Analysis Techniques for Supporting Harmonic Real-Time Tasks with Suspensions

Cong Liu[†], Jian-Jia Chen[§], Liang He[‡], Yu Gu[‡]
[†]The University of Texas at Dallas, USA
[§]Karlsruhe Institute of Technology (KIT), Germany
[‡]Singapore University of Technology and Design, Singapore

Abstract

In many real-time systems, tasks may experience suspension delays when they block to access shared resources or interact with external devices such as I/O. It is known that such suspensions delays may negatively impact schedulability. Particularly in hard real-time systems, a few negative results exist on analyzing the schedulability of such systems, even for very restricted suspending task models on a uniprocessor.

In this paper, we focus on the particular case of hard real-time suspending task systems with harmonic periods, which is a special case of practical relevance. We propose a new uniprocessor suspension-aware analysis technique for supporting such task systems under rate-monotonic scheduling. Our analysis technique is able to achieve only $\Theta(1)$ *suspension-related utilization loss on a uniprocessor.Based upon this technique, we further propose a partitioning scheme that supports suspending task systems with harmonic periods on multiprocessors. The resulting schedulability test shows that compared to existing schedulability tests designed for ordinary non-suspending task systems, suspensions only results in* $\Theta(m)$ *additional suspension-related utilization loss, where* m *is the number of processors. Furthermore, experiments presented herein show that both our uniprocessor and multiprocessor schedulability tests improve upon prior approaches by a significant margin.*

1 Introduction

In many real-time systems, suspension delays may occur when tasks block to access shared resources or interact with external devices such as I/O. Such delays can be quite lengthy (e.g., $15ms$ for a disk read), in which schedulability in real-time systems is negatively impacted. It has been shown that precisely analyzing hard real-time (HRT) systems with suspensions is difficult, even for very restricted suspending task models on uniprocessors [22]. We consider this problem in the context of analyzing HRT suspending task systems with harmonic periods, where the periods of the tasks pairwise divide each other. Task systems with harmonic periods are seen in a number of application domains [2, 3, 6, 11, 18]. It is know that for ordinary sporadic and periodic task systems (without suspensions), harmonic periods may allow a larger system utilization [9, 17]. In this paper, we show that system utilization can be significantly improved for suspending task systems with harmonic periods under rate-monotonic (RM) scheduling on both a uniprocessor and a multiprocessor.

For analyzing suspensions, there exist two major categories of techniques, *suspension-oblivious* v.s. *suspension-aware* analysis. Under suspension-oblivious analysis (which is perhaps the most commonly used approach), suspensions are simply integrated into per-task worst-case execution time requirements. However, this approach clearly yields $\Theta(n)$ utilization loss, where n is the number of suspending tasks in the system. Unless n is small and suspension delays are short, this approach may sacrifice significant system capacity. The alternative is to employ *suspension-aware* analysis, where suspensions are explicitly considered in the task model and resulting schedulability analysis. On a uniprocessor, suspension-aware analysis techniques that are correct in a sufficiency sense have been proposed [8, 19–21, 24]. However, these analysis can be quite pessimistic in many cases. On multiprocessors, the only existing suspension-aware analysis is the one proposed in [16]. This analysis is an improvement over suspension-oblivious analysis for many task systems, but does not fully address the root cause of pessimism due to suspensions, and thus may still cause significant utilization loss. Indeed, suspensions are hard to analyze because they may leave the processor idle and it is impossible to predict when suspensions may occur in the runtime schedule. This causes pessimism in the analysis because we have to assume that all suspending tasks suspend concurrently at any idle time instant.

In this paper, we focus on studying periodic suspending task systems with harmonic periods. This class of task systems is appealing from a practical point of view as harmonic periods are seen in many application domains such as avionics [1]. For ordinary periodic task systems without suspensions, a well-known result is that RM scheduling on

a uniprocessor is not optimal for non-harmonic periods but becomes optimal for harmonic periods. Attempts have also been made to extend such results for systems with harmonic periods to other system models. For example, the utilization bound of uniprocessor RM scheduling has been generalized using the notion of harmonic chains [9] and techniques have been proposed to transform task periods to make them nearly harmonic [23]. However, in the case of suspending task systems with harmonic periods, no schedulability analysis techniques and tests exists.

