On Strong and Weak Sustainability, with an Application to Self-Suspending Real-Time Tasks

Felipe Cerqueira[1], Geoffrey Nelissen[2], and Björn B. Brandenburg[1]

1 **Max Planck Institute for Software Systems, Kaiserslautern, Germany**
2 **CISTER/INESC TEC, ISEP, Polytechnic Institute of Porto, Portugal**

Abstract

Motivated by an apparent contradiction on whether some scheduling policies are sustainable or not, we revisit the topic of sustainability in real-time scheduling and argue that the existing definitions of sustainability could be further clarified and generalized. After proposing a formal, generic sustainability theory, we relax the existing notion of (strongly-)sustainable scheduling policy to provide a new classification called weak sustainability, which enables less pessimistic schedulability analyses for policies that were deemed not (strongly-)sustainable in the past. As a proof of concept, and to better understand a model for which many mistakes were found in the literature [9], we study weak sustainability in the context of dynamic self-suspending tasks, where we formalize a generic suspension model in a proof assistant and provide a mechanized proof that any JLFP scheduling policy is weakly-sustainable with respect to job costs and variable suspension times.

1998 ACM Subject Classification Real-time systems, Real-time schedulability

Keywords and phrases Real-time Scheduling, Sustainability, Self-Suspending Tasks, Mechanized Proofs

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2018.

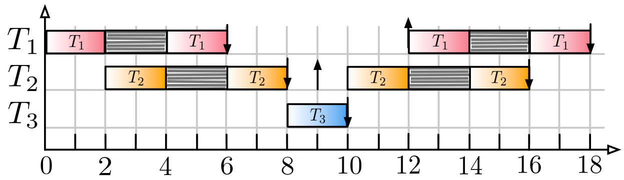
1 What Really Is Sustainability?

Since the seminal paper by Liu and Layland [14], the analysis and certification of real-time systems has often relied on the fundamental notion of sustainability, which at a high level expresses the idea that "if a system is proven to be safe under extreme conditions, then it will remain safe if the conditions improve at runtime".

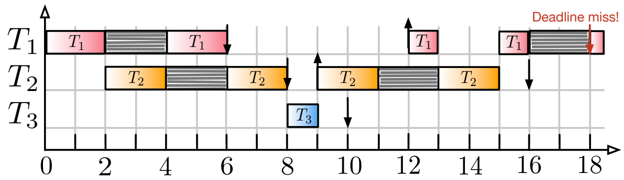
One common application of this principle is to determine the schedulability of the system by identifying worst-case scenarios. For example, any schedulability analysis for uniprocessor fixed-priority (FP) scheduling of sporadic tasks [15] that assumes that jobs execute for their worst-case execution time (WCET) or arrive at maximum rate, exploits the fact that the FP scheduling policy for sporadic tasks is sustainable, *i.e.*, it guarantees that "having better job parameters (namely, larger inter-arrival times or smaller execution times) at runtime does not cause any deadline miss".

While precursors to this concept were already identified and proven in earlier papers [12, 11], the general concept of a sustainable scheduling policy was first formalized by Burns and Baruah in 2008 [7] and later refined by Baker and Baruah in 2009 [3]. Although the definition by Baker and Baruah is more rigorous than the original definition, we argue in this paper that there is still a need for improvement in terms of clarity and precision.

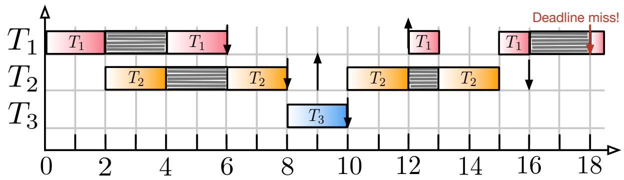
To support our claim, in §1.1 and §1.2 we present an example in the context of uniprocessor scheduling with self-suspending tasks [16], where we show a scheduling policy that can be interpreted as both *sustainable* and *not sustainable* with respect to job execution times (also



(a) EDF schedule of the original job set $\mathcal{J}$. Assume that no jobs are released after time 18.



(b) Scheduling anomaly generated from job set $\mathcal{J}$, where reducing the cost of task T_3's job yields a job set $\mathcal{J}_{better}$ in which task T_1 misses a deadline at time 18.



(c) Alternative job set $\mathcal{J}_{susp}$ with original costs and shorter suspensions that is as hard to schedule for task T_1 as $\mathcal{J}_{better}$.

Figure 1 (adapted from [2]) The figure above depicts three schedules under the segmented suspension model. It shows that the impact of the lack of sustainability tightly depends on the considered task model. Despite the anomaly shown in schedule (b), there exists a harder schedule (c) with no anomaly that still incurs a deadline miss. If we assume that the task model allows variable suspension times, then any schedulability analysis that covers all these different scenarios would never claim the job set in (c) (and thus the task set) to be schedulable, regardless of the anomaly present in schedule (b).

called job costs hereafter). Both claims are correct according to the existing definitions of sustainability and only depend on varying interpretations by the reader. This example shows that, despite being a well-established concept, the theory of sustainability needs further clarification and formalization.

1.1 Uniprocessor EDF Scheduling with Self-Suspensions is Not Sustainable With Respect to Job Costs

Consider uniprocessor earliest-deadline-first (EDF) scheduling of self-suspending tasks under the segmented suspension model. Self-suspending tasks are used to model workloads that may have their execution suspended at given times, for example, to perform remote operations on co-processors, acquire locks, wait for data, or synchronize with other tasks. The segmented self-suspending task model can be formalized as follows.

▶ **Definition 1** (Sporadic Task Model with Segmented Self-Suspensions). *Let τ be any task set and let $\mathcal{J}$ be any job set generated by τ. Each task $T_i \in \tau$ is defined by a period p_i, deadline d_i and a sequence of execution and suspension segments $S_i = [e_i^1, s_i^1, e_i^2, s_i^2, \ldots, e_i^n]$. These task parameters encode the constraints that any two jobs generated by T_i must be separated by a minimum inter-arrival time p_i. Each job released by T_i must finish its execution by a* *relative deadline d_i, and alternates between execution and suspension segments as defined by the sequence S_i. The execution time of the k-th execution segment of job j is upper-bounded by e_i^k, and the suspension time of its k-th suspension segment is upper-bounded by s_i^k.*

Next, let us recall the definition of sustainable policy as proposed by Burns and Baruah [7].

▶ **Definition** 2 (Sustainable Policy – original definition from [7]).
A scheduling policy and/or a schedulability test for a scheduling policy is sustainable if any system deemed schedulable by the schedulability test remains schedulable when the parameters of one or more individual tasks are changed in any, some, or all of the following ways: (i) decreased execution requirements; (ii) larger periods; (iii) smaller jitter; and (iv) larger relative deadlines. ◀

As explained by Burns and Baruah [7], the interpretation of Definition 2 for scheduling policies concerns the values of job parameters at runtime: *"[...] a scheduling policy that guarantees to retain schedulability if actual execution requirements during run-time are smaller than specified WCET's, and if actual jitter is smaller than the specified maximum jitters, would be said to be sustainable with respect to WCET's and jitter"*.

Thus, in order to show that a scheduling policy is not sustainable with respect to execution requirements (*i.e.*, job costs), we must find a counterexample that shows a job set $\mathcal{J}$ that is schedulable under that policy, along with a job set $\mathcal{J}_{better}$ with lower or equal job execution times that is not schedulable under the same policy.

Fig. 1 depicts such a counterexample for uniprocessor EDF scheduling on the segmented self-suspending task model, adapted from prior work by Abdeddaïm and Masson [2]. In Fig. 1-(a), we can observe the original EDF schedule of three tasks T_1, T_2 and T_3, which contains no deadline misses. Next, by reducing the cost of T_3's job by 1 time unit as shown in Fig. 1-(b), the different interleaving of suspension times during [13, 16) increases the interference incurred by task T_1, causing a deadline miss at time 18.

This counterexample, which is simple enough to make the claim non-disputable, proves that, according to Definition 2, EDF scheduling under the segmented suspension model is *not sustainable* with respect to job costs.

1.2 Uniprocessor EDF Scheduling with Self-Suspensions is Sustainable With Respect to Job Costs

Consider the same platform, task model and scheduling policy as in §1.2, and recall the definition of sustainable policy proposed by Baker and Baruah [3].

▶ **Definition** 3 (Sustainable Policy – original definition from [3]).
Let A denote a scheduling policy. Let τ denote any sporadic task system that is A-schedulable. Let $\mathcal{J}$ denote a collection of jobs generated by τ. Scheduling policy A is said to be sustainable if and only if A meets all deadlines when scheduling any collection of jobs obtained from $\mathcal{J}$ by changing the parameters of one or more individual jobs in any, some, or all of the following ways: (i) decreased execution requirements; (ii) larger relative deadlines; and (iii) later arrival times with the restriction that successive jobs of any task $T_i \in \tau$ arrive at least p_i time units apart. ◀

Definition 3 is similar to Definition 2, except that it explicitly makes the difference between the notion of jobs and tasks. It requires the job set $\mathcal{J}$ with original parameters to be generated by a task set τ that is A-schedulable, *i.e.*, all job sets generated by τ have no deadline misses. However, note that the modified job set (which we call $\mathcal{J}_{better}$) does not have to be generated by τ.

Now, we must check whether the counterexample in Fig. 1 is still valid. At a first glance, the job sets $\mathcal{J}$ and $\mathcal{J}_{better}$ depicted in Fig. 1-(a) and Fig. 1-(b) seem to prove that uniprocessor EDF scheduling with segmented self-suspending tasks is *not sustainable* with respect to job costs, according to Definition 3. After all, we can assume that job set $\mathcal{J}$ is generated for instance by some task set $\tau = \{(p_1 = 12, d_1 = 6, S_1 = [2,2,2]), (p_2 = 9, d_3 = 7, S_2 = [2,2,2]), (p_3 = 10, d_3 = 10, S_3 = [2])\}$ where, for each task $T_i \in \tau$, the parameters p_i, d_i, S_i denote, respectively, the minimum inter-arrival time of T_i, the deadline of T_i, and the sequence of execution and suspension segments of T_i.

However, let us consider the alternative job set $\mathcal{J}_{susp}$ in Fig. 1-(c), in which the job of task T_3 has the original cost of 2 time units, and the suspension time of the second job of task T_2 is reduced by 1 time unit. Clearly, $\mathcal{J}_{susp}$ can be generated by task set τ, since the job costs are the same as in $\mathcal{J}$ and the suspension segments are no larger than those in $\mathcal{J}$, which is allowed by the segmented suspension model. Moreover, we can observe that in the schedule of $\mathcal{J}_{susp}$, task T_1 again misses a deadline at time 18.

Since job set $\mathcal{J}_{susp}$ generated by τ is not schedulable, it is clear that τ does not satisfy the assumption of being A-schedulable required by Definition 3. Therefore, job sets $\mathcal{J}$ and $\mathcal{J}_{better}$ in Fig. 1 are not a valid counterexample for establishing that the policy is not sustainable. Since the counterexample is not valid, what can we really say about the sustainability of this policy? Why do the two definitions disagree?

One aspect that is implicit but unclear in both definitions is whether all job parameters other than the sustainable parameter (*i.e.*, job costs) must remain constant. In fact, as shown in $\mathcal{J}_{susp}$ from Fig. 1-(c), in some cases we can vary the other parameters (*i.e.*, job suspension times) to compensate the increase in interference that would otherwise cause the scheduling anomaly. Since this parameter variation is allowed by the task constraints, this suggests that a task set that is schedulable for any possible value of its job suspension times may in effect be resilient to scheduling anomalies on job costs, even though individual schedulable job sets are not.

In fact, by constructing job sets similar to $\mathcal{J}_{susp}$ in Fig. 1-(c), we provide a *mechanized* proof (*i.e.*, a proof that is verified by the Coq proof assistant) in §4 that establishes that uniprocessor job-level fixed priority (JLFP) scheduling of sporadic tasks under the dynamic suspension model is, what we later define as, *weakly-sustainable* with respect to job costs and *variable suspension times*.

Note that this result does not make the counterexample of Abdeddaïm and Masson incorrect. Their result is simply based on a different interpretation of sustainability where nothing but the job parameter under consideration for the sustainability property can vary between the compared schedules; thus, the results presented in §1.1 and §4 are both correct. In §3, we will complement the existing sustainability theory with the notions of *strong* and *weak* sustainability to distinguish those contradictory but correct interpretations of sustainability.

1.3 This Paper

The seemingly contradictory results in §1.1 and §1.2 suggest the need for clarification in the definitions of sustainability, which are currently restricted to the standard sporadic task model and are not precise with respect to how parameters can vary across the original and modified job sets $\mathcal{J}$ and $\mathcal{J}_{better}$.

We believe that the solution to this problem lies in formalizing the abstract concepts of *real-time scheduling meta-theory* such as "job and task parameters" in a rigorous way, so that the different notions of sustainability can be stated precisely. Additionally, this approach allows transcribing those concepts into a proof assistant such as Coq to formalize and mechanically prove key results [8]. With that in mind, we propose a formal sustainability theory for real-time scheduling, which we present in §2.

Our goal in this paper is not only to clarify what sustainability means, but also to provide a foundation for less pessimistic schedulability analyses for policies that are sustainable *with varying parameters* such as the suspension times in the example from §1.2, a new concept that we call *weakly-sustainable policy*. The exact definition and implications of weak sustainability will be discussed in §3.

Finally, we apply this newly defined notion of weak-sustainability in §4, where we formalize self-suspending tasks in Coq and mechanically prove that uniprocessor, job-level fixed priority (JLFP) scheduling of self-suspending tasks under the dynamic suspension model is weakly-sustainable with respect to job costs and varying suspension times.

To summarize, this paper makes the following contributions:

1. a formal theory of sustainability in real-time scheduling, with definitions of sustainable policy [3, 7], sustainable analysis [3, 6, 7] and self-sustainable analysis [3] generalized to *any scheduling policy and any task and platform models* (§2);
2. the definition of the new notions of strongly- and weakly- sustainable policies (§3), and the corresponding composition rules (§3.2);
3. the first formalization of sustainability theory and real-time scheduling with self-suspensions in a proof assistant (§4.1 and online appendix [1]); and
4. a mechanized proof of weak sustainability of uniprocessor JLFP scheduling of dynamic self-suspending tasks with respect to job costs and varying suspension times (§4.2–§4.4 and online appendix [1]).

2 Formalization of Sustainability Theory

In this section, we formalize the theory of sustainability in real-time scheduling and characterize the basic notions of sustainability proposed in the literature, namely *sustainable policy* [3, 7], *sustainable analysis* [6, 7] and *self-sustainable analysis* [3].

Our motivation for developing this theory is twofold: we aim to (a) clarify and generalize the existing notions of sustainability so that they become compatible with *any scheduling policy and any task and platform models*, and (b) provide the theoretical support for defining the new concept of *weak sustainability*, which will be covered in §3 and mechanically proven in §4 for uniprocessor JLFP scheduling of dynamic self-suspending tasks.

Note that this section does not fundamentally introduce new concepts but spells out precisely common implicit assumptions about the task and platform models and gives a more formal presentation of the *real-time scheduling meta-theory*, which will be used to mechanically prove the results (see §4).

In order to distinguish the different nuances of sustainability, one must be able to correlate the variation of job and task parameters with schedulability. Hence, we must formalize the system model and present the basic definitions related to jobs and tasks.

2.1 Platform Model

We begin by stating the main assumptions about the platform model, in particular the notions of time, and platform parameters, which specify part of the scheduling problem to be solved.

First, note that all definitions in this paper are compatible with both *discrete and dense time.*

Next, to be able to represent the different system models from the literature, we introduce the concept of a processor platform and its associated parameters.

▶ **Definition** 4 (Processor Platform). *Let platform* Π *be the system on which jobs are scheduled.*

▶ **Definition** 5 (Platform Parameter). *Each platform* Π *has a finite set of parameters* $\mathcal{P}_{plat} = \{p_1, \ldots, p_n\}$*.*

▶ **Example** 1 (Common Platforms). *Examples of platforms include uniprocessor systems, identical multiprocessors [10], and uniform multiprocessors [4]. Multiprocessor platforms usually have an associated parameter* $m \in \mathcal{P}_{plat}$ *that indicates the number of processors.*

Note that Definition 3 does not limit the set of parameters defining a platform to its number of processors; in fact, the set of parameters $\mathcal{P}_{plat}$ could also express the heterogeneity of the platform [5], its power consumption, or execution speed profiles [17]. We keep the set of parameters unspecified in order to retain maximal generality.

This approach is uncommon. Most works tend to limit their results to a specific system model (e.g., task-level fixed priority scheduling of sequential tasks on single or multi-core processors). Instead, we prefer generality to specificity, so that the concepts and properties presented hereafter can be instantiated for any scheduling problem.

2.1.1 Jobs

After discussing the general aspects of the system model, we now define a job set.

▶ **Definition** 6 (Job Set). *We define a job set* $\mathcal{J}$ *as a (potentially infinite) collection of jobs* $\mathcal{J} = \{j_1, j_2, \ldots\}$*.*

Next, in order to define sustainability without being restricted to a particular task model, we generalize the notion of a job parameter.

▶ **Definition** 7 (Job Parameter). *We denote as job parameters any finite set* $\mathcal{P}_{job} = \{p_1, \ldots, p_n\}$*, where each parameter* $p_i \in \mathcal{P}_{job}$ *is a function over jobs.* ◀

▶ **Example** 2. *Common job parameters include* $cost(j)$*, the actual job execution time,* $arrival(j)$*, the absolute job arrival time, and* $deadline(j)$*, the relative job deadline. They may for instance also include the job suspension time in the case of self-suspending jobs, its level of parallelism and/or its energy consumption if such properties are of interest.* ◀

Next, we define the notion of scheduling policy, which specifies the strategy for selecting jobs to be scheduled.

▶ **Definition** 8 (Scheduling Policy). *Given a platform* π *and a job set* $\mathcal{J}$ *with job parameters* $\mathcal{P}_{job}$*, we define a scheduling policy* σ *as any algorithm that determines which jobs in* $\mathcal{J}$ *are scheduled at any time* t *on platform* Π*.*

For job sets that have associated deadlines, we can also define whether they are schedulable.

▶ **Definition** 9 (Schedulable Job Set). *Assume that jobs have a deadline as one of their parameters. Then, we say that a job set* $\mathcal{J}$ *is schedulable on platform* Π *under policy* σ *iff none of its jobs misses a deadline when scheduled on* Π *under policy* σ*.*

To compare different job sets, we must also be able to express how job parameters can vary across job sets (*e.g.*, a job's cost increased while its arrival time remained constant). For that, we define whether two job sets differ only by a given set of parameters.

▶ **Definition 10** (Varying Job Parameters in V). *Consider any subset of job parameters $V \subseteq \mathcal{P}_{job}$, which we call variable parameters, and consider two enumerated job sets $\mathcal{J} = \{j_1, j_2, \ldots\}$ and $\mathcal{J}' = \{j_1', j_2', \ldots\}$. We say that $\mathcal{J}$ and $\mathcal{J}'$ differ only by V iff $|\mathcal{J}| = |\mathcal{J}'|$ and $\forall i, \forall p \in (\mathcal{P}_{job} \setminus V),\ p(j_i) = p(j_i')$, where $|\mathcal{J}|$ denotes the cardinality of job set $\mathcal{J}$.*

▶ **Example 3.** By stating that $\{j_1, j_2\}$ and $\{j_1', j_2'\}$ differ only by $V = \{cost\}$, we claim that jobs j_1 and j_1' (respectively, j_2 and j_2'), are identical in all parameters other than *cost*. This is useful to formalize, for example, the idea that "schedulability is maintained when reducing *only* the cost of a job".

