Self-Suspension Strikes Back

Jian-Jia Chen
Department of Computer Science
TU Dortmund University, Germany
jian-jia.chen@tu-dortmund.de

Mario Günzel
Department of Computer Science
TU Dortmund University, Germany
mario.guenzel@tu-dortmund.de

Georg von der Brüggen
Department of Computer Science
TU Dortmund University, Germany
georg.von-der-brueggen@tu-dortmund.de

I. Introduction

Self-suspension is when a task pauses its execution voluntarily, e.g., when offloading to another device or waiting for a shared resource, and has been an activate research area from 1988. The review paper by Chen et al. [1] in 2019 shows that several of these results have been flawed. In the last five years, new analytical results have mostly been limited to periodic tasks under segmented self-suspension [2], [3], and sporadic tasks under dynamic self-suspension [4]–[6]. Furthermore, the generalization of self-suspension results into different scenarios has been analyzed [7]–[9]. However, considering different release constraints (frame-based, periodic harmonic, periodic arbitrary, and sporadic) and suspension models (segmented self-suspension, and dynamic self-suspension), there have been almost no significant advancements for the most fundamental questions: (i) What is the computational complexity of scheduling self-suspending task sets? (ii) When are self-suspending task sets schedulable?

More specifically, besides exploiting periodic releases to tighten suspension-oblivious analysis for dynamic self-suspending tasks under Earliest-Deadline-First (EDF) [6], any extension of computational complexity results towards more general solutions or exploiting additional information that comes from refined models either failed under several attempts from the authors or resulted in only incomplete results. In this work, we discuss current solutions for (i) and (ii), and how they are limited to their task model. It is surprising that even a minor modification of the task model is such a challenging endeavor.

II. Task Models

We assume a set $\mathbf{T} = \{\tau_1, \ \ldots, \ \tau_n\}$ of n recurrent real-time tasks, which releases an infinite number of *task instances* (called *jobs*). Each task $\tau_i = (C_i, \ T_i, \ D_i)$ is specified by its *worst-case execution time* (WCET) C_i, its minimum inter-arrival time or *period* T_i, and its *relative deadline* D_i. The *utilization* of τ_i is $U_i = \frac{C_i}{T_i}$.

The *period* T_i is the minimum inter-arrival time between any two consecutive job releases of τ_i. A task is called (strictly) *periodic* if any two subsequent job releases are always separated by T_i, while for sporadic tasks two subsequent releases are always separated by at least T_i. A periodic task is further described by a *phase* parameter ϕ_i which indicates the time the first instance of the job is released. We assume this parameter to be 0 for all tasks and omit it for convenience. For periodic tasks, we consider two restricted, simpler scenarios. A task set is called a *harmonic* if for any two periods in the system, one is an integer multiple of the other. A task set is called *frame-based* if all tasks have the same period. A task set has *implicit deadlines* if $D_i = T_i \ \forall \tau_i \in \mathbf{T}$, *constrained deadlines* if $D_i \leq T_i \ \forall \tau_i \in \mathbf{T}$, and *arbitrary deadlines* otherwise.

Two self-suspension models are primarily studied in the literature [1], [10]. The *dynamic self-suspension* task model specifies a task τ_i as an ordinary periodic or sporadic task with an addition maximum total self-suspension time parameter S_i, and task τ_i may suspend itself at any moment before it finishes and infinitely often as long as the maximum self-suspension time S_i is not violated. The *segmented self-suspension* task model specifies tasks by an array $(C_{i,1}, S_{i,1}, C_{i,2}, S_{i,2}, ..., S_{i,m_i}, C_{i,m_i+1})$ of m_i+1 computation segments separated by m_i suspension intervals, where $C_{i,j}$ is the worst-case execution time of a computation segment and $S_{i,j}$ is the maximum length of a self-suspension interval. The hybrid self-suspension model [11] allows tradeoffs between the restrictive segmented and the flexible dynamic model but is not covered in the following discussion.