Overview of related work. Recently the problem of scheduling soft real-time (with guaranteed bounded response times) suspending task systems on multiprocessor has received much attention [12–15]. In our recent work [15], we proposed a transformation technique that converts a portion of certain tasks' suspensions into computation, which results in a much improved schedulability test for soft real-time self-suspending task systems. The essential idea behind the uniprocessor analysis technique proposed in this paper (Sec. 4) is mainly motivated by this prior technique [15], which is to convert the analyzed task's suspensions into computation such that the cumbersomeness on analyzing suspensions is eliminated. However, different from the technique presented in [15] that is designed for soft real-time suspending task systems, the partial suspension-to-computation conversion technique presented in Sec. 4 is tailored to handle the HRT case.

For the HRT case, besides the suspension-oblivious approach of treating all suspensions as computation, several schedulability tests have been presented for analyzing periodic tasks that may suspend at most once on a uniprocessor [8, 10, 19–21, 24]. Unfortunately, these tests are rather pessimistic as their techniques involve straightforward execution control mechanisms, which modify task deadlines (often known as the *end-to-end* approach [4, 17]). For example, a suspending task that suspends once can be divided into two subtasks with appropriately shorted deadlines and modified release times. Such techniques inevitably suffer from significant capacity loss due to the artificial shortening of deadlines. On multiprocessors, [16] presents the only existing global suspension-aware analysis for periodic suspending task systems scheduled under global EDF and global fixed-priority schedulers. This test improves upon the suspension-oblivious approach for some task systems. Generally speaking, however, these two tests are incomparable.

Contributions. In this paper, we present sufficient schedulability tests for scheduling periodic suspending task systems with harmonic periods under RM on a uniprocessor and on a multiprocessor, respectively. On a uniprocessor, our analysis technique yields a new utilization-based schedulability test (Theorem 1), $\max_k \left\{ \sum_{i=1}^{k} u_i + \frac{s_k}{p_k} \right\} \leq 1$, where u_k, s_k and p_k denote task τ_k's utilization, suspension length, and period, respectively. This test results in only $\Theta(1)$ suspension-related utilization loss and theoretically dominates the suspension-oblivious approach. Then based upon this uniprocessor test, we present a partitioning scheme for solving the multiprocessor case. The derived schedulability test (Theorem 2) shows that any periodic harmonic task system is schedulable if $U_{sum} \leq m - U_{m-1} - V_m$, where U_{sum} is the total system utilization, m is the number of processors, U_{m-1} is the sum of $m-1$ largest task utilizations, and V_m is the sum of m largest task suspension ratio, where a task's *suspension ratio* is given by the ratio of its suspension time over its period. This result shows that on a multiprocessor, the negative impact due to suspensions on HRT schedulability can be limited to an order of $\Theta(m)$ suspension-related utilization loss.[1] As demonstrated by experiments, our proposed tests improve upon prior methods with respect to schedulability, and are often able to guarantee schedulability with little or no utilization loss.

The rest of this paper is organized as follows. In Secs. 2 and 3, we present the system model and a suspending task set with a minimum utilization that exhibits the worst-case behavior due to suspensions. In Sec. 4, we present our $\Theta(1)$ analysis technique and the corresponding uniprocessor schedulability test. Then, in Sec. 5, we present our proposed partitioning scheme and the corresponding multiprocessor schedulability test. In Sec. 6, we experimentally compare our proposed schedulability tests with prior methods. In these experiments, our tests exhibited superior performance, typically by a wide margin. We conclude in Sec. 7.