2.1.2 Tasks

While some notions of sustainability apply exclusively to job sets, one can also describe how the variation of task parameters affects schedulability analysis results. To be able to reason at the task level, we begin by defining task set and task parameters.

▶ **Definition 11** (Task Set). *We define a task set τ as a finite set of tasks $\{T_1, \ldots, T_n\}$.*

▶ **Definition 12** (Task Parameters). *We call task parameters any finite set $\mathcal{P}_{task} = \{p_1, \ldots, p_n\}$, where each parameter $p_i \in \mathcal{P}_{task}$ is a function over tasks.*

▶ **Example 4.** Similar to the job parameters in Example 2, common task parameters include, but are not limited to, $cost(T_i)$, the worst-case execution time of task T_i, and $period(T_i)$, the period or minimum inter-arrival time of task T_i.

Next, we define a task model, which determines how job sets are related to task sets.

▶ **Definition 13** (Task Model). *We define a task model $\mathcal{M}$ as the collection of all task sets that can be defined with given task parameters $\mathcal{P}_{task}$, along with a set of constraints relating job parameters with task parameters.*

▶ **Definition 14** (Generated Job Sets). *Every task set $\tau \in \mathcal{M}$ generates a (potentially infinite) collection of job sets denoted $jobsets(\tau) = \{\mathcal{J}_1, \mathcal{J}_2, \ldots\}$, with the condition that for every job set $\mathcal{J} \in jobsets(\tau)$ and every job $j \in \mathcal{J}$, (a) j belongs to an associated task in τ, denoted $task(j)$, and (b) the job parameters of j are constrained by the task parameters of $task(j)$, as determined by $\mathcal{M}$.*

One example of such task model constraint is the upper bound on job execution times.

▶ **Example 5** (Constraint on Job Execution Time). *Let $\mathcal{M}$ be the sporadic task model. Let the job parameter $cost(j)$ denote the actual execution time of job j and let the task parameter $cost(T_i)$ denote the WCET of task T_i. For every job set $\mathcal{J}$ generated by $\mathcal{M}$, the cost of each job $j \in \mathcal{J}$ is upper-bounded by the cost of its task, i.e.,*

$$\forall \tau \in \mathcal{M}, \forall \mathcal{J} \in jobsets(\tau), \forall j \in \mathcal{J}, cost(j) \leq cost(task(j)).$$

Using the notion of generated job sets, we can now define whether a task set is schedulable.

▶ **Definition 15** (Schedulable Task Set). *We say that task set $\tau \in \mathcal{M}$ is schedulable on platform Π under scheduling policy σ iff every generated job set $\mathcal{J} \in jobsets(\tau)$ is schedulable on Π under σ.*

Similarly to Definition 10, in order to relate parameters across task sets, we define whether two task sets differ only by a given set of parameters.

▶ **Definition 16** (Varying Task Parameters in V). *Consider any subset of task parameters $V \subseteq \mathcal{P}_{task}$, which we call variable parameters, and consider two task sets $\tau = \{T_1, T_2, \ldots\}$ and $\tau' = \{T_1', T_2', \ldots\}$. We say that τ and τ' differ only by V iff $|\tau| = |\tau'|$ and $\forall\ i, \forall\ p \in (\mathcal{P}_{task} \setminus V),\ p(T_i) = p(T_i')$, where $|\tau|$ denotes the cardinality of task set τ.*

2.2 Generalized Sustainability Definitions

In this section, we use the basic concepts about jobs and tasks to formalize the notions of sustainability found in the literature, namely *sustainable policy* (§2.2.1), *sustainable analysis* (§2.2.2) and *self-sustainable analysis* (§2.2.3). Note that, differently from prior work [3, 6, 7], our definitions are generic and compatible with different task and platform models.

2.2.1 Sustainable Scheduling Policy

We begin by generalizing the concept of a *sustainable scheduling policy* [7, 3], which was briefly discussed in §1. The definition captures the idea that if a policy is sustainable with respect to a set of job parameters, having "better" values for those parameters at runtime does not cause any deadline misses. We call this notion, "strong sustainability" for reasons that will be made clear in §3. Formally, it is stated as follows.

▶ **Definition** 17 (Strongly-Sustainable Policy). *Assume any real-time scheduling policy σ and platform Π, and consider any subset of job parameters $S \subseteq \mathcal{P}_{job}$, which we call sustainable parameters. For each parameter $p \in S$, let $\mathcal{J} \preceq_p \mathcal{J}'$ be any partial-order relation over job sets $\mathcal{J}$ and $\mathcal{J}'$ that indicates that every job in $\mathcal{J}$ has no worse parameter p than its corresponding job in $\mathcal{J}'$. Then, we say that the scheduling policy σ is strongly-sustainable with respect to the job parameters in S iff*

$$\begin{aligned}&\forall\ \mathcal{J}\ \textbf{s.t.}\ \mathcal{J} \text{ is schedulable on platform } \Pi \text{ under policy } \sigma,\\ &\quad \forall\ \mathcal{J}_{better}\ \textbf{s.t.}\ \mathcal{J} \text{ and } \mathcal{J}_{better} \text{ differ only by } S \textbf{ and } \forall p \in S,\ \mathcal{J}_{better} \preceq_p \mathcal{J},\\ &\qquad \mathcal{J}_{better} \text{ is schedulable on platform } \Pi \text{ under policy } \sigma.\end{aligned}$$

Definition 17 states that, under a strongly-sustainable scheduling policy σ, whenever we compare two job sets and show that the job set with "worse parameters" does not miss any deadline, then the job set with "better parameters" must also not miss any deadline.

Note that the relation $\preceq_p$ is a crucial part of the specification and should be clearly indicated in the sustainability claim, as shown in the next examples.

▶ **Example** 6 (Sustainability with Decreasing Job Costs).

Let σ denote any uniprocessor work-conserving, fixed-priority scheduling policy under the sporadic task model. Let $cost(j)$ denote the actual execution time of job j. Given any job sets $\mathcal{J} = \{j_1, j_2, \ldots\}$ and $\mathcal{J}' = \{j'_1, j'_2, \ldots\}$, we define the relation $\mathcal{J} \preceq_{cost} \mathcal{J}'$ as $\forall i, cost(j_i) \leq cost(j'_i)$.

Using the relation $\preceq_{cost}$, we can instantiate Definition 17. This property expresses the notion that, under policy σ, decreasing job execution times does not render the system unschedulable. This property was proven by Ha and Liu [12] for different job models. ◀

Similarly, one can also define sustainability with respect to job inter-arrival times. It just requires a more nuanced parameter order definition, as shown in the following example.

▶ **Example** 7 (Sustainability with Increasing Job Inter-Arrival Times).

Let σ denote any work-conserving, fixed-priority scheduling policy under the sporadic task model. Let $arrival(j)$ denote the absolute arrival time of job j. Next, given any job *sets* $\mathcal{J} = \{j_1, j_2, \ldots\}$ *and* $\mathcal{J}' = \{j'_1, j'_2, \ldots\}$*, we define the relation* $\preceq_{interarrival}$ *as*

$$\begin{aligned}
&\forall i, \forall j_{prev}, \forall j'_{prev} \textbf{ s.t.} \\
&\quad task(j_i) = task(j_{prev}) = task(j'_i) = task(j'_{prev}) \textbf{ and} \\
&\quad arrival(j_{prev}) < arrival(j_i) \textbf{ and } arrival(j'_{prev}) < arrival(j'_i), \\
&\qquad arrival(j_i) - arrival(j_{prev}) \geq arrival(j'_i) - arrival(j'_{prev}).
\end{aligned}$$

This relation expresses that if $\mathcal{J} \preceq_{interarrival} \mathcal{J}'$*, then the distance between two consecutive jobs of the same task in* $\mathcal{J}$ *is no worse (i.e., no smaller) than in* $\mathcal{J}'$*.*

Based on the job arrival function and the relation $\preceq_{interarrival}$*, we can instantiate Definition 17 to obtain the corresponding sustainability property for policy* σ*.* ◀

Finally, note that Definition 17 differs from Definition 3 (in §1.2) due to Baker and Baruah [3], as it does not require the original job set $\mathcal{J}$ to belong to some task set τ that is schedulable. Thus, according to our definition, Figs. 1-(a) and 1-(b) are a valid counterexample for establishing non-sustainability (in the strong sense), which agrees with Definition 2 in §1.1.

2.2.2 Sustainable Schedulability Analysis

Having discussed how sustainability applies to scheduling policies, we now present the corresponding definitions for schedulability analyses, starting with the notion of *sustainable schedulability analysis* [6, 7, 3]. Before we proceed, we must define schedulability analysis.

▶ **Definition** 18 (Schedulability Analysis). *Let a schedulability analysis* $\mathcal{A}$ *for task model* $\mathcal{M}$*, platform* Π*, and scheduling policy* σ *denote any algorithm that assesses whether a task set* $\tau \in \mathcal{M}$ *is schedulable on* Π *under policy* σ*.*

Now we state whether a given schedulability analysis $\mathcal{A}$ is sustainable. The intuition is that if analysis $\mathcal{A}$ is sustainable with respect to certain job parameters, then for any task set τ that is deemed schedulable by $\mathcal{A}$, having better values for such parameters at runtime than those in the job sets generated by τ does not cause any deadline misses.

