ_1 in Figure 1 transforms the previously unschedulable task set under EDF into a schedulable one.

When scheduling a task set containing self-suspending tasks, empirically finding the job priorities that lead to schedulability can be challenging due to the extensive search space. To address this issue, we adopt EDF-Like (EL) scheduling [3], which enables the assignment of a task-specific priority point (PP) Π_i for each task τ_i. By configuring the PPs accordingly,



(a) Not schedulable using EDF (b) Give τ_2 a higher priority

Figure 1: Task priority affects schedulability. Task τ_2 is a self-suspending task. Scheduling the task set using EDF leads to a deadline violation of τ_2 at time 6 since τ_1 has a higher priority.

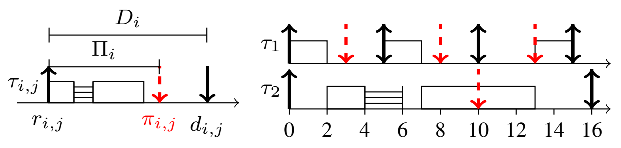
EL scheduling can cover a variety of different scheduling algorithms, including deadline-monotonic (DM) and EDF. In this work, we propose possible approaches and ideas on how to generate appropriate PPs to make a self-suspending task set schedulable, based on the schedulability test from our previous work [2]. Specifically, our contributions are as follows.

Contribution: We present two approaches in finding the PPs in EL scheduling that makes a given task set schedulable in Section III. The first approach tunes the PPs by iteratively increasing the PPs of tasks with WCRT less than their deadline. In the second approach, we design a genetic algorithm in finding a general rule for generating PPs based on the task parameters. We discuss further optimization ideas and open research questions in Section V.

II. System Model

Let $\mathbb{T} = \{\tau_1, \ldots, \tau_n\}$ be a set of *dynamic self-suspending sporadic* real-time tasks in a uniprocessor system. Each task τ_i is described by a 4-tuple $\tau_i = (C_i, S_i, D_i, T_i)$, composed of the worst-case execution time (WCET) $C_i > 0$, the maximum suspension time $S_i \geq 0$, a relative deadline $D_i > 0$, and the minimum inter-arrival time $T_i > 0$. Each task τ_i releases an infinite number of jobs $(\tau_{i,j})_{j\in\mathbb{N}}$, where $r_{i,j}$ and $d_{i,j} = r_{i,j} + D_i$ denote the release time and the absolute deadline of the j-th job of τ_i, respectively. We denote by $U_i := \frac{C_i}{T_i}$ the utilization of task τ_i and by $U := \sum_{i=1}^{n} U_i$ the total utilization of the whole task set $\mathbb{T}$.

In EDF-Like (EL) scheduling, jobs are scheduled according to their *absolute priority points* $\pi_{i,j} \in \mathbb{R}$. A job $\tau_{i,j}$ has higher priority than $\tau_{i',j'}$ if $\pi_{i,j} < \pi_{i',j'}$. The absolute PP is computed by summing the release of the job and a task specific parameter Π_i denoted as *relative priority point*, i.e., $\pi_{i,j} = r_{i,j} + \Pi_i$. Figure 2a depicts the notation, and Figure 2b



(a) notations (b) Example of EL Scheduling

Figure 2: (a) Notations used in this work. (b) Two tasks scheduled under EL scheduling. All jobs finish until their deadline (i.e., the schedule is feasible) and job priority is given by the absolute PPs $\pi_{1,1} < \pi_{1,2} < \pi_{2,1} < \pi_{1,3}$.

shows an example of EL scheduling for two simple tasks, namely

- $\tau_1 = (C_1 = 2, S_1 = 0, D_1 = 5, T_1 = 5)$ and
- $\tau_2 = (C_2 = 8, S_2 = 2, D_2 = 16, T_2 = 16)$,

with relative PPs $\Pi_1 = 3$ and $\Pi_2 = 10$.

III. Approaches for Finding the Priority Points

We present two approaches for finding relative PPs that describe a scheduling algorithm that ensures the schedulability of the task set.

A. Iterative Tuning

Given a task set $\mathbb{T}$, we initialize the relative PP to $\Pi_i = D_i$ as a configuration for typical EDF behavior. We then compute the WCRT R_i of each task τ_i using the analysis from our previous work [2]. If there exists a task τ_i such that $R_i > D_i$, then the task set is not guaranteed to be schedulable. In that case we want to tune the R_i by modifying the relative priority points Π_i. More specifically, we aim to reduce the R_i which are $> D_i$. The following observation describes a connection between modifications of Π_i and the reduction of some R_i. This observation is the backbone of our tuning process.

Observation 1. *Increasing the relative PP of τ_i by $X > 0$ does not increase its WCRT by more than X and does not increase the WCRT of any other task. Instead, it favors the execution of other jobs with higher relative PPs, which potentially reduces their WCRTs.*

For task sets that are not schedulable, we tune the relative PPs to improve the schedulability with the following steps:

1) Compute the set I of all tasks τ_i with $R_i < D_i$.
2) Increase the relative PP for tasks τ_i in the set I by setting $\Pi_i := \Pi_i + D_i - R_i$.

The process is repeated until a) the task set becomes schedulable, b) a predefined iteration limit is reached, or c) $I = \emptyset$.

B. Genetic Algorithm

Aside from iteratively tuning the relative PPs for each individual task set, we also aim to find a general expression for generating relative PPs using a genetic algorithm. Given a data set consists of m task sets $\mathbb{T}_1, \ldots, \mathbb{T}_m$, our objective is to construct a function $f(C, S, D, T)$ which maps the task parameters to a relative PP such that the number of schedulable task sets is maximized. In our preliminary attempt, we assume

$$f = C^{g_1} \cdot S^{g_2} \cdot T^{g_3} + g_4 \cdot C + g_5 \cdot S + g_6 \cdot T, \qquad (1)$$

where $g_1, \ldots, g_6 \in \mathbb{R}$ are the coefficients to be determined. We use these coefficients as the genes in a chromosome, i.e., $\chi = [g_1, \ldots, g_6]$, and the number of schedulable task sets as the fitness function in our genetic algorithm. The function f is chosen because it allows multiplication of C, S and T (e.g., $f = CST$ with $\xi = [1, 1, 1, 0, 0, 0]$ or $f = CS$ with $\xi = [1, 1, 0, 0, 0, 0]$) and also approximates linear combinations of C, S and T (if $g_1 = g_2 = g_3$ approaches $-\infty$).

Our genetic algorithm works as follows:

1) The *initialization* operation randomly generates μ chromosomes, where μ is the size of the population in one generation. To randomly generate a chromosome, $[g_1, \ldots, g_6]$ is drawn uniformly at random from $[-1, 1]^6$.
2) For each iteration, the *crossover* operation keeps selecting two chromosomes χ_1 and χ_2 randomly from the previous generation, and perform uniform crossover to produce a new chromosome χ' until there are 10μ new chromosomes.
3) For each gene χ' achieved by crossover, the *mutation* operation adds a value drawn uniformly from the interval $[-0.2, 0.2]$ to each gene.
4) To introduce variety, we also randomly generate 5μ new chromosomes (drawn uniformly at random from $[-1, 1]^6$) and add them to the chromosome pool after the crossover operation.
5) Finally, the *selection* operation selects μ chromosomes with the highest fitness value, i.e., the number of schedulable task sets, among the previous generation and the newly generated chromosomes as a new generation. To avoid overfitting, we generate 300 new tasksets for each selection operation using the UUnifast algorithm. To achieve this, the UUnifast algorithm is used to generate 50 tasksets with 10 tasks each for each utilization in $[0.4, 0.5, \ldots, 0.9]$.

IV. Preliminary Evaluation

We generated a data set containing 500 task sets for each utilization $U = 0.1, 0.2, \ldots, 1.0$. The WCETs of tasks were generated using the UUnifast algorithm and the suspension was randomly drawn from the interval $[0, 0.5] \cdot (T_i - C_i)$ for each task τ_i. For **Iterative Tuning**, we set the maximal number of iterations to 200. For **Genetic Algorithm**, we used a population size of $\mu = 10$. After 100 generations, the highest performing chromosome is

$$\chi = [0.42, -0.10, 0.76, -0.52, -3.79, 2.47], \qquad (2)$$

which translates to

$$f = C^{0.42} \cdot S^{-0.10} \cdot T^{0.76} - 0.52 \cdot C - 3.79 \cdot S + 2.47 \cdot T. \quad (3)$$

We evaluated the acceptance ratio of the task sets using the relative PPs generated by the two approaches, and compared them with **EDF** and Deadline Monotonic (**DM**) scheduling. Note that for EDF and DM scheduling, the state-of-the-art schedulability analysis is utilized. Figure 3 demonstrates the results. In that figure, **EL IT** illustrates the acceptance ratio of the iterative tuning approach, and **EL GA** illustrates the

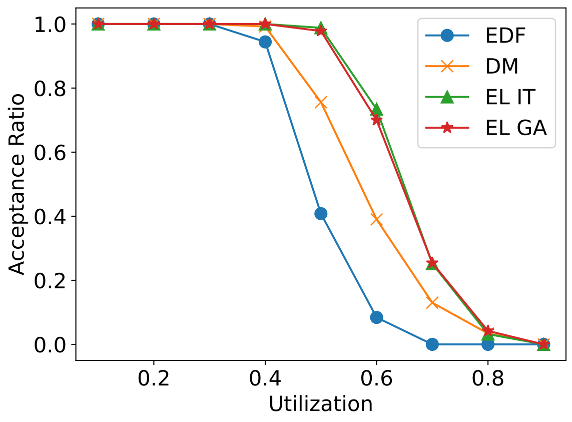


Figure 3: Acceptance ratio of different PP configurations.

acceptance ratio obtained by the genetic algorithm. We observe that the acceptance ratio of the task sets is significantly improved by applying the relative PPs generated by the two approaches.

V. Discussion and Open Questions

In our preliminary evaluation, we have shown that the acceptance ratio of task sets can be significantly improved by tuning the relative PPs. Still, there are several open questions that need to be addressed.

First, the analytical reasoning behind **Iterative Tuning** is not fully developed yet. Major questions are whether Observation 1 holds and whether the tuning process always improves the schedulability of the task set. Moreover, it is unclear how the choice of the initial relative PPs affects the result.

Furthermore, instead of reducing the priority of all tasks τ_i with $R_i < D_i$ by increasing their relative PP by $D_i - R_i$, we could instead increase the priority of all tasks with $R_i > D_i$ by reducing their relative PP by $R_i - D_i$. Which approach is better and how much should the relative PP be increased or decreased remains to be investigated. However, first evaluations indicate that this approach is less effective than our showcased iterative tuning.

Regarding the **Genetic Algorithm** approach, the selection of a suitable structure for the function f remains an open question. As a proof of concept we showed that even a rather simple choice of f (c.f., Equation (1)) leads already to a significant improvement of the acceptance ratio. Furthermore, fine-tuning parameters like the number of generations and population size is essential for enhancing the overall performance has not been investigated yet. Additionally, exploration of diverse mutation and crossover operations could be valuable. We hope to receive some feedback or guidance from the community to improve these aspects even further.

Furthermore, we are unsure if using a genetic algorithm is a good choice among the machine learning methods. It would be valuable to explore further machine learning approaches for finding the relative PPs.

VI. Conclusion and Future Work

The choice of a scheduler majorly affects the schedulability of a task set. Although EDF is the optimal choice for ordinary tasks, different algorithms may perform better for self-suspending tasks. In this work, we employ EDF-Like (EL) scheduling and provide two approaches, *Iterative Tuning* and *Genetic Algorithm*, to generate relative priority points for each task. The preliminary evaluation shows that the acceptance ratio of task sets can be significantly improved by tuning the relative PPs, which motivates further research on this topic. Guidance on configuration and choice of machine learning models, as well as new ideas and discussion of our proposed methods is pursued during the workshop.

We note that this paper focuses on job-level fixed priority scheduling due to the lack of a more general analysis. To apply our optimization approach, scheduling algorithms must be modeled using task parameters and a sufficient schedulability test must ensure schedulability under given task parameters. As an example, extensions of EL scheduling with multiple relative priority points for different task segments may be analyzed and optimized in future work.

Currently, the paper assumes that relative priority points Π_i are real numbers. Limiting Π to rationals or only a discrete set may speed up or simplify the optimization. However, this is not discussed or even worked out yet.

Acknowledgment

This result is part of a project (PropRT) that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 865170). This work has been supported by Deutsche Forschungsgemeinschaft (DFG), as part of Sus-Aware (Project No. 398602212).

References

[1] J. Chen, T. Hahn, R. Hoeksma, N. Megow, and G. von der Brüggen. Scheduling self-suspending tasks: New and old results. In S. Quinton, editor, *31st Euromicro Conference on Real-Time Systems, ECRTS 2019, July 9-12, 2019, Stuttgart, Germany*, volume 133 of *LIPIcs*, pages 16:1–16:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[2] M. Günzel, G. von der Brüggen, K. Chen, and J. Chen. EDF-like scheduling for self-suspending real-time tasks. In *RTSS*, 2022.

[3] H. Leontyev and J. H. Anderson. Generalized tardiness bounds for global multiprocessor scheduling. In *RTSS*, 2007.

[4] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 1973.