III. Computational Complexity of Exact Solutions

There are two correlated problems for task models with self-suspension. One is to design scheduling policies to generate schedules and another is to design schedulability tests to validate whether there is no deadline miss of real-time tasks. For computational complexity studies, the former one is to validate whether there exists a feasible schedule to meet all timing constraints (denoted as the feasibility problem by Chen et al. [1]) and the latter one is to validate whether there is a polynomial-time algorithm (denoted as a schedulability test) to validate whether all timing constraints are met for a specific scheduling policy. Chen et al. [1] summarized, as recapped in Table I, the computational complexity classes of the feasibility and schedulability problems to deal with self-suspension.

We note that the results in Table I consider sporadic real-time tasks with implicit deadlines. We now take a closer look for frame-based, periodic (harmonic), periodic (arbitrary), and sporadic tasks. For frame-based real-time tasks with segmented self-suspension, Chen et al. [17] showed that the feasibility problem is $\mathcal{NP}$-hard in the strong sense even when there is one suspension interval and the two computation segments are with unit execution time. This means that the other more general task models with different recurrent behavior are all $\mathcal{NP}$-hard in the strong sense.

Table I: The computational complexity classes of scheduling and schedulability analysis for sporadic self-suspending tasks with implicit deadlines, from [1].

Task Model	Feasibility	Schedulability		
		Fixed-Priority Scheduling	Dynamic-Priority Scheduling	
			Constrained Deadlines	Implicit Deadlines
segmented self-suspension	$\mathcal{NP}$-hard in the strong sense [12], [13]	co$\mathcal{NP}$-hard in the strong sense [14], [15]	co$\mathcal{NP}$-hard in the strong sense [15]	co$\mathcal{NP}$-hard in the strong sense [15]
dynamic self-suspension	unknown	(at least) $\mathcal{NP}$-hard in the weak sense [16]	co$\mathcal{NP}$-hard in the strong sense [15]	unknown

Table II: Computational complexity for different task models for the *feasibility problem*.

Release Model	Segmented Self-Suspension	Dynamic Self-Suspension		
		Implicit	Constrained	Arbitrary
frame-based	$\mathcal{NP}$-hard in the strong sense [17]	polynomial-time	unknown	unknown
periodic harmonic	$\mathcal{NP}$-hard in the strong sense, generalized from [17]	unknown	unknown	unknown
periodic arbitrary	$\mathcal{NP}$-hard in the strong sense, generalized from [17]	unknown	unknown	unknown
sporadic	$\mathcal{NP}$-hard in the strong sense [12], [13]	unknown	unknown	unknown

Table III: Computational complexity for different task models for the *schedulability test problem* under fixed-priority scheduling.

Release Model	Segmented Self-Suspension	Dynamic Self-Suspension		
		Implicit	Constrained	Arbitrary
frame-based	unknown	polynomial-time	polynomial-time	unknown
periodic harmonic	unknown	unknown	unknown	unknown
periodic arbitrary	unknown	unknown	unknown	unknown
sporadic	co$\mathcal{NP}$-hard in the strong sense [14], [15]	(at least) $\mathcal{NP}$-hard in the weak sense [16]		

For frame-based real-time tasks with dynamic self-suspension, the scheduling problem can resolved in polynomial time by always prioritizing the task with the longer $T - S_i$ the higher priority, where T is the common period of the tasks. However, all the other cases are unknown and remain open, even for frame-based real-time tasks with constrained deadlines. Table II summarizes the computational complexity for the feasibility problem to deal with self-suspension tasks.

For the computational complexity for the schedulability test problem under task-level fixed-priority scheduling, there has been no new result since the review paper [1] was published in 2019. Still, although schedulability tests for dynamic self-suspension frame-based real-time tasks have not been studied, it is easy to see that this schedulability test problem can be resolved in polynomial time for implicit deadlines or constrained deadlines. More specifically, the worst case for a job of τ_k is achieved if all higher priority tasks τ_i execute for C_i time units without being suspended. Therefore, $C_k + S_k + \sum_{\tau_i \in hp(\tau_k)} C_i$ is an exact upper bound on the worst-case response time of a task τ_k, where $hp(\tau_k)$ is the set of higher-priority tasks of τ_k. However, all the other cases for periodic tasks remain open, as summarized in Table III.

IV. Sufficient Schedulability Tests

In the previous section, we have discussed the computational complexity of determining the exact schedulability. jj:???As a generalization of testing the exact schedulability, we consider sufficient schedulability tests in this section. Sufficient schedulability tests, summarized in Table IV, use over-approximation, resulting in usually more pessimistic analyses with less computational complexity. We observe that most analytical bounds are for the scenarios (i) periodic tasks with segmented self-suspension, and (ii) sporadic tasks with dynamic self-suspension. The remaining scenarios are only sparsely examined. In the following we first detail scenarios (i) and (ii), and then discuss the sparse literature for the remaining scenarios.

For scenario (i), i.e., periodic tasks with segmented self-suspension behavior, results usually convert self-suspending jobs into a sequence of non-self-suspending jobs. The non-self-suspending jobs can for example be aggregated to be modeled

Table IV: Sufficient schedulability tests for different task models. Novel approaches in the recent five years are marked red.

Release Model	Segmented Self-Suspension	Dynamic Self-Suspension
frame-based	–	–
periodic harmonic	–	[18]
periodic arbitrary	[19]; [20][4] new: [2], [3]	new: [6]
sporadic	[21]; [22][2]	[23]–[25]; [26][1]; [27][1]; [28][3] new: [4]–[6]

[1] Although the schedulability test is stated for periodic tasks, the proof in [23] shows the formula is applicable to sporadic tasks.
[2] A fix for Section VI has been provided in [29].
[3] A fix this test has been provided in [30].
[4] Although the maximum number of execution segments is upper bounded, we classify this paper as dynamic self-suspension since there is no upper bound on individual execution and suspension segments.

as non-self-suspending tasks as done by Palencia and Harbour [19]. Alternatively, exhaustive approaches are pursued, e.g., by examining possible schedules over one hyperperiod as done by Yalcinkaya et al. [2] using UPPAAL. To the best of our knowledge, all results for non-preemptive scheduling are limited to the segmented self-suspension model. That is, there has been no successful effort to analyze non-preemptive tasks under dynamic self-suspension.

For scenario (ii), i.e., sporadic tasks with dynamic self-suspension behavior, results typically apply the following analytical approaches, as detailed in the review on self-suspending tasks [1]:

- Modelling the interfered task as suspension oblivious: The suspension of the interfered task is converted into additional execution time.
- Modelling the suspension of the interfering task as carry-in, release jitter, or blocking: The maximum number of interfering jobs is determined, and a carry-in job, release jitter or blocking term is added to account for the self-suspending behavior.

All results for scenario (ii) are designed for the preemptive task model.

Since 2019, there has only been one new analysis approach apart from scenarios (i) and (ii). Specifically, in [6] we are the first to tighten the suspension-oblivious utilization-based test under EDF by exploiting the strict periodicity of job releases. That is, besides scenarios (i) and (ii), to date there are only four results. Furthermore, it is unclear if those analyses indeed are not applicable to sporadic tasks with dynamic self-suspension or if the categorization is only due to limitations of the existing proof strategies. We conclude the following open questions:

- How can we exploit additional information that comes from a refined release model or suspension model? Especially, how can we derive tighter analytical results for frame-based and periodic harmonic task sets?
- Are the analyses apart from scenarios (i) and (ii) using information of their respective task model or is their categorization in the task model only due to a lack of more general proof?
- How can we design schedulability tests for dynamic self-suspending tasks under non-preemptive scheduling?