2 System Model

We model a given real-time system as a set $\tau = \{\tau_1, ..., \tau_n\}$ of n synchronous periodic suspending tasks with harmonic periods scheduled on $m \geq 1$ identical processors $M_1, M_2, ..., M_m$. Tasks are indexed by periods, i.e., $p_i \leq p_{i+1}$, and ties are broken arbitrarily. A task system is harmonic if and only if the periods of its tasks are pairwise divisible. A task system is synchronous if all n tasks in the system release their first jobs at the same time. Each task is released repeatedly, with each such an invocation called a job. Jobs alternate between computation and suspension phases. We assume that each job of τ_i executes for at most e_i time units (across all of its execution phases) and suspends for at most s_i time units (across all of its suspension phases). We place no restrictions on how these phases interleave (a job can even begin or end with a suspension phase). Different jobs belong to the same task can also have different phase-interleaving patterns. For many applications, such a general suspension model is needed due to the unpredictable nature of I/O operations. The j^{th} job of τ_i, denoted by $\tau_{i,j}$, is released at time $r_{i,j}$ and has a deadline at time $d_{i,j}$. Suc-

[1] This claim is obtained by comparing our test to a known multiprocessor schedulability test, namely, $U_{sum} \leq m - (m-1) \cdot u_{max}$, where u_{max} is the maximum task utilization in the system [7].

m	Number of processors
M_i	i^{th} processor
n	Number of tasks
v_i	Suspension ratio of task τ_i
V_m	Sum of the m largest task suspension ratios
U_{sum}	Utilization of task system τ
U_{m-1}	Sum of the $m-1$ largest task utilizations

Table 1: Summary of notation.

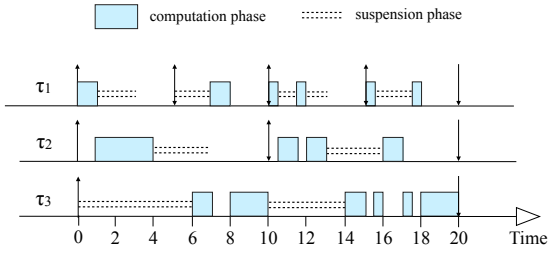


Figure 1: Example task system.

cessive jobs of the same task are required to execute in sequence. Associated with each task τ_i are a period p_i, which specifies the exact time between two consecutive job releases of τ_i, and a deadline d_i, which specifies the relative deadline of each such job, i.e., $d_{i,j} = r_{i,j} + d_i$. The utilization of a task τ_i is defined as $u_i = e_i/p_i$, and the utilization of the task system τ as $U_{sum} = \sum_{\tau_i \in \tau} u_i$. The suspension ratio of a task τ_i is defined to be $v_i = \dfrac{s_i}{p_i}$. An SSS task system τ is said to be an implicit-deadline system if $d_i = p_i$ holds for each τ_i. Due to space constraints, we limit attention to implicit-deadline suspending task systems in this paper.

We focus on RM scheduling, where tasks are prioritized by their periods. We assume that ties are broken by task ID (lower IDs are favored). According to our task indexing rule, we know that task τ_i has a higher priority than any task τ_k where $i < k$. A summary of the terms defined so far, as well as some additional terms defined later, is presented in Table 1.

Example A uniprocessor RM schedule of a periodic suspending task system with harmonic periods is shown in Fig. 1. This task system contains three suspending tasks, $\tau_1(1, 2, 5)$, $\tau_2(3, 3, 10)$, and $\tau_3(7, 10, 20)$, where $\tau_i(e_i, s_i, p_i)$ is used to characterize a task. As seen in Fig. 1, different jobs of the same task can have different interleaving patterns of computation and suspension phases.

3 Worst-Case Behavior due to Suspensions

In this section, we present a suspending task set with a minimum utilization that exhibits the worst-case behavior due to suspensions, where all tasks have long suspensions that occur in parallel and thus cause deadline misses.

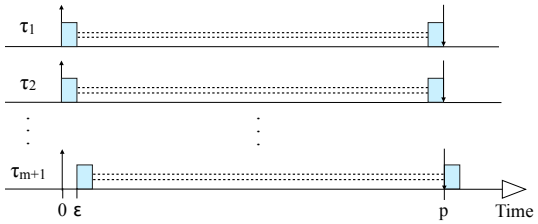


Figure 2: Schedule of the worst-case suspending task set.

Consider a task set containing $m+1$ identical suspending tasks. Each task has a period of p time units, and first executes for ε time units, then suspends for $p - 2 \cdot \varepsilon$ time units, and finally executes for another ε time units, where ε can be arbitrarily small. A schedule of this task set is shown in Fig. 2. Regardless of which scheduling algorithms we use, it is impossible to meet all deadlines of this task set on m processors. Note that this task set has a total utilization of $(m+1) \cdot \dfrac{2 \cdot \varepsilon}{p}$, which can be arbitrarily small.

As seen in Fig. 2, all $m+1$ tasks have a suspension length of $p - 2 \cdot \varepsilon$ and all these suspensions occur concurrently. These suspensions force jobs to be denied processor time, which reduce the time available for their completions. This leads to a task not behaving like a periodic task because it may demand more execution time over some interval than a periodic task. For this task set, within time interval $[p-\varepsilon, p)$, a total demand of $(m+1) \cdot \varepsilon$ is requested by all tasks in order for them to meet deadlines, which is clearly infeasible.

Due to the fact that we cannot precisely predict when a job's suspension may occur, we have to consider such worst-case behaviors in the analysis. That is, when one task suspends at some time t, all tasks may suspend at the same time at t and thus cause t to be idle. Interestingly, the suspension-oblivious approach is able to eliminate this worst-case behavior at the cost of pessimism by converting all n tasks' suspensions into computation. Next, we present a uniprocessor suspension-aware analysis technique that can avoid such worst-case behaviors at the cost of converting only one task's suspension into computation, and thus yields a much improved schedulability test.

4 Uniprocessor Schedulability Analysis

We now present our proposed suspension-aware schedulability analysis on a uniprocessor. We first describe the proof setup, then present a new analysis technique, and finally derive a utilization-based schedulability test.

Definition 1. A job is considered to be *completed* if it has finished its last phase (be it suspension or computation). A job is *pending* at time t if it is released before t but has not completed by t.

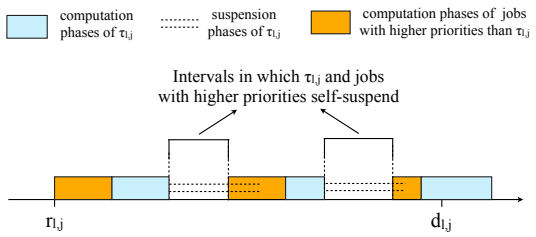


Figure 3: Our analyzed interval $[r_{l,j}, d_{l,j})$.

Definition 2. If a job is enabled and not suspended at time t but does not execute at t, then it is *preempted* at t.

Let S be an RM schedule of τ such that a job $\tau_{l,j}$ of task τ_l is the first job in S to miss its deadline at $d_{l,j}$, as shown in Fig 3. Under RM, tasks with lower priorities than τ_l do not affect the scheduling of τ_l and tasks with higher priorities than τ_l, so we will henceforth discard from S all tasks with priorities lower than τ_l (i.e., any task τ_k where $k > l$).

Our objective is to determine conditions necessary for $\tau_{l,j}$ to miss its deadline, i.e., for $\tau_{l,j}$ to execute its computation and suspension phases for strictly fewer than $e_l + s_l$ time units over $[r_{l,j}, d_{l,j})$.

As discussed in Sec. 3, the pessimism of analyzing suspending task systems is due to the worst-case scenario where all suspending tasks might have enabled jobs that suspend concurrently. For ordinary periodic task systems without suspensions, if job $\tau_{l,j}$ misses its deadline, then it must be the case that the total demand placed in $[r_{l,j}, d_{l,j})$ due to $\tau_{l,j}$ and jobs with higher priorities than $\tau_{l,j}$ exceeds the available processor capacity in this interval (i.e., p_l). For suspending tasks, unfortunately, this fact no longer holds. As seen in Fig. 3, due to suspensions, the total amount of such work does not need to exceed p_l in order for $\tau_{l,j}$ to miss its deadline.

Therefore, our goal is to avoid such a worst case. Our key observation is that $\tau_{l,j}$ must suspend at any non-busy time $t \in [r_{l,j}, d_{l,j})$ since it does not complete by $d_{l,j}$. The key idea behind our new technique is the following: *Treating $\tau_{l,j}$'s suspensions happening within idle time intervals in $[r_{l,j}, d_{l,j})$ as computation forces $[r_{l,j}, d_{l,j})$ to be a busy interval. Doing so avoids the need of analyzing suspensions and thus eliminates the analytical pessimism due to suspensions, at the expense of treating only one task's suspensions as computation.*

Our new technique involves transforming the schedule S partially within $[r_{l,j}, d_{l,j})$. The goal of this transformation is to convert $\tau_{l,j}$'s suspensions into computation in non-busy intervals to eliminate idleness as discussed above. Since $\tau_{l,j}$ misses its deadline, $\tau_{l,j}$ must suspend in all non-busy interval within $[r_{l,j}, d_{l,j})$. Thus, within all such non-busy intervals, we convert $\tau_{l,j}$'s suspensions into computation. Note that if $\tau_{l,j}$'s suspensions are converted into computation in a time interval $[t_1, t_2)$, then $\tau_{l,j}$ is considered to execute in $[t_1, t_2)$.

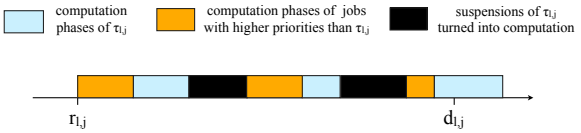


Figure 4: Convert suspensions of $\tau_{l,j}$ within all idle intervals in $[r_{l,j}, d_{l,j})$.

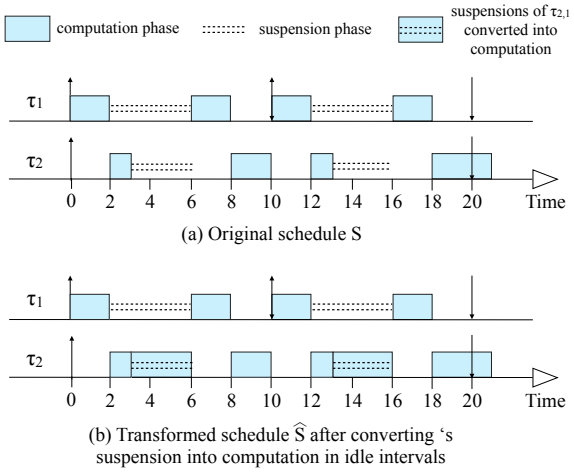


Figure 5: Example schedule and the corresponding transformed schedule.

Due to the fact that $\tau_{l,j}$ misses its deadline, interval $[r_{l,j}, d_{l,j})$ consists of four types of subintervals: *(i)* those in which $\tau_{l,j}$ is executing, *(ii)* those in which $\tau_{l,j}$ is preempted, *(iii)* those in which $\tau_{l,j}$ is suspending and some other job is executing, and *(iv)* those in which $\tau_{l,j}$ and (possibly) other enabled jobs are suspending. Note that only intervals of the last type are idle. Within all such idle time intervals in $[r_{l,j}, d_{l,j})$, convert suspensions of $\tau_{l,j}$ into computation, as illustrated in Fig. 4. Let $\widehat{S}$ denote the transformed schedule.

Example Fig. 5(a) shows an RM schedule S of an example task system consisting of two tasks $\tau_1(4, 4, 10)$ and $\tau_2(7, 6, 20)$. Although this task system has a total utilization less than 1, job $\tau_{2,1}$ misses its deadline at time 20 because its suspensions reduce the time available for its completion. Fig. 5(b) shows the corresponding schedule $\widehat{S}$ after applying the transformation method. As seen in Fig. 5(b), the transformation method is able to eliminate all idle intervals due to suspensions. Job $\tau_{2,1}$ misses its deadline exactly because the total amount of computation (including both original and converted computation) demanded in interval $[r_{2,1}, d_{2,1})$ exceeds the interval length, which is similar to the ordinary periodic task case.

Analysis. Since $\tau_{l,j}$ misses its deadline, we know that it must suspend within all non-busy intervals in $[r_{l,j}, d_{l,j})$ in the original schedule S. According to the our transformation method, $\tau_{l,j}$'s suspensions are converted into compu-

tation within all non-busy intervals in $[r_{l,j}, d_{l,j})$ in $\widehat{S}$. The following property of schedules $\widehat{S}$ thus follows.

Property 1. *$[r_{l,j}, d_{l,j})$ in $\widehat{S}$ is a busy interval.*

The following theorem gives a sufficient RM schedulability test.

Theorem 1. *An HRT synchronous periodic harmonic suspending task system τ with $U_{sum} \leq 1$ is schedulable under RM if*

$$\max_k \left\{ \sum_{i=1}^{k} u_i + \frac{s_k}{p_k} \right\} \leq 1 \tag{1}$$

holds.

Proof. We prove the theorem by contrapositive: if τ is not schedulable, then $\max_k \left\{ \sum_{i=1}^{k} u_i + \frac{s_k}{p_k} \right\} > 1$.

Assume that τ is not schedulable. Let $\tau_{l,j}$ be the first job that misses its deadline in the corresponding schedule S. We first transform S to $\widehat{S}$ by applying the transformation technique discussed earlier. By Property 1, we know that $[r_{l,j}, d_{l,j})$ in $\widehat{S}$ is a busy interval. Therefore, in order for $\tau_{l,j}$ to miss its deadline, the amount of work due to $\tau_{l,j}$ and jobs with higher priorities placed in $[r_{l,j}, d_{l,j})$, denoted by W, must exceed p_l. That is, $W > p_l$ is a necessary condition for $\tau_{l,j}$ to miss its deadline.

Since tasks have harmonic periods and jobs are released in a synchronous periodic manner, the number of jobs released by any task τ_k with $p_k \leq p_l$ within $[r_{l,j}, d_{l,j})$ is given by $\frac{p_l}{p_k}$. Note that under RM and our task indexing policy (defined in Sec. 2), tasks with IDs $> l$ have lower priorities than τ_l and thus do not contribute to W.

For task τ_l, the amount of work it contributes to W is at most $e_l + s_l$, where s_l represents the maximum amount of suspensions of $\tau_{l,j}$ that can be converted into computation. Therefore, we have

$$\begin{aligned} p_l <& W \\ \leq& \sum_{k=1}^{l-1} \frac{p_l}{p_k} \cdot e_k + (e_l + s_l) \\ =& \sum_{k=1}^{l} u_k \cdot p_l + s_l \end{aligned}$$

By rearrangements, we have

$$\begin{aligned} 1 <& \sum_{i=1}^{l} u_i + \frac{s_l}{p_l} \\ \leq& \max_k \left\{ \sum_{i=1}^{k} u_i + \frac{s_k}{p_k} \right\} \end{aligned}$$

Thus the theorem follows. □

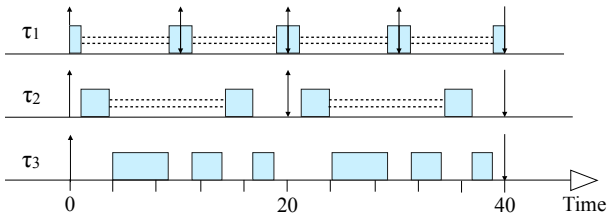


Figure 6: RM schedule of an example task system with $U_{sum} = 1$.

Theoretical dominance over the suspension-oblivious approach. We now show that our derived schedulability test theoretically dominates the suspension-oblivious approach. When using the approach of treating all suspensions as computation, the resulting utilization constraint for harmonic task systems on a uniprocessor under RM is given by $U_{sum} + \sum_{i=1}^{n} \frac{s_i}{p_i} \leq 1$ [17]. Clearly the constraint $\max_k \left\{ \sum_{i=1}^{k} u_i + \frac{s_k}{p_k} \right\} \leq 1$ in Theorem 1 is less restrictive than this prior approach. As will be seen in Sec. 6, our proposed schedulability test is able to support many task systems with little or even no utilization loss. Next we show that an example task system with $U_{sum} = 1$ that satisfies Eq. (1) is schedulable.

Example Consider a task system consisting of three suspending tasks $\tau_1(2, 8, 10)$, $\tau_2(6, 10, 20)$, and $\tau_3(20, 40)$. This task system has a total utilization equal to 1 and is schedulable since it satisfies Eq. (1). Fig. 6 shows an example RM schedule of this task system.

Negative impact due to asynchronous (or sporadic) releases, arbitrary periods, and EDF scheduling. Theorem 1 applies to synchronous periodic suspending task systems with harmonic periods under RM scheduling. We comment here why it is non-trivial to apply this technique to more general task models and EDF scheduling.

The negative impact due to asynchronous (or sporadic) releases is that since we do not have a deterministic releasing pattern, we do not know the worst-case behavior of a suspending task system. For ordinary task systems without suspensions, we know that the worst-case behavior under RM scheduling occurs at a well-defined critical instant, where all higher-priority tasks release their first jobs at the same time [17]. However, the critical instant theorem does not hold for suspending task systems. This forces us to always consider the worst-case suspending pattern in the analysis. Consider an example schedule shown in Fig. 7, where a task τ_i with $p_i = p_k$ releases two jobs whose execution windows (i.e., from job release to job deadline) overlap with our analyzed interval $[r_{l,j}, d_{l,j}]$. In this example, we have to assume that job $\tau_{i,k}$ first suspends and carries all its computation into the interval, and job $\tau_{i,k}$ first executes its computation after

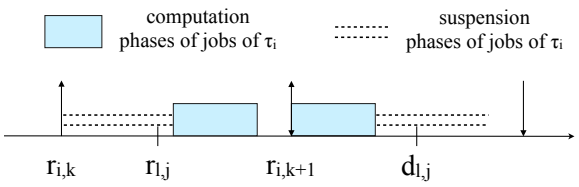


Figure 7: Asynchronous releases cause problems.

its release. Thus in this case, τ_i contributes $\left(\frac{p_i}{p_k}+1\right)\cdot e_i$ instead of $\frac{p_i}{p_k}\cdot e_i$ to W, which will cause the utilization condition to be rather pessimistic.

The case of arbitrary periods causes similar problems as sporadic releases under RM scheduling, as the amount of work contributed to W by a higher priority task cannot be efficiently bounded.

EDF scheduling may also cause significant pessimism in the analysis. The fundamental reason is due to an invalid property of the "idle instant" in the presence of suspensions. Recall that in the original EDF proof [17], the utilization-based test ($U_{sum} \leq 1$) utilizes an important property of an idle instant, i.e., no job is pending at any idle instant, which certainly holds for ordinary tasks without suspensions. Unfortunately, jobs that suspend at an idle instant can still carry computation work into later intervals. This again causes the amount of competing work due to jobs with higher priorities than $\tau_{l,j}$ to be inefficiently bounded.

Due to these observations, we identify the issue of supporting more general task models as open problems and leave them as future work.

5 Self-Suspending Task Partitioning on Multiprocessors

In this section, we propose a partitioning approach for supporting suspending task systems on multiprocessors. Although partitioning suffers from bin-packing-related utilization loss, previous research [5] has shown that it is preferable to global scheduling for hard real-time task systems, in which most industrial systems also adopt such partitioning approaches. A major reason is due to the optimality of EDF on a uniprocessor for ordinary task systems without suspensions. In our context, partitioning also appears to be an appealing choice due to the fact that Theorem 1 ensures little utilization loss in many cases on a uniprocessor. Next, we present an algorithm *SSPartition* that efficiently partitions suspending tasks on a multiprocessor based upon Theorem 1. We then analyze the properties of *SSPartition* and derive a corresponding schedulability test.

The utilization loss seen in the utilization constraint $\max_k \left\{\sum_{i=1}^{k} u_i + \frac{s_k}{p_k}\right\} \leq 1$ (Eq. (1)) is mainly caused by large values of p_i and $\frac{s_i}{p_i}$ (i.e., large periods and suspension ratios). A large value of p_i implies a large value of $\sum_{j=1}^{i} u_j$ because tasks with shorter periods than p_i contribute to $\sum_{j=1}^{i} u_j$; while a large value of $\frac{s_i}{p_i}$ may also lead to a large value of the left-hand side of Eq. (1). An interesting observation on Eq. (1) is that on a single processor, among the tasks with large p_i and/or $\frac{s_i}{p_i}$ values, the utilization loss due to the term $\frac{s_i}{p_i}$ in Eq. (1) is caused by only one such task. If we can partition x tasks onto one processor, such undesirable properties of $x-1$ of them can actually be "masked" by the one task that yields the maximum value of the left-hand side of Eq. (1). Motivated by this, Algorithm *SSPartition* always seeks to assign tasks with long periods and large suspension ratios to the same processor, as described next.

Definition 3. Let $\delta(\tau_i, M_k)$ denote the difference between values of the left-hand side of Eq. (1) for processor M_k before and after assigning task τ_i to that processor. Let $u(M_j)$ denote the available utilization of M_j.

Algorithm description. Algorithm *SSPartition* partitions tasks in a suspending task system τ onto m processors. The pseudocode is shown in Fig. 8. Tasks are ordered and re-indexed according to non-increasing suspension ratio.[2] Then for each task τ_i in order, we find a set of used processors[3] onto which τ_i can be assigned (lines 3-7). We then assign τ_i to one such processor M_j that results in the smallest $\delta(\tau_i, M_j)$ (lines 8-10). As discussed earlier, the intuition of doing this is to always partition tasks with large suspension ratios to the same processor so that (ideally) on each processor M_j, tasks assigned to M_j with large suspension ratios can be "masked" by the one task assigned to M_j that has the largest suspension ratio. If a task cannot be assigned to any used processor, then it is assigned to a new processor on which no task has been assigned (lines 12-13). The algorithm returns failure if a task cannot be assigned to any processor (line 11). *SSPartition* clearly yields a polynomial-time complexity of $\Theta(n \cdot logn + n \cdot m)$.

Example Consider a two-processor suspending task system consisting of six tasks: $\tau_1(1,4,5)$, $\tau_2(3,5,10)$, $\tau_3(2,4,10)$, $\tau_4(1,2,5)$, $\tau_5(12,0,20)$, $\tau_6(10,0,20)$ (ordered according to Algorithm *SSparition*). By applying this algorithm, tasks τ_1, τ_2, and τ_6 are assigned to processor 1 and τ_3, τ_4, and τ_5 are assigned to processor 2. This task system can be successfully partitioned onto two processors even if it has a total utilization of two. This is mainly due to the fact that τ_1 and τ_2 which are the two tasks with the largest suspension ratio are assigned to the same processor.

[2]Note that after partitioning tasks onto processors, on each processor, tasks are again indexed by periods. This is because this indexing rule is required by Theorem 1 in order to check the schedulability on each individual processor using Eq. 1.

[3]A used processor is one to which at least one task has been assigned.

SSPARTITION(τ, m)

▷ A set of periodic suspending tasks with harmonic periods $\tau = \{\tau_1, \tau_2, ..., \tau_n\}$ is to be partitioned on m processors denoted by $M_1, M_2, ..., M_m$. Tasks are indexed according to non-increasing suspension ratio: $\frac{s_i}{p_i} \geq \frac{s_{i+1}}{p_{i+1}}$ for all i. Ties are broken by p_i ($p_i \leq p_{i+1}$ for all i).

▷ $\{M_1, ..., M_k\}$ represents the set of used processors that have been assigned at least one task

```
1  for i ← 1 to n
2  ▷ i ranges over the tasks
3     for j ← 1 to k
4     ▷ j ranges over the processors that have been
        assigned at least one task
5        if u(M_j) ≤ 1 and Eq. (1) both hold after
         assigning τ_i to M_j
6           mark M_j as a feasible processor for τ_i
7     endfor
8     if the feasible processor set for τ_i is not empty
9        assign τ_i to a processor M_j in this set that
         results in the smallest δ(τ_i, M_j)
10       continue;
11    if k == m return PARTITIONING FAILