Work-in-Progress: Evaluation Framework for Self-Suspending Schedulability Tests

Mario Gnzel, Harun Teper, Kuan-Hsun Chen, Georg von der Brggen and Jian-Jia Chen
TU Dortmund, Department of Computer Science, Dortmund, Germany

Citation: 10.1109/RTSS52674.2021.00058

BibTeX:

```
@inproceedings{guenzel21rtss_evalframework,
  author={Mario Gnzel, Harun Teper, Kuan-Hsun Chen, Georg von der Brggen, Jian-Jia Chen},
  booktitle={42nd IEEE Real-Time Systems Symposium (RTSS)},
  title={Work-in-Progress: Evaluation Framework for Self-Suspending Schedulability Tests},
  year={2021},
  volume={},
  number={},
  pages={},
  doi={}
}
```

Work-in-Progress: Evaluation Framework for Self-Suspending Schedulability Tests

Mario Günzel, Harun Teper, Kuan-Hsun Chen, Georg von der Brüggen, Jian-Jia Chen
TU Dortmund University, Dortmund, Germany
{mario.guenzel, harun.teper, kuan-hsun.chen, georg.von-der-brueggen, jian-jia.chen}@tu-dortmund.de

***Abstract*—Numerical simulations often play an important role when evaluating and comparing the performance of schedulability tests, as they allow to empirically demonstrate their applicability using synthesized task sets under various configurations. In order to provide a fair comparison of various schedulability tests, von der Brüggen et al. presented the first version of an evaluation framework for self-suspending task sets. In this work-in-progress, we further enhance the framework by providing more features to ease the use, e.g., Python 3 support, an improved GUI, multiprocessing, Gurobi optimization, and external task evaluation. In addition, we integrate the state-of-the-arts we are aware of into the framework. Moreover, the documentation is improved significantly to simplify the application in further research and development. To the best of our knowledge, the framework contains all suspension-aware schedulability tests for uniprocessor systems and we aim to keep it up-to-date.**

I. Introduction

In real-time systems, tasks need to comply with timing requirements. In particular, each task instance (called job) has to meet its deadline to guarantee correct behavior or prevent catastrophic outcomes. To provide such timing guarantees, various schedulability tests were developed and applied for assessing a given task set. In particular, if such tests return *True*, then the task set is *schedulable*, i.e., each job meets its deadline even in the worst case.

When a task is eligible to leave the processor and continue being executed after a certain time interval, such an interval can be utilized by other jobs. Such a behavior is known as *self-suspension* and often occurs in different scenarios like GPU-Offloading or Multiprocessor synchronization [5]. However, self-suspension behavior also complicates the schedulability tests. In order to mitigate the unnecessary pessimism, several advanced techniques have been developed to take various self-suspension models into consideration.

In WATERS 2019, an evaluation framework has been presented by von der Brüggen et al. [26]. It provides an easy-to-use framework for evaluating the performance of various schedulability tests, based on synthesized task sets. However, only a few schedulability tests were implemented. In this work, we enhance the framework by implementing the state-of-the-arts we are aware of and provide additional features. The functionality of the framework is threefold: First, it can be used to generate self-suspending task sets, using existing synthesis approaches [7]. Second, it provides schedulability tests that can be applied to the generated task sets or external task sets, which are loaded into the framework. Third, it illustrates the performance of schedulability tests by providing plots of acceptance ratio, i.e., the amount of task sets that are deemed schedulable by the test divided by the total amount of task sets, over task utilizations.

For categorization, the schedulability tests can be distinguished according to self-suspension models as defined in [5]:

- Segmented self-suspension: The structure of execution and suspension segments is predefined for each task.
- Dynamic self-suspension: Only upper bounds on *total* execution time and *total* suspension time are provided. The segmentation of jobs may change within one task.
- Hybrid self-suspension: Different segmentations are predefined for one task. Each job follows one of these segmentations.

Moreover, there are different deadline constraints:

- Implicit deadline: Deadline and period/minimal inter-arrival time of each task coincide.
- Constrained deadline: The relative deadline of each task is less than or equal to its period/minimal inter-arrival time.
- Arbitrary deadline: There are no constraints on the deadlines, i.e., the deadline can be higher than, equal to, or lower than the period/minimal inter-arrival time.