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).

References

[1] J.-J. Chen, G. Nelissen, W.-H. Huang, M. Yang, B. Brandenburg, K. Bletsas, C. Liu, P. Richard, F. Ridouard, Neil, Audsley, R. Rajkumar, D. de Niz, and G. von der Brüggen, "Many suspensions, many problems: A review of self-suspending tasks in real-time systems," *Real-Time Systems*, 2018, https://link.springer.com/article/10.1007/s11241-018-9316-9.

[2] B. Yalcinkaya, M. Nasri, and B. B. Brandenburg, "An exact schedulability test for non-preemptive self-suspending real-time tasks," in *DATE*. IEEE, 2019, pp. 1228–1233.

[3] D. Casini, P. Pazzaglia, A. Biondi, and M. D. Natale, "Optimized partitioning and priority assignment of real-time applications on heterogeneous platforms with hardware acceleration," *J. Syst. Archit.*, vol. 124, p. 102416, 2022.

[4] M. Günzel, G. von der Brüggen, K. Chen, and J. Chen, "Edf-like scheduling for self-suspending real-time tasks," in *RTSS*. IEEE, 2022, pp. 172–184.

[5] M. Günzel, N. Ueter, and J. Chen, "Suspension-aware fixed-priority schedulability test with arbitrary deadlines and arrival curves," in *RTSS*. IEEE, 2021, pp. 418–430.

[6] M. Günzel, G. von der Brüggen, and J. Chen, "Suspension-aware earliest-deadline-first scheduling analysis," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 39, no. 11, pp. 4205–4216, 2020.

[7] F. Aromolo, A. Biondi, G. Nelissen, and G. C. Buttazzo, "Event-driven delay-induced tasks: Model, analysis, and applications," in *RTAS*. IEEE, 2021, pp. 53–65.

[8] K. Chen, M. Günzel, B. Jablkowski, M. Buschhoff, and J. Chen, "Unikernel-based real-time virtualization under deferrable servers: Analysis and realization (artifact)," *Dagstuhl Artifacts Ser.*, vol. 8, no. 1, pp. 02:1–02:2, 2022.

[9] N. Ueter, M. Günzel, G. von der Brüggen, and J. Chen, "Hard real-time stationary gang-scheduling," in *ECRTS*, ser. LIPIcs, vol. 196. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, pp. 10:1–10:19.

[10] J. Chen, G. von der Brüggen, W. Huang, and C. Liu, "State of the art for scheduling and analyzing self-suspending sporadic real-time tasks," in *23rd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2017, Hsinchu, Taiwan, August 16-18, 2017*. IEEE Computer Society, 2017, pp. 1–10. [Online]. Available: https://doi.org/10.1109/RTCSA.2017.8046321

[11] G. von der Brüggen, W.-H. Huang, and J.-J. Chen, "Hybrid self-suspension models in real-time embedded systems," in *International Conference on Real-Time Computing Systems and Applications (RTCSA)*, 2017.

[12] P. Richard, "On the complexity of scheduling real-time tasks with self-suspensions on one processor," in *Proceedings. 15th Euromicro Conference onReal-Time Systems, (ECRTS)*, July 2003, pp. 187–194.

[13] F. Ridouard, P. Richard, and F. Cottet, "Negative results for scheduling independent hard real-time tasks with self-suspensions," in *RTSS*, 2004, pp. 47–56.

[14] M. Mohaqeqi, P. Ekberg, and W. Yi, "On fixed-priority schedulability analysis of sporadic tasks with self-suspension," in *Proceedings of the 24th International Conference on Real-Time Networks and Systems, RTNS*, 2016, pp. 109–118.

[15] J.-J. Chen, "Computational complexity and speedup factors analyses for self-suspending tasks," in *Real-Time Systems Symposium (RTSS)*, 2016, pp. 327–338.

[16] P. Ekberg and W. Yi, "Fixed-priority schedulability of sporadic tasks on uniprocessors is np-hard," in *2017 IEEE Real-Time Systems Symposium, RTSS 2017, Paris, France, December 5-8, 2017*, 2017, pp. 139–146. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/RTSS.2017.00020

[17] J. Chen, T. Hahn, R. Hoeksma, N. Megow, and G. von der Brüggen, "Scheduling self-suspending tasks: New and old results," in *31st Euromicro Conference on Real-Time Systems, ECRTS*, S. Quinton, Ed., vol. 133, 2019, pp. 16:1–16:23. [Online]. Available: https://doi.org/10.4230/LIPIcs.ECRTS.2019.16

[18] C. Liu, J.-J. Chen, L. He, and Y. Gu, "Analysis techniques for supporting harmonic real-time tasks with suspensions," in *26th Euromicro Conference on Real-Time Systems, ECRTS 2014, Madrid, Spain, July 8-11, 2014*, 2014, pp. 201–210.

[19] J. C. Palencia and M. G. Harbour, "Schedulability analysis for tasks with static and dynamic offsets," in *Proceedings of the 19th IEEE Real-Time Systems Symposium (RTSS)*, 1998, pp. 26–37.

[20] Z. Dong, C. Liu, S. Bateni, K. Chen, J. Chen, G. von der Brüggen, and J. Shi, "Shared-resource-centric limited preemptive scheduling: A comprehensive study of suspension-based partitioning approaches," in *RTAS*. IEEE Computer Society, 2018, pp. 164–176.

[21] D. Casini, A. Biondi, G. Nelissen, and G. C. Buttazzo, "Partitioned fixed-priority scheduling of parallel tasks without preemptions," in *RTSS*. IEEE Computer Society, 2018, pp. 421–433.

[22] G. Nelissen, J. Fonseca, G. Raravi, and V. Nélis, "Timing Analysis of Fixed Priority Self-Suspending Sporadic Tasks," in *Euromicro Conference on Real-Time Systems (ECRTS)*, 2015, pp. 80–89.

[23] J.-J. Chen, G. Nelissen, and W.-H. Huang, "A unifying response time analysis framework for dynamic self-suspending tasks," in *Euromicro Conference on Real-Time Systems (ECRTS)*, 2016.

[24] C. Liu and J. Chen, "Bursty-interference analysis techniques for analyzing complex real-time task models," in *Real-Time Systems Symposium (RTSS)*, 2014, pp. 173–183.

[25] J.-J. Chen, W.-H. Huang, and C. Liu, "k2U: A general framework from k-point effective schedulability analysis to utilization-based tests," in *Real-Time Systems Symposium (RTSS)*, 2015, pp. 107–118.

[26] J. W. S. Liu, *Real-Time Systems*, 1st ed. Prentice Hall PTR, 2000.

[27] W.-H. Huang, J.-J. Chen, H. Zhou, and C. Liu, "PASS: Priority assignment of real-time tasks with dynamic suspending behavior under fixed-priority scheduling," in *Proceedings of the 52nd Annual Design Automation Conference (DAC)*, 2015, pp. 154:1–154:6.

[28] C. Liu and J. H. Anderson, "Suspension-aware analysis for hard real-time multiprocessor scheduling," in *ECRTS*. IEEE Computer Society, 2013, pp. 271–281.

[29] G. Nelissen, J. Fonseca, G. Raravi, and V. Nélis, "Errata: Timing analysis of fixed priority self-suspending sporadic tasks," CISTER-Research Centre in Realtime and Embedded Computing Systems, Tech. Rep., 2017.

[30] C. Liu and J. H. Anderson, "Erratum to "suspension-aware analysis for hard real-time multiprocessor scheduling"," 2015, https://cs.unc.edu/~anderson/papers/ecrts13e_erratum.pdf.