▶ **Definition** 19 (Sustainable Analysis). *Consider any schedulability analysis* $\mathcal{A}$ *for task model* $\mathcal{M}$*, platform* Π*, and scheduling policy* σ*, and consider any subset of job parameters* $S \subseteq \mathcal{P}_{job}$*, which we call sustainable parameters. For each parameter* $p \in S$*, let* $\mathcal{J} \preceq_p \mathcal{J}'$ *be any partial-order relation over job sets* $\mathcal{J}$ *and* $\mathcal{J}'$ *that indicates whether every job in* $\mathcal{J}$ *has no worse parameter* p *than its corresponding job in* $\mathcal{J}'$*. Then, we say that analysis* $\mathcal{A}$ *is sustainable with respect to* S *iff*

$$\begin{aligned}
&\forall\ \tau\ \in \mathcal{M} \textbf{ s.t. } \tau \text{ is deemed schedulable by } \mathcal{A}, \\
&\quad \forall\ \mathcal{J} \in jobsets(\tau), \forall\ \mathcal{J}_{better} \textbf{ s.t.} \\
&\qquad \mathcal{J} \text{ and } \mathcal{J}_{better} \text{ differ only by } S \textbf{ and } \forall p \in S, \mathcal{J}_{better} \preceq_p \mathcal{J}, \\
&\qquad\quad \mathcal{J}_{better} \text{ is schedulable on } \Pi \text{ under policy } \sigma.
\end{aligned}$$

Although the definitions of strongly-sustainable policy (Definition 17) and sustainable analysis (Definition 19) both refer to the runtime behavior of the policy, the two notions are different. If the analyzed policy σ is strongly-sustainable w.r.t. some parameters S, then any sufficient or exact schedulability analysis for σ is also sustainable w.r.t. S. However, even if σ is not strongly-sustainable, it is possible to find sufficient schedulability analyses that are sustainable. In fact, we argue this is exactly the case that an intuitive notion of a "safe analysis" is trying to address: the underlying policy σ may exhibit various kinds of scheduling anomalies, but if a specific task set is deemed schedulable by a sustainable analysis, then no deadlines will be missed in the actual system even if some parameters turn out to be "better in the real system than assumed during analysis".

2.2.3 Self-Sustainable Analysis

Another type of sustainability that can be found in the literature, also related to schedulability analysis, is the notion of *self-sustainable analysis* [3]. The intuition is that if analysis $\mathcal{A}$ is self-sustainable with respect to a set of task parameters, then for any task set that is deemed schedulable by analysis $\mathcal{A}$, every task set with better values for those parameters will also be deemed schedulable by $\mathcal{A}$.

▶ **Definition** 20 (Self-Sustainable Analysis). *Let $\mathcal{A}$ be any schedulability analysis for task model $\mathcal{M}$, platform Π, and scheduling policy σ, and consider any subset of task parameters $S \subseteq \mathcal{P}_{task}$. For each parameter $p \in S$, let $\tau \preceq_p \tau'$ be any partial-order relation over task sets τ and τ' that indicates whether every task in τ has no worse parameter p than its corresponding task in τ'. Then we say that schedulability analysis $\mathcal{A}$ is self-sustainable with respect to S iff*

$$\begin{aligned} &\forall\ \tau \in \mathcal{M} \textbf{ s.t. } \tau \text{ is deemed schedulable by } \mathcal{A}, \\ &\quad \forall\ \tau_{better} \textbf{ s.t. } \tau \text{ and } \tau_{better} \text{ differ only by } S \textbf{ and } \forall p \in S, \tau_{better} \preceq_p \tau, \\ &\qquad \tau_{better} \text{ is deemed schedulable by } \mathcal{A}. \end{aligned} \tag{1}$$

To clarify the definition, we provide an example.

▶ **Example** 8 (RTA is Self-Sustainable with respect to Decreasing Task Costs).
Let $\mathcal{A}$ be some response-time analysis (RTA) for the sporadic task model and let $WCET(T_i)$ denote the worst-case execution time of task T_i. Given any task sets $\tau = \{T_1, T_2, \ldots\}$ and $\tau' = \{T'_1, T'_2, \ldots\}$ with same number of tasks, we define the relation $\tau \preceq_{WCET} \tau'$ as $\forall i$, $WCET(T_i) \leq WCET(T'_i)$.
Based on the task parameter WCET and the relation $\preceq_{WCET}$, we can instantiate the self-sustainability property as in Definition 20, which then expresses the notion that if the RTA claims τ to be schedulable, then it must also claim task sets with lower WCETs to be schedulable. ◀

Note that, despite their similarity, the notions of sustainable and self-sustainable analysis are fundamentally different. While sustainability refers to job parameters, self-sustainability concerns task parameters. Moreover, to prove that an analysis $\mathcal{A}$ is sustainable, one must show that the job sets generated by a task set τ deemed schedulable by $\mathcal{A}$ do not have any anomalies. On the other hand, proving that analysis $\mathcal{A}$ is self-sustainable is a *purely mathematical* property, akin to a notion of monotonicity, of the analysis algorithm itself (seen as a black box) and has nothing to do with actual schedules. For example, to prove the property in Example 8, one must show that if the RTA computes a fixed point R for given task costs, then it will compute a fixed point $R' \leq R$ if lower task costs are provided.

3 Weakly-Sustainable Scheduling Policies

Recall from §1.1 that uniprocessor EDF scheduling of self-suspending tasks was proven to be not strongly-sustainable with respect to job costs, and as mentioned at the end of §2.2.1, this result agrees with our notion of strongly-sustainable policy (Definition 17).

However, in §1.2, we also hinted (but did not prove) that this scheduling policy is still sustainable to some extent with respect to job costs. As shown in Fig. 1-(c), by reducing suspension times (*i.e.*, a transformation that is compliant with the task model and the constraints set by the task set τ), we were able to construct a job set $\mathcal{J}_{susp} \in jobsets(\tau)$ that is as hard to schedule as job set $\mathcal{J}_{better}$. This suggests that any schedulability analysis $\mathcal{A}$ applied to task set τ would deem it "not schedulable" anyway because of job set $\mathcal{J}_{susp}$.

The fact that $\mathcal{J}_{better}$ itself is not schedulable does not straightforwardly prove that the uniprocessor EDF scheduling policy applied to self-suspending tasks is not sustainable in some sense w.r.t. job costs, at least if self-suspension times may vary at runtime. In fact, whether or not any parameters *other than the sustainable parameters* should be allowed to vary at runtime is the cause of most confusion in the various interpretations of sustainability found in the state of the art [3, 6, 7]. That is the motivation behind formalizing the notion of varying job and task parameters as defined in Definitions 10 and 16.

Thus, while the notion of strongly-sustainable policy (Definition 17) expresses that the system remains schedulable if we decrease job costs *while maintaining all other parameters constant*, we believe that this is too strong an assumption in many, if not most, settings. In contrast, the sustainability property that we are going to define allows *other parameters to vary*, subject to the constraints of the given task set. The reasoning behind that is that one can build more efficient schedulability analyses by knowing specifically what job parameters can be assumed to have maximal values and which ones should be considered as variables. The current theory does not allow such fine-grained categorization.

To develop a supporting theory for schedulability analyses based on this idea, in this section we propose a new classification for sustainable scheduling policies that differentiates between strong sustainability and weak sustainability.

3.1 Definition of Weakly-Sustainable Policy

As suggested in the previous section, in order to define weak sustainability, we must be able to infer that a collection of job sets remains schedulable when certain parameters are allowed to vary. This idea is captured by the following definition.

▶ **Definition** 21 (Schedulable with Varying Job Parameters V). *Given a task set τ and subset of job parameters $V \subseteq \mathcal{P}_{job}$, we say that a job set $\mathcal{J}$ is schedulable with varying parameters V subject to task set τ on platform Π under policy σ iff for any jobset $\mathcal{J}_{other} \in jobsets(\tau)$ such that $\mathcal{J}$ and $\mathcal{J}_{other}$ differ only by V, then $\mathcal{J}_{other}$ is schedulable on Π under policy σ.*

To illustrate the definition, we provide an example.

▶ **Example** 9 (Schedulable with Varying Costs).
Assume any scheduling policy σ and consider the set of variable parameters $V = \{cost\}$. Given a job set $\mathcal{J} = \{j_1, j_2\}$ generated by task set τ, we say that $\mathcal{J}$ is schedulable with varying costs subject to task set τ iff every job set $\mathcal{J}_{other}$ generated by τ that has two jobs and the same parameters as $\mathcal{J}$ except for their costs is schedulable. That is, any job set constructed by changing only the job costs of $\mathcal{J}$ (to higher or lower values), without violating the constraints set forth by the parameters of task set τ, must be schedulable. ◀

In other words, one way to think about this notion is to say that job set $\mathcal{J}$ is not only schedulable itself, but also a "schedulability witness" for a whole family of related job sets that are identical in all parameters except for those in V. Using the concept of schedulability with varying parameters, we can now define whether a policy is weakly-sustainable.

▶ **Definition** 22 (Weakly-Sustainable Policy). *Assume any platform* Π*, task model* $\mathcal{M}$*, and scheduling policy* σ*, and consider any disjoint subsets of job parameters* $S \subseteq \mathcal{P}_{job}$ *and* $V \subseteq \mathcal{P}_{job}$*, which we call sustainable and variable parameters, respectively. For each sustainable parameter* $p \in S$*, let* $\mathcal{J} \preceq_p \mathcal{J}'$ *be any partial-order relation over job sets* $\mathcal{J}$ *and* $\mathcal{J}'$ *that indicates whether every job in* $\mathcal{J}$ *has no worse parameter* p *than its corresponding job in* $\mathcal{J}'$*. Then we say that scheduling policy* σ *is weakly-sustainable with sustainable parameters* S *and variable parameters* V *iff*

$$\forall\, \tau \in \mathcal{M}, \forall\, \mathcal{J} \in \mathit{jobsets}(\tau) \textbf{ s.t.}$$
$$\mathcal{J} \text{ is schedulable with varying } V \text{ subject to } \tau \text{ on platform } \Pi \text{ under policy } \sigma,$$
$$\forall\, \mathcal{J}_{better} \textbf{ s.t. } \mathcal{J} \text{ and } \mathcal{J}_{better} \text{ differ only by } S \textbf{ and } \forall\, p \in S, \mathcal{J}_{better} \preceq_p \mathcal{J},$$
$$\mathcal{J}_{better} \text{ is schedulable on platform } \Pi \text{ under policy } \sigma.$$

The idea of weak sustainability is that if we can determine (with some schedulability analysis $\mathcal{A}$) that a job set is schedulable for all variations of parameters in V (subject to the constraints imposed by its associated task set), then all job sets with better parameters S must be schedulable. For clarity, we provide the following example.