In the framework, we also classify the schedulability tests, according to the targeted self-suspension models. However, as the tests are not always compatible with constrained deadlines or arbitrarily deadlines, the framework only considers the implicit-deadline model for generating automatic comparisons. Furthermore, since there is no standardized way of constructing constrained- and arbitrary-deadline task sets yet in the literature, the users are recommended to adjust the program for evaluating constrained- or arbitrary-deadline task systems if necessary. The user can easily configure it as different deadline models, so we leave that out.

In a nutshell, the schedulability tests from 18 relevant papers are realized in the current framework. The enhanced framework does not only support the evaluation and comparison of schedulability tests, but also integrates the Gurobi optimization solver [9] to supply tests which use linear programming. The framework is published on Github at [18] and there is an ongoing effort to keep the framework up to date.

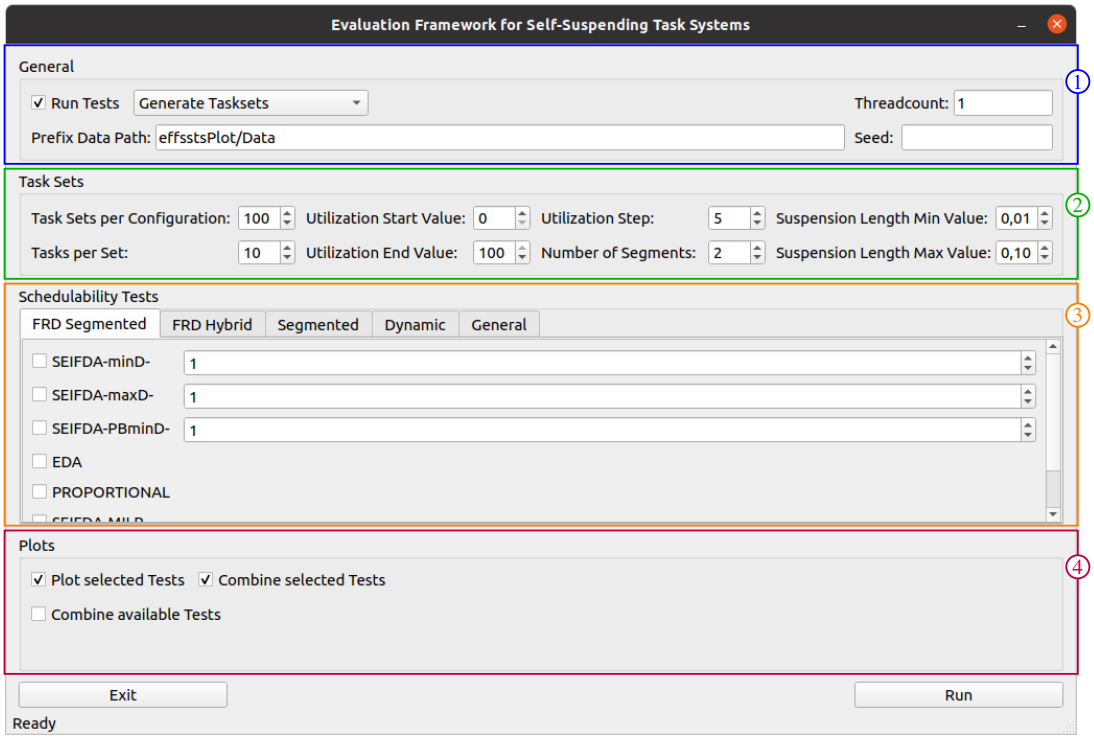

Figure 1: Overview of the enhanced framework.

II. Framework Description

In this section, we first introduce the features of the current framework in detail. In Figure 1 the GUI of our framework is presented. Besides the general setting of the framework (①), the framework consists of three components (②, ③ and ④):

1) A *task set generator* to provide periodic/sporadic task sets for evaluation.
2) The *schedulability test evaluator* applies a choice of schedulability tests to the generated task sets.
3) The *plotting tool configurator* illustrates the performance of the schedulability tests.

Please note that the following description is explained in a sequential manner to demonstrate the usage from beginning to end. However, these components can also be used independently for different scenarios (see Section III).

General Setting: In part ①, the user selects the option to either generate task sets in the framework or to load an external file that contains the task set data. The task sets can be reused by saving them after evaluation and reloading them at a later point in time. Additionally, the number of threads used to evaluate the task sets in parallel can be set. The generated or loaded task sets are evenly distributed to the selected number of threads and evaluated in parallel. The evaluation of a large number of task sets can therefore be done in a fraction of the time compared to the case without multiprocessing. Furthermore, the seed for the random number generator can be set, so that the results of a previous run can be reproduced.

Task Set Configurations: The parameters of synthesized task sets can be configured in part ②. The task sets are generated using the integrated UUnifast algorithm [1] and can be configured by using the parameters that are provided in the framework. In particular, these parameters are: the number of task sets, the number of tasks per set, the number of suspension segments, the utilization values to evaluate, and the suspension parameters for the tasks. At first, the utilization of each task is drawn randomly according to UUnifast [1]. Afterwards, the period and deadline are drawn from a log-uniform distribution, where the default is over two orders of magnitude, i.e., [1,100]. Depending on the considered suspension models, the suspension time of each task is drawn accordingly[1].

Test Selection: After configuring the parameters, the user can select a set of schedulability tests in part ③ to evaluate the task sets. All currently available schedulability tests are presented in Table I. Please note that some schedulability tests can set custom parameters, e.g., SEIFDA-based approaches [25]. After pressing the `Run` button at the bottom, each generated task set will be tested by the selected schedulability tests iteratively. If a tested task set is deemed schedulable, the considered test will return *True*. After checking all task sets, the framework counts the number of *True*s, and calculates the percentage of task sets that are deemed schedulable by each test. It is worth to note that the framework can also support tests which require linear programming solvers, e.g., MILP-ReleaseJitter, by using the Gurobi optimization solver [9].

Plotting Format: The results of the schedulability analysis can then be plotted using an integrated tool of the framework. The user can determine the format of the plots in part ④,

[1]Details of implementation can be found in the readme file on [18].

Year	Papers	Methods	Suspension	Deadline	New
	Baseline approaches	Suspension as computation, i.e., SCEDF, SCRM, SUSPOBL	—	(up to tests)	
2000	Liu [16, pp. 164–165]	SUSPBLOCK (proof in [4])	Dynamic	Implicit	✓
2014	Liu et al. [17]	PROPORTIONAL	Segmented	Implicit	
2014	Chen and Liu [3]	EDA	Segmented	Implicit	
2014	Liu and Chen [15]	Idv-Burst-RM	Dynamic	Implicit	✓
2015	Liu and Anderson et al. [14][1]	WLAEDF	Dynamic	Implicit	✓
2015	Huang et al. [13]	PASS-OPA	Dynamic	Constrained	
2015	Nelissen et al. [20], [21], Biondi et al. [2][2]	MILP-ReleaseJitter	Segmented	Constrained	✓
2016	Dong and Liu [6][1]	UDLEDF	Dynamic	Implicit	✓
2016	Mohaqeqi et al. [19]	SRSR	Segmented	Implicit	✓
2016	von der Brüggen et al. [25]	SEIFDA-minD, SEIFDA-maxD, SEIFDA-PBminD, SEIFDA-MILP, NC	Segmented	Implicit	
2016	Peng and Fisher [22]	GMFPA	Segmented	Arbitrary	✓
2016	Huang and Chen [12]	EDAGMF-OPA	Segmented	Constrained	✓
2016	Chen et al. [4]	UNIFRAMEWORK, SUSPOBL, SUSPJIT, SUSPBLOCK	Dynamic	Constrained	✓
2017	von der Brüggen et al. [24]	Oblivious-IUB, Clairvoyant-SSSD, Oblivious-MP, Clairvoyant-PDAB	Hybrid	Implicit	
2018	Schönberger et al. [23][3]	SCAIR-RM, SCAIR-OPA	Dynamic	Constrained	
2019	Yalcinkaya et al. [27]	UPPAAL	Dynamic	Implicit	✓
2020	Günzel et al. [10]	RSS, RTEDF	Dynamic	Implicit	✓

[1] Uniprocessor version is presented in [10].
[2] Originally proposed in Section VI in [20], revised in [21], implemented in [2] with open source.
[3] Originally proposed in [11] and revised in [23].

Table I: Schedulability tests included in the framework.

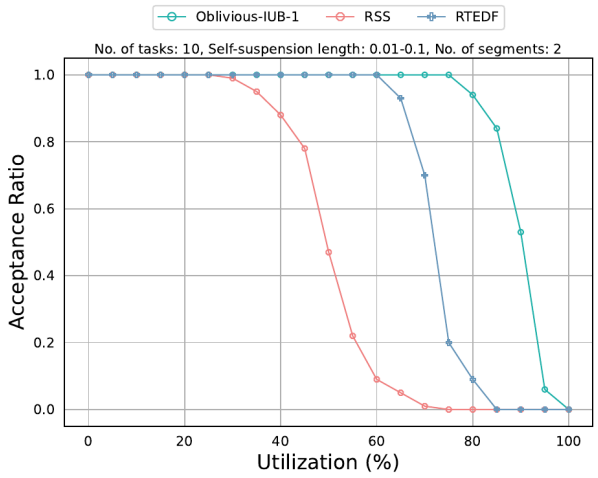

Figure 2: Plot provided by the framework schedulability analysis with three tests and 100 tasks per set.

either plot each selected test by itself, or combine the plots of all selected tests, so that they can be directly compared against each other. A resulting exemplary plot is shown in Figure 2. The framework relies on a plotting library, namely `matplotlib` in Python. The raw data of the evaluation can also be post-processed by other plotting schemes.

III. Research Applications

The framework now can also be applied in different research scenarios. In the following, we detail three scenarios, in which the framework can be used:

- **Scenario 1:** Evaluate and compare the performance of an own schedulability test.
- **Scenario 2:** Analyze own task sets with already known schedulability tests.
- **Scenario 3:** Utilize already known schedulability tests in an own python program.

Scenario 1: Additional schedulability tests can be integrated into the framework. This involves two steps: First, the analysis needs to be implemented in Python. Alternatively for C++-algorithms, a pre-built binary can also be executed from a Python script. Second, the additional test needs to be integrated into the framework infrastructure. The test has to be integrated into the testing component of the framework and it has to be made available in part ③ of the GUI. In particular, several things should be added into `effsstsMain.py`, e.g., add an entry to the GUI and an additional case to the `switchTest` method. After integration, it is possible to evaluate the test using the provided task generator and plotting tool.

Scenario 2: Custom task sets can be loaded into the framework to evaluate them using the provided schedulability tests. For this scenario the *task set generator* component (②) is replaced, but the other two components, i.e., ③ and ④ are utilized. In this regard, we additionally provide a script `SaveTaskSet` to convert task sets into the required format. This script will guide the user to create a serialized file for custom task sets, so that they can be loaded into the framework properly. To this end, the general setup ① also has to be configured accordingly to load the own task sets, i.e., ticking "Load Taskset" and enter the name of the saved task set. In fact, this feature is particularly useful for a large number of task sets which need to be evaluated all at once.

Scenario 3: The realized schedulability tests in the framework can also be imported into other programs, which can be done by a standard import statement in Python. The names for the schedulability tests are identical as the ones in part ③ of the GUI. After the import, it is possible to evaluate custom external task sets, as long as the required arguments are given correctly. Please note that the return value of every schedulability test is always in a Boolean expression.

IV. Future Work

In this work, we enhance an evaluation framework for self-suspending schedulability tests, by providing more complete features to ease the use for the users. In addition, the enhanced framework has included all suspension-aware schedulability tests developed for uniprocessor systems, and there is an ongoing effort to keep the framework up to date.

The current version of the framework is focused on schedulability test comparison. For task creation, we only consider workload/task constructions based on UUnifast [1], which have been widely used in the literature for evaluating self-suspending task systems. The current approach generates the self-suspension time based on a randomized percentage of the slack of a task (i.e., a randomized percentage of the relative deadline minus the worst-case execution time). This approach is sensitive to the number of tasks and the target utilization. In future work, we plan to implement a collection of benchmarks for task set creation to cover a large variety of task set characteristics and explore the potential pitfalls of different configurations. In particular, we aim to integrate an extended version of the Dirichlet Rescale (DRS) [8] task generation algorithm for self-suspending task systems. Moreover, we would like to extend our work to multiprocessor scenarios. In this regard, we aim to provide a collection of benchmarks for multiprocessor self-suspension task set generation and analysis for those task sets.

V. Acknowledgement

We would like to acknowledge the support of all the authors who helped to extend the framework. In particular, we would like to thank Bo Peng, Morteza Mohaqeqi, Alessandro Biondi and Beyazit Yalcinkaya for providing the source code and additional information of their work. Furthermore, we thank Milad Nayebi, Tai-Jung Chang and Junjie Shi who were part of the team that developed the initial version.

This work has been supported by Deutsche Forschungsgemeinschaft (DFG), as part of Sus-Aware (Project No. 398602212), and by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Projektnummer 124020371 – SFB 876.

References

[1] E. Bini and G. C. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005.

[2] A. Biondi, A. Balsini, M. Pagani, E. Rossi, M. Marinoni, and G. Buttazzo. A framework for supporting real-time applications on dynamic reconfigurable fpgas. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 1–12, 11 2016.

[3] J.-J. Chen and C. Liu. Fixed-relative-deadline scheduling of hard real-time tasks with self-suspensions. In *Real-Time Systems Symposium (RTSS)*, pages 149–160, 2014. **A typo in the schedulability test in Theorem 3 was identified on 13, May, 2015**.

[4] 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.

[5] J.-J. Chen, G. Nelissen, W.-H. Huang, M. Yang, B. B. Brandenburg, K. Bletsas, C. Liu, P. Richard, F. Ridouard, N. C. 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*, 55(1):144–207, 2019.

[6] Z. Dong and C. Liu. Closing the loop for the selective conversion approach: A utilization-based test for hard real-time suspending task systems. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 339–350, 2016.

[7] P. Emberson, R. Stafford, and R. I. Davis. Techniques for the synthesis of multiprocessor tasksets. In *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2010)*, pages 6–11, 2010.

[8] D. Griffin, I. Bate, and R. I. Davis. Generating utilization vectors for the systematic evaluation of schedulability tests. In *RTSS*, pages 76–88. IEEE, 2020.

[9] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2021.

[10] M. Günzel, G. von der Brüggen, and J.-J. Chen. Suspension-aware earliest-deadline-first scheduling analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(11):4205–4216, 2020.

[11] W.-H. Huang and J.-J. Chen. Schedulability and priority assignment for multi-segment self-suspending real-time tasks under fixed-priority scheduling. Technical report, Technical University of Dortmund, Dortmund, Germany, 2015.

[12] W.-H. Huang and J.-J. Chen. Self-suspension real-time tasks under fixed-relative-deadline fixed-priority scheduling. In *Design, Automation, and Test in Europe (DATE)*, pages 1078–1083, 2016.

[13] 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)*, pages 154:1–154:6, 2015.

[14] 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.

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

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

[17] W. Liu, J.-J. Chen, A. Toma, T.-W. Kuo, and Q. Deng. Computation offloading by using timing unreliable components in real-time systems. In *Design Automation Conference (DAC)*, volume 39:1 – 39:6, 2014.

[18] T. D. LS12. Evaluation framework for self-suspending task systems. https://github.com/tu-dortmund-ls12-rt/SSSEvaluation, 2021.

[19] 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*, pages 109–118, 2016.

[20] 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)*, pages 80–89, 2015.

[21] G. Nelissen, J. Fonseca, G. Raravi, and V. Nélis. Errata: Timing analysis of fixed priority self-suspending sporadic tasks. Technical Report CISTER-TR-170205, CISTER, ISEP, INESC-TEC, 2017.

[22] B. Peng and N. Fisher. Parameter adaptation for generalized multiframe tasks and applications to self-suspending tasks. In *International Conference on Real-Time Computing Systems and Applications*, 2016.

[23] L. Schönberger, W.-H. Huang, G. von der Brüggen, K.-H. Chen, and J.-J. Chen. Schedulability analysis and priority assignment for segmented self-suspending tasks. In *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 157–167, 2018.

[24] 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.

[25] G. von der Brüggen, W.-H. Huang, J.-J. Chen, and C. Liu. Uniprocessor scheduling strategies for self-suspending task systems. In *International Conference on Real-Time Networks and Systems*, RTNS '16, pages 119–128, 2016.

[26] G. von der Brüggen, M. Nayebi, J. Shi, K.-H. Chen, and J.-J. Chen. Evaluation framework for self-suspending task systems. In *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2019.

[27] B. Yalcinkaya, M. Nasri, and B. Brandenburg. An exact schedulability test for non-preemptive self-suspending real-time tasks. pages 1228–1233, 03 2019.