▶ **Example** 10 (Weak Sustainability w.r.t. Job Costs and Varying Suspension Times). *Consider any uniprocessor JLFP scheduling policy* σ *under the dynamic suspension model, i.e., jobs can suspend at any time but the total suspension duration of each job is bounded by its task's maximum suspension time. Let* $\mathit{susp}(j)$ *denote the total suspension time of job* j *and* $\mathit{cost}(j)$ *the execution time of job* j*.*

Using job parameters $S = \{\mathit{cost}\}$ *and* $V = \{\mathit{susp}\}$*, and the relation* $\preceq_{cost}$ *as in Example 6, one can instantiate Definition 22 and prove (as will be shown in §4) that for any task set* $\tau \in \mathcal{M}$*, if job set* $\mathcal{J}$ *generated by* τ *is schedulable for all variations of suspension times (subject to the constraints imposed by* τ*), then all job sets with lower or equal job costs will also be schedulable.* ◀

In the specific case where the set of varying parameters V is empty, we call the scheduling policy *strongly-sustainable*.

▶ **Definition** 23 (Strongly-Sustainable Policy). *We say that a policy is strongly-sustainable with respect to the job parameters in* S *iff it is weakly-sustainable with respect to the sustainable parameters in* S *and an empty set of variable parameters* $V = \emptyset$*.*

Note that if $V = \emptyset$, proving that job set $\mathcal{J}$ is schedulable with varying parameters V is the same as establishing that $\mathcal{J}$ itself is schedulable. That implies the following equivalence, which connects the definitions of sustainable policy in §2 and §3.

▶ **Corollary** 1 (Equivalence of Strong Sustainability). *The notion of strongly-sustainable policy as defined in Definition 23 is equivalent to Definition 17.*

For practical purposes, the weak sustainability property is useful for constraining the search space when developing schedulability analyses for some scheduling policy σ. As is already known, if policy σ is strongly-sustainable with respect to the parameters in S, maximizing/minimizing such parameters enables constructing worst-case scenarios (*e.g.*, the critical instant for uniprocessor FP scheduling of sporadic tasks [14]).

However, recall that policy σ might not be strongly-sustainable with respect to S. But if we are still able to prove that σ is weakly-sustainable with respect to S and variable parameters V, we can still maximize/minimize the parameters in S, as long as the schedulability analysis checks all values of the parameters in V. In other words, establishing a weak sustainability property can be thought of as a dimensionality reduction.

For instance, having proven in Theorem 4 in §4.4 that uniprocessor JLFP scheduling of self-suspending tasks is weakly-sustainable with respect to job costs and variable suspension times, we know that any schedulability analysis for that model may assume that all jobs generated by the tasks execute for their maximum execution time, and must only search for the worst-case assignments of job suspension times.

3.2 Composing Weak Sustainability Results

Although the definition of strong sustainability refers to a set S of multiple parameters, one can still establish the sustainability of each parameter in isolation. In fact, the critical instant for the sporadic task model is obtained by composing worst-case assumptions about individual job parameters: maximizing job costs, minimizing inter-arrival time, etc.

As will be shown in Theorem 1, this composition rule applies not only for strong sustainability (as discussed in prior work [3]), but can also be extended to weak sustainability. Before presenting the theorem, we first provide an alternative definition of weak sustainability (based on the contrapositive of Definition 22), which simplifies the proofs.

▶ **Definition** 24 (Weakly-Sustainable Policy – alternative definition). *Assume any platform* Π, *task model* $\mathcal{M}$ *and scheduling policy* σ, *and consider any disjoint subsets of job parameters* $S \subseteq \mathcal{P}_{job}$ *and* $V \subseteq \mathcal{P}_{job}$, *which we call sustainable and variable parameters, respectively. For each sustainable parameter* $p \in S$, *let* $\mathcal{J} \preceq_p \mathcal{J}'$ *be any partial-order relation over job sets* $\mathcal{J}$ *and* $\mathcal{J}'$ *that indicates whether every job in* $\mathcal{J}$ *has no worse parameter* p *than its corresponding job in* $\mathcal{J}'$. *Then, we say that the scheduling policy is weakly-sustainable with sustainable parameters* S *and variable parameters* V *iff*

$$
\begin{aligned}
&\forall\ \mathcal{J}\ \textbf{s.t.}\ \mathcal{J} \text{ is } \textit{not} \text{ schedulable on platform } \Pi \text{ under policy } \sigma,\\
&\quad \forall\ \tau \in \mathcal{M},\ \forall \mathcal{J}_{worse} \in \textit{jobsets}(\tau)\ \textbf{s.t.}\\
&\qquad \mathcal{J} \text{ and } \mathcal{J}_{worse} \text{ differ only by } S \textbf{ and } \forall p \in S, \mathcal{J} \preceq_p \mathcal{J}_{worse},\\
&\qquad\quad \exists\ \mathcal{J}'_{worse} \in \textit{jobsets}(\tau)\ \textbf{s.t.}\\
&\qquad\qquad \mathcal{J}_{worse} \text{ and } \mathcal{J}'_{worse} \text{ differ only by } V \textbf{ and}\\
&\qquad\qquad \mathcal{J}'_{worse} \text{ is not schedulable on platform } \Pi \text{ under policy } \sigma.
\end{aligned}
$$

Put differently, for any job set $\mathcal{J}$ that is not schedulable, if we can find another job set $\mathcal{J}_{worse}$ that is generated by some task set τ and $\mathcal{J}$ is "better" than $\mathcal{J}_{worse}$, then there exists a member in $\mathcal{J}_{worse}$'s "family" of related jobs that is also not schedulable.

In addition, we must introduce the notion of independent sets of job parameters.

▶ **Definition** 25 (Independent Sets of Job Parameters). We say that subsets of job parameters $A \subset \mathcal{P}_{job}$ and $B \subset \mathcal{P}_{job}$ are independent with respect to task model $\mathcal{M}$ iff for all task parameter p_{task} defined by $\mathcal{M}$, and for every $p_A \in A$ and $p_B \in B$, if p_A is constrained by p_{task} according to model $\mathcal{M}$, then p_B is not constrained by p_{task} according to model $\mathcal{M}$.

In the most common task models considered in the real-time literature, job parameters are usually independent of each other.

▶ **Example** 11 (Parameters Are Usually Independent). In the sporadic task model with self-suspending tasks, the sets of job parameters $A = \{\textit{cost}, \textit{arrival}\}$ and $B = \{\textit{susp}\}$ have independent task constraints, since these job parameters are each constrained by a different task parameter, namely, the task WCET, minimum inter-arrival time and maximum suspension time. In contrast, in a hypothetical task model where every job j is split into two execution sections of length $\textit{cost}_1(j)$ and $\textit{cost}_2(j)$ such that $\textit{cost}_1(j)+\textit{cost}_2(j) \leq \textit{cost}(\textit{task}(j))$, the parameters $\{\textit{cost}_1\}$ and $\{\textit{cost}_2\}$ are clearly non-independent.

Using the definition of weak sustainability above (Definition 24) and the notion of independent sets of job parameters (Definition 25), we establish the composition rule for weakly-sustainable policies.

▶ **Theorem** 1 (Composition Rule: Weak – Weak). *Consider any task model $\mathcal{M}$, scheduling policy σ and processor platform Π. Let S_a, V_a, S_b, V_b denote subsets of the job parameters $\mathcal{P}_{job}$ such that $S_a \cap V_b = \emptyset$ and $S_b \cap V_a = \emptyset$, and such that either S_b is independent of $\mathcal{P}_{job} \setminus S_b$, or S_a is independent of $\mathcal{P}_{job} \setminus S_a$, with respect to task model $\mathcal{M}$. Assume that (a) σ is weakly-sustainable with respect to S_a and variable parameters V_a, and that (b) σ is weakly-sustainable with respect to S_b and variable parameters V_b. Then (c) σ is weakly-sustainable with respect to $S_a \cup S_b$ and variable parameters $V_a \cup V_b$.*

Proof. Consider any job set $\mathcal{J}$ that is not schedulable on platform Π under policy σ. Let τ be any task set, and let $\mathcal{J}_{worse} \in jobsets(\tau)$ be any job set that only differs from $\mathcal{J}$ by the parameters in $S_a \cup S_b$ and that w.r.t. $S_a \cup S_b$ has no better parameters than $\mathcal{J}$. Then, according to Definition 24, we must prove that there exists a job set $\mathcal{J}'_{worse} \in jobsets(\tau)$ that only differs from $\mathcal{J}_{worse}$ with respect to $V_a \cup V_b$ and that is also not schedulable.

Using the independent parameters assumption, assume without loss of generality that S_b is independent of all other job parameters $\mathcal{P}_{job} \setminus S_b$ with respect to model $\mathcal{M}$. If this is not the case, then by assumption we have that S_a is independent of other parameters $\mathcal{P}_{job} \setminus S_a$ and we can build a job set J'_b using parameters in S_b instead of building a job set J'_a using parameters in S_a in Step 1 of the proof.

1. **Step 1 – Construction of $\mathcal{J}'_a$ from $\mathcal{J}$:** Let $\mathcal{J}_a$ be the same job set as $\mathcal{J}$, but with the same job parameters in S_a as $\mathcal{J}_{worse}$. That is, let $\mathcal{J} = \{j_1, j_2, \ldots\}$ and $\mathcal{J}_{worse} = \{j_1^w, j_2^w, \ldots\}$ and recall that they have the same number of jobs. Then, we define $\mathcal{J}_a = \{j_1^a, j_2^a, \ldots\}$ with same cardinality such that for any index i, we have $\forall p \in S_a, p(j_i^a) = p(j_i^w)$ and $\forall p \notin S_a, p(j_i^a) = p(j_i)$.
   Next, we construct a task set $\tau_a \in \mathcal{M}$ such that for every task parameter p_{task} that constrains job parameters in $\mathcal{P}_{job} \setminus S_b$, the value of p_{task} in τ_a is the same as in τ, and for for every task parameter p_{task} that constrains job parameters in S_b, the value of p_{task} in τ_a is the same to the task set that generated job set $\mathcal{J}$. Since $\mathcal{J}_a$ only differs from $\mathcal{J}_{worse} \in jobsets(\tau)$ with respect to S_b, and S_b is independent of the other job parameters, it follows that $\mathcal{J}_a \in jobsets(\tau_a)$.
   Since $\mathcal{J}$ is not schedulable, and $\mathcal{J}$ and $\mathcal{J}_a$ differ only by S_a, we can exploit the fact that σ is weakly-sustainable with S_a and varying V_a. Thus, it follows that there exists a job set $\mathcal{J}'_a \in jobsets(\tau_a)$ that differs from $\mathcal{J}_a$ only by the parameters in V_a and that is not schedulable on platform Π under policy σ.
2. **Step 2 – Construction of $\mathcal{J}'_{ab}$ from $\mathcal{J}'_a$:** Let $\mathcal{J}_{ab}$ be the same job set as $\mathcal{J}'_a$ except that the job parameters in S_b are the same as in $\mathcal{J}_{worse}$. That is, let $\mathcal{J}'_a = \{j_1^{a'}, j_2^{a'}, \ldots\}$ and $\mathcal{J}_{worse} = \{j_1^w, j_2^w, \ldots\}$ and recall that they have the same number of jobs. Then, we define $\mathcal{J}_{ab} = \{j_1^{ab}, j_2^{ab}, \ldots\}$ with same cardinality such that for any index i, we have $\forall p \in S_b, p(j_i^{ab}) = p(j_i^w)$ and $\forall p \notin S_b, p(j_i^{ab}) = p(j_i^{a'})$.
   Note that by construction, $\mathcal{J}_{ab}$ has the same job parameters as $\mathcal{J}_{worse} \in jobsets(\tau)$, except for V_a, which was obtained when generating $\mathcal{J}'_a$ via weak-sustainability. However, note that $\mathcal{J}'_a$ is generated by task set τ_a, which has the same constraints for V_a as τ, since V_a is independent of the other parameters. Thus, every job parameter of $\mathcal{J}_{ab}$ is compatible with τ, *i.e.*, $\mathcal{J}_{ab} \in jobsets(\tau)$.
   Since $\mathcal{J}'_a$ is not schedulable, and $\mathcal{J}'_a$ and $\mathcal{J}_{ab}$ differ only by S_b, we can exploit the fact that σ is weakly-sustainable with S_b and varying V_b. Thus, there must exist a job set

$\mathcal{J}'_{ab} \in jobsets(\tau)$ that differs from $\mathcal{J}_{ab}$ only by the parameters in V_b and that is not schedulable on platform Π under policy σ.

Since $\mathcal{J}$ has the same parameters as $\mathcal{J}_{worse}$ except for those in $S_a \cup S_b$, and because $\mathcal{J}_a$ and $\mathcal{J}_{ab}$ were constructed from $\mathcal{J}$ by copying the parameters S_a and S_b from $\mathcal{J}_{worse}$ and varying the parameters in $V_a \cup V_b$, it follows that $\mathcal{J}'_{ab}$ has the same parameters as $\mathcal{J}_{worse}$, except for the variable parameters V_a and V_b. Moreover, since $S_a \cap V_b = \emptyset$ and $S_b \cap V_a = \emptyset$, this guarantees that S_a and S_b do not vary during the construction of $\mathcal{J}'_a$ and $\mathcal{J}'_{ab}$, so for every $p \in S_a \cup S_b$, the order $\preceq_p$ is preserved across the successive job set transformations.

Thus, we conclude that there exists a job set $\mathcal{J}'_{worse} = \mathcal{J}'_{ab}$ that belongs to $jobsets(\tau)$, that only differs from $\mathcal{J}_{worse}$ with respect to $V_a \cup V_b$ and is also not schedulable on platform Π under policy σ. ◀

Assuming $V_b = \emptyset$ yields a rule for combining strong and weak sustainability results.

▶ **Corollary** 2 (Composition Rule: Weak – Strong). *Consider any scheduling policy σ and processor platform Π. Let S_a, V_a and S_b denote subsets of the job parameters $\mathcal{P}_{job}$ such that $S_b \cup V_a = \emptyset$. Assume that σ is weakly-sustainable with respect to S_a and variable V_a and also strongly-sustainable with respect to S_b. Then, σ is weakly-sustainable with respect to $S_a \cup S_b$ and variable V_a.*

Finally, assuming $V_a = V_b = \emptyset$ yields the composition rule for strong sustainability, which was already proven by Baker and Baruah [3].

▶ **Corollary** 3 (Composition Rule: Strong – Strong). *Consider any scheduling policy σ and processor platform Π. Let S_a and S_b denote subsets of the job parameters $\mathcal{P}_{job}$. Assume that σ is strongly-sustainable with respect to S_a and also strongly-sustainable with respect to S_b. Then, σ is strongly-sustainable with respect to $S_a \cup S_b$.*

4 Uniprocessor Scheduling of Dynamic Self-Suspending Tasks is Weakly-Sustainable w.r.t. Job Costs and Variable Suspensions

In this section, we prove that uniprocessor JLFP scheduling with dynamic self-suspending tasks is weakly-sustainable with respect to job costs and variable suspension times. Although we could have focused on other real-time task models, we chose to study the sustainability of self-suspending tasks for the following reasons.

1. **Recent Errors:** This topic has faced many misunderstandings and errors in the past, with a considerable number of unsound results being published [9]. We hope that our work on sustainability introduces helpful formalism and provides a better understanding of the task model.
2. **Future Work on Schedulability Analysis:** Proving weak sustainability of uniprocessor JLFP scheduling of dynamic self-suspending tasks can provide directions for future work. It enables less pessimistic and more efficient schedulability analyses to be developed, by reducing the search space to only the parameters that must be kept variable (i.e., suspension times), while the others (i.e., execution times) remain constant.

To address the issue of recent errors and increase the degree of confidence in the results, our proof has been mechanized in Prosa [8], a library for the Coq proof assistant that allows formal specification and mechanized proofs of real-time scheduling theory. The specification and proofs are available online [1] and can be checked independently with the CoqChk tool. Simple step-by-step instructions are provided on the website.

Note that, despite being phrased in terms of sporadic tasks under discrete time for the sake of simplicity, this proof is conceptually also compatible with other job arrival models (periodic, sporadic, bursty, *etc.*) and dense time.

The rest of this section is structured as follows. First, we present our formalization of the dynamic suspension model, which is required for stating the theorems in Prosa. Next, we provide an overview of our proof strategy based on schedule reductions, which can be reused in other sustainability proofs. In the remaining subsections, we discuss the high-level steps of the proof, which despite being specific for scheduling with self-suspensions, highlight key steps necessary in a rigorous proof of sustainability.

4.1 A Generic Suspension Model

In order to instantiate the sustainability claim for real-time scheduling of self-suspending tasks, we must formally define the concept of self-suspension.

▶ **Definition** 26 (Job Suspension Time). *We define job suspension time as a function $susp(j, s)$ such that for any job j and any value $s \in \mathbb{N}$, $susp(j, s)$ expresses the duration for which j must suspend immediately after receiving s units of service.*

The job suspension parameter is explained more clearly in the following example.

> ▶ **Example** 12 (Table of Suspension Durations).
>
> *Job suspension times $susp(j, s)$ can be understood as a table containing the duration of the suspension intervals associated with job j. For example, for a job j such that $cost(j) = 5$, we can define $susp(j, s)$ that equals 0 except for $susp(j, 3) = 2$ and $susp(j, 4) = 3$.*
>
> *This suspension table indicates that job j suspends for 2 time units just after it receives 3 units of service, and suspends for 3 time units just after it receives 1 additional unit of service. This is equivalent to saying that job j executes for 3 time units, then suspends for 2 time units, then executes for 1 more time unit, then suspends for 3 more time units and finally completes its last time unit of execution.* ◀

Note that, by allowing arbitrary suspension durations between each unit of service, this model is generic enough to represent any suspension pattern under discrete time. Thus, it supports both the segmented [16] and the dynamic suspension model [13], which are both commonly used in the literature on self-suspensions in real-time systems.

By accumulating suspension durations, we can define the total suspension time of a job.

▶ **Definition** 27 (Total Suspension Time). *We define the total suspension time $susp_\Sigma(j)$ of job j as the cumulative suspension time up to completion, i.e.,*

$$susp_\Sigma(j) = \sum_{s < cost(j)} susp(j, s)$$

After clarifying job suspension times, we now define task suspension times and show how both are related under the dynamic suspension model.

▶ **Definition** 28 (Task Suspension Time). *For any task T_i, we define the task suspension time $susp(T_i)$ as an upper-bound on the total suspension time of any job of τ.*

▶ **Definition** 29 (Suspension Time Constraints). *The dynamic suspension model requires that the total suspension time of any job is upper-bounded by the suspension time of its task, i.e.,*

$$\forall\, \tau \in \mathcal{M}, \forall\, \mathcal{J} \in jobsets(\tau), \forall\, j \in \mathcal{J},\ susp_\Sigma(j) \leq susp(task(j)).$$

Beside its suspension time, every task T_i is defined by a WCET, a minimum inter-arrival time or period, and a deadline, as explained in Definition 1.

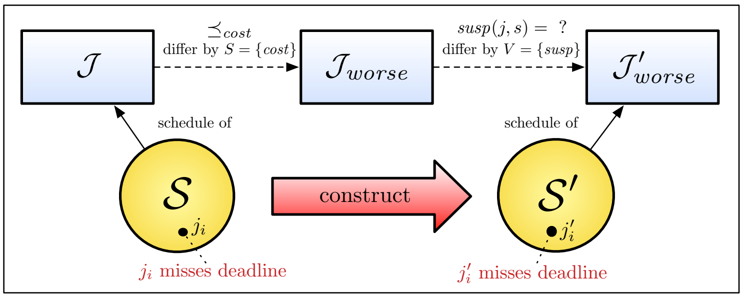


Figure 2 Proof strategy for establishing weak sustainability with respect to job costs and variable suspension times. Given a job j_i that misses a deadline in schedule $\mathcal{S}$ of the original job set $\mathcal{J}$, we construct a new job set $\mathcal{J}'_{worse}$ and a new schedule $\mathcal{S}'$ where the corresponding job j'_i misses a deadline. Note that in schedule $\mathcal{S}'$, job costs are no smaller than in $\mathcal{S}$, suspension times can be defined arbitrarily (within the bounds of the task set), and all other job parameters (i.e., arrival time, deadline) remain unchanged.

4.2 Overview of the Proof Strategy

Having presented the main characteristics of the dynamic self-suspending task model, we now explain our proof strategy for establishing weak sustainability of uniprocessor JLFP scheduling of dynamic self-suspending tasks w.r.t. job costs and variable suspension times. For simplicity, the proof is based on the alternative definition of weakly-sustainable policy (Definition 24). According to Definition 24, we must prove that

$$
\begin{aligned}
&\forall\, \mathcal{J} \textbf{ s.t. } \mathcal{J} \text{ is } \textit{not} \text{ schedulable under uniprocessor JLFP scheduling,}\\
&\quad \forall\, \tau \in \mathcal{M},\ \forall \mathcal{J}_{worse} \in \textit{jobsets}(\tau) \textbf{ s.t.}\\
&\qquad \mathcal{J} \text{ and } \mathcal{J}_{worse} \text{ differ only by } S = \{cost\} \textbf{ and } \mathcal{J} \preceq_{cost} \mathcal{J}_{worse},\\
&\qquad\quad \exists\, \mathcal{J}'_{worse} \in \textit{jobsets}(\tau) \textbf{ s.t.}\\
&\qquad\qquad \mathcal{J}_{worse} \text{ and } \mathcal{J}'_{worse} \text{ differ only by } V = \{susp\} \textbf{ and}\\
&\qquad\qquad \mathcal{J}'_{worse} \text{ is not schedulable under uniprocessor JLFP scheduling.}
\end{aligned}
$$

That is, first we consider any job set $\mathcal{J}$ that is not schedulable and any job set $\mathcal{J}_{worse}$ that has "no better *job costs*" than $\mathcal{J}$ (and that is otherwise identical). Then, we must show that there exists a job set $\mathcal{J}'_{worse}$ generated by the same task set as $\mathcal{J}_{worse}$ that differs from $\mathcal{J}_{worse}$ only by its job *suspension times* and that it is not schedulable. In particular $\mathcal{J}$ and $\mathcal{J}_{worse}$ have equal suspension times (but not necessarily equal execution costs), whereas $\mathcal{J}_{worse}$ and $\mathcal{J}'_{worse}$ have equal execution costs (but not necessarily equal suspension times).

Our proof begins by considering any job set $\mathcal{J}$ and its associated schedule $\mathcal{S}$ where some job misses a deadline. Then, we construct a job set $\mathcal{J}'_{worse}$ together with its schedule $\mathcal{S}'$ where some job also misses a deadline. This strategy is illustrated in Fig. 2.

In the next section, we propose an algorithm for iteratively constructing schedule $\mathcal{S}'$ (and hence the associated job set $\mathcal{J}'_{worse}$) based on $\mathcal{S}$. It is followed by the two main proof obligations: (a) proving that some job misses a deadline in $\mathcal{S}'$ (§4.3.1) and (b) proving that $\mathcal{S}'$ is a valid schedule of $\mathcal{J}'_{worse}$ (§4.3.2). This proves that $\mathcal{J}'_{worse}$ is not schedulable.

4.3 Constructing $\mathcal{J}'_{worse}$ and Schedule $\mathcal{S}'$

Based on the strategy proposed in §4.2, we now present the algorithm to construct schedule $\mathcal{S}'$ and the associated job set $\mathcal{J}'_{worse}$, based on the original schedule $\mathcal{S}$. In the remainder of this paper, whenever we want to refer to the same job before and after the parameter transformation (*i.e.*, from $\mathcal{J}$ to $\mathcal{J}'_{worse}$), we refer to them as corresponding jobs j_i and j'_i.

First, recall that jobs in $\mathcal{J}'_{worse}$ have no better job costs than in $\mathcal{J}$, *i.e.*, for any corresponding jobs j_i and j'_i, $cost(j_i) \leq cost(j'_i)$. Since they might have to execute for different durations, we begin by defining the notion of added cost.

▶ **Definition** 30 (Added Cost). *We define the added cost $\Delta_{cost}(j'_i)$ of job j'_i in $\mathcal{J}'_{worse}$ as the difference between its original and inflated costs, i.e., $\Delta_{cost}(j'_i) = cost(j'_i) - cost(j_i) \geq 0$.*

In order to guarantee that schedule $\mathcal{S}'$ becomes as hard as schedule $\mathcal{S}$ (*i.e.*, so that jobs still miss their deadlines) and at the same time easy to compare in terms of received job service (*i.e.*, the time for which a job excuted since its release), we construct $\mathcal{S}'$ based on the idea of "picking jobs that are late with respect to $\mathcal{S}$", where late is defined as follows.

▶ **Definition** 31 (Late job). *We say that job j'_i is late in schedule S' at time t iff the service received by j'_i in S' up to time t is less than the service received by the corresponding job j_i in schedule $\mathcal{S}$ (compensated by the added cost), i.e.,*

$$service(j'_i, t) < service(j_i, t) + \Delta_{cost}(j'_i).$$

We now present the algorithm used to iteratively build schedule $\mathcal{S}'$ and job set J'_{worse}. Algorithm 1 ensures that (i) every job $j'_i \in \mathcal{J}'_{worse}$ executes for its total execution cost $cost(j'_i)$ ($\geq cost(j_i)$), (ii) every job $j'_i \in \mathcal{J}'_{worse}$ has a total suspension time $susp(j'_i)$ upper-bounded by the suspension time $susp(j_i)$ of its corresponding job in schedule $\mathcal{S}$, and (iii) at least one job of $\mathcal{J}'_{worse}$ misses its deadline in $\mathcal{S}'$ (as proven in Sec 4.3.1).

▶ **Algorithm** 1 (Construction of Job Set $\mathcal{J}'_{worse}$ and Schedule $\mathcal{S}'$). *Consider any time t and let $J(t)$ denote the set that contains every job j'_i that is pending (i.e., released, not completed, and not suspended) in schedule $\mathcal{S}'$ at time t and such that either (a) j'_i is late at time t or (b) the corresponding job j_i is scheduled in $\mathcal{S}$ at time t.*

1. **Schedule:** *We schedule in S' at time t the highest-priority job of $J(t)$, or idle the processor if $J(t)$ is empty.*
2. **Suspensions:** *Any job $j'_i \in \mathcal{J}'_{worse}$ suspends in $\mathcal{S}'$ at time t iff the corresponding job j_i is suspended in $\mathcal{S}$ and j'_i is not late.*

Note that Algorithm 1 not only picks late jobs, but (a) favors higher-priority jobs, and (b) tries to copy schedule $\mathcal{S}$ if possible. While rule (a) is required to ensure that the schedule respects the JLFP policy, rule (b) provides a tie-break rule if there are multiple jobs that can be picked, in which case we choose the same job as the job scheduled in $\mathcal{S}$.

It only remains to be shown that schedule $\mathcal{S}'$ results in a deadline miss (Theorem 2) and schedule $\mathcal{S}'$ does not violate any property of the scheduling policy, platform, and task model, such as work conservation, priority enforcement, *etc.* (Theorem 3).

But first, we emphasize that Algorithm 1 builds a schedule $\mathcal{S}'$ and hence a job set $\mathcal{J}'_{worse}$ that has different suspension times than the original job set $\mathcal{J}$. Therefore, the presented argument indeed proves the *weak*-sustainability of uniprocessor JLFP scheduling under the dynamic self-suspending task model w.r.t. job cost and *variable suspension time.*

4.3.1 Proving that $\mathcal{S}'$ Misses a Deadline

In order to prove that some job $j'_i \in jobsets'_{worse}$ misses a deadline in $\mathcal{S}'$, we first recall the concept of job service.

▶ **Definition 32** (Job Service). *Given a schedule $\mathcal{S}$, we define job service $service(j, t)$ as the cumulative service received by job j in the interval $[0, t)$.*

Then, based on the definition of service and the construction of $\mathcal{S}'$ (Algorithm 1), we can prove the following powerful invariant that relates the two schedules.

▶ **Lemma 1** (Service Invariant). *For any corresponding jobs $j_i \in \mathcal{J}$ and $j'_i \in \mathcal{J}'_{worse}$, at any time t, we have the service invariant that $service(j'_i, t) \leq service(j_i, t) + \Delta_{cost}(j'_i)$.*

Proof. Proven in Prosa [1]. Consider any pair of corresponding jobs j_i and j'_i. The proof follows by induction on time t.

1. **Base Case:** At time $t = 0$, jobs have received no service, thus $service(j'_i, 0) = 0 = service(j_i, t) \leq service(j_i, t) + \Delta_{cost}(j'_i)$.
2. **Inductive Step:** Assume as the induction hypothesis that, for some t, $service(j'_i, t) \leq service(j_i, t) + \Delta_{cost}(j'_i)$. Then we must prove $service(j'_i, t+1) \leq service(j_i, t+1) + \Delta_{cost}(j'_i)$. First, consider the simple case where job j'_i is not scheduled in $\mathcal{S}'$ at time t. Then,

$$\begin{aligned} service(j'_i, t+1) &= service(j'_i, t) && (j'_i \text{ is not scheduled in } \mathcal{S}' \text{ at } t) \\ &\leq service(j_i, t) + \Delta_{cost}(j'_i) && (\text{by induction hypothesis}) \\ &\leq service(j_i, t+1) + \Delta_{cost}(j'_i). && (\text{by monotonicity of service}) \end{aligned}$$

Otherwise, assume that j'_i is scheduled in $\mathcal{S}'$ at time t. From the schedule construction (Algorithm 1), it follows that either (a) $\mathcal{S}$ and $\mathcal{S}'$ schedule *corresponding jobs* at time t, or (b) $\mathcal{S}'$ schedules a late job at time t. We analyze both cases.

   a. **Corresponding Jobs are Scheduled:** The corresponding jobs scheduled in $\mathcal{S}$ and $\mathcal{S}'$ at time t must be j_i and j'_i, so

$$\begin{aligned} service(j'_i, t+1) &= service(j'_i, t) + 1 && (j'_i \text{ is scheduled in } \mathcal{S}' \text{ at time } t) \\ &\leq service(j_i, t) + \Delta_{cost}(j'_i) + 1 && (\text{by induction hypothesis}) \\ &= service(j_i, t+1) + \Delta_{cost}(j'_i) && (j_i \text{ is scheduled in } \mathcal{S} \text{ at time } t). \end{aligned}$$

   b. **Late Job:** Job j'_i must be the highest-priority late job in $\mathcal{S}'$ at time t. By the definition of late job (Definition 31), it follows that $service(j'_i, t) < service(j_i, t) + \Delta_{cost}(j'_i)$, so

$$\begin{aligned} service(j'_i, t+1) &= service(j'_i, t) + 1 && (j'_i \text{ is scheduled in } \mathcal{S}' \text{ at time } t) \\ &< service(j_i, t) + \Delta_{cost}(j'_i) + 1 && (\text{by assumption}) \\ &\leq service(j_i, t) + \Delta_{cost}(j'_i). && (\text{by converting } < \text{ to } \leq) \end{aligned}$$

The claim holds in all cases, which concludes the proof by induction. ◀

Since we must prove that schedule $\mathcal{S}'$ results in a deadline miss, we use the service invariant above to conclude that jobs complete earlier in $\mathcal{S}$ than in $\mathcal{S}'$.

▶ **Corollary 4** (Jobs Complete Earlier in $\mathcal{S}$). *For any corresponding jobs $j_i \in \mathcal{J}$ and $j'_i \in \mathcal{J}'_{worse}$, if j'_i has completed in schedule $\mathcal{S}'$ by time t, then j_i has completed in $\mathcal{S}$ by time t.*

Proof. Proven in Prosa [1]. Follows from Lemma 1, since j_i receives enough service in $\mathcal{S}$ to complete before the corresponding j'_i in $\mathcal{S}'$. ◀

Recall that we initially assumed that some job misses a deadline in $\mathcal{S}$. We can thus conclude that the corresponding job also misses a deadline in $\mathcal{S}'$.

▶ **Theorem** 2 (Deadline Miss). *There exists a job $j_i' \in \mathcal{J}_{worse}'$ that misses a deadline in $\mathcal{S}'$.*

Proof. Proven in Prosa [1]. The proof follows by contradiction. Assume that there is no deadline miss in $\mathcal{S}'$. Next, recall that there exists a job j_i in schedule $\mathcal{S}$ that misses a deadline. However, by Corollary 4, if the corresponding job j_i' completed on time in $\mathcal{S}'$, then it must have completed no later in the original schedule $\mathcal{S}$. Since j_i and j_i' differ only in execution cost and suspension times, but not in their deadlines, this implies that j_i cannot have missed a deadline in $\mathcal{S}$, which is a contradiction. ◀

4.3.2 Proving that $\mathcal{S}'$ is a Valid Schedule

Although we have already established the non-schedulability of the generated schedule $\mathcal{S}'$, it remains to be shown that schedule $\mathcal{S}'$ is valid and compatible with the task model.

▶ **Theorem** 3 (Valid Schedule). *Schedule $\mathcal{S}'$ is a valid uniprocessor schedule of job set $\mathcal{J}_{worse}'$ assuming JLFP scheduling of sporadic, dynamic self-suspending tasks.*

Proof. Proven in Prosa [1]. Follows from Algorithm 1, since suspension intervals in schedule $\mathcal{S}'$ are no longer than those in $\mathcal{S}$ and the fact that the dynamic self-suspension model imposes only an upper bound on total job suspension time, and since by construction the derived schedule $\mathcal{S}'$ is work-conserving, respects self-suspensions, and respects job priorities. ◀

4.4 Main Claim

Based on the strategy explained in §4.2, by combining Theorems 2 and 3, we prove that the scheduling policy is weakly-sustainable.

▶ **Theorem** 4 (Weak Sustainability). *Uniprocessor JLFP scheduling of sporadic self-suspending tasks under the dynamic suspension model is weakly-sustainable with respect to job costs and variable suspension times.*

Proof. Proven in Prosa [1]. Instantiate Definition 24 with uniprocessor JLFP scheduling of sporadic self-suspending tasks under the dynamic suspension model for $S = \{cost\}$ and $V = \{susp\}$. Theorems 2 and 3 imply that for any schedule $\mathcal{S}$ of job set $\mathcal{J}$ that misses a deadline, there exists a schedule $\mathcal{S}'$ of the transformed job set $\mathcal{J}_{worse}'$ that misses a deadline. ◀

5 Conclusion and Future Work

We have identified that the existing notions of sustainability in real-time scheduling allow for multiple interpretations on whether real-time scheduling of self-suspending tasks is sustainable. To resolve the issue, we developed a precise sustainability theory for real-time scheduling that is compatible with any task and platform model (§2), and also proposed the new notions of strongly- and weakly-sustainable policies (§3), which can be used to derive less pessimistic schedulability analyses for policies that were shown to not be strongly-sustainable.

To better understand a model for which many mistakes were found in the literature [9], we chose to study weak sustainability in the context of self-suspending tasks. For that, we developed a generic model for self-suspensions (§4.1) that was formalized in the Coq proof assistant and integrated into Prosa [8, 1]. Finally, we mechanically proved in Prosa that uniprocessor JLFP scheduling of self-suspending tasks is weakly-sustainable with respect to job costs and variable suspension times (§4.2–§4.4).

In ongoing work, we are working towards leveraging the obtained weak sustainability result to derive new, mechanized schedulability tests for the dynamic suspension model.

References

1 Prosa – Weak Sustainability. Supplemental material and formal proofs, `http://prosa.mpi-sws.org/releases/sustainability`.

2 Yasmina Abdeddaïm and Damien Masson. The scheduling problem of self-suspending periodic real-time tasks. In *Proc. of the 20th International Conference on Real-Time and Network Systems*, RTNS'12, pages 211–220, 2012.

3 Theodore P. Baker and Sanjoy K. Baruah. Sustainable multiprocessor scheduling of sporadic task systems. In *Proc. of the 21st Euromicro Conference on Real-Time Systems*, ECRTS '09, pages 141–150, 2009.

4 Sanjoy Baruah. Scheduling periodic tasks on uniform multiprocessors. *Information Processing Letters*, 80(2):97–104, 2001.

5 Sanjoy Baruah. Feasibility analysis of preemptive real-time systems upon heterogeneous multiprocessor platforms. In *Real-Time Systems Symposium, 2004. Proceedings. 25th IEEE International*, pages 37–46. IEEE, 2004.

6 Sanjoy Baruah and Alan Burns. Sustainable scheduling analysis. In *Proc. of the 27th Real-Time Systems Symposium*, RTSS '06, pages 159–168, 2006.

7 Alan Burns and Sanjoy Baruah. Sustainability in real-time scheduling. *Journal of Computing Science and Engineering*, 2(1):74–97, 2008.

8 Felipe Cerqueira, Felix Stutz, and Björn B Brandenburg. Prosa: A case for readable mechanized schedulability analysis. In *Real-Time Systems (ECRTS), 2016 28th Euromicro Conference on*, pages 273–284. IEEE, 2016.

9 Jian-Jia Chen, Geoffrey Nelissen, Wen-Hung Huang, Maolin Yang, Björn Brandenburg, Konstantinos Bletsas, Cong Liu, Pascal Richard, Frédéric Ridouard, Neil Audsley, Raj Rajkumar, and Dionisio de Niz. Many suspensions, many problems: A review of self-suspending tasks in real-time systems. Technical Report 854, Department of Computer Science, TU Dortmund, 2016.

10 Sudarshan K. Dhall and C. L. Liu. On a real-time scheduling problem. *Oper. Res.*, 26(1):127–140, 1978.

11 Rhan Ha. Validating timing constraints in multiprocessor and distributed systems. 1995.

12 Rhan Ha and Jane WS Liu. Validating timing constraints in multiprocessor and distributed real-time systems. In *Distributed Computing Systems, 1994., Proceedings of the 14th International Conference on*, pages 162–171. IEEE, 1994.

13 In-Guk Kim, Kyung-Hee Choi, Seung-Kyu Park, Dong-Yoon Kim, and Man-Pyo Hong. Real-time scheduling of tasks that contain the external blocking intervals. In *Real-Time Computing Systems and Applications, 1995. Proceedings., Second International Workshop on*, pages 54–59. IEEE, 1995.

14 Chung Laung Liu and James W Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

15 Aloysius Ka-Lau Mok. *Fundamental design problems of distributed systems for the hard-real-time environment*. PhD thesis, Massachusetts Institute of Technology, 1983.

16 Ragunathan Rajkumar. Dealing with suspending periodic tasks. *IBM Thomas J. Watson Research Center*, 1991.

17 Dongkun Shin and Jihong Kim. A profile-based energy-efficient intra-task voltage scheduling algorithm for hard real-time applications. In *Low Power Electronics and Design, International Symposium on, 2001.*, pages 271–274, 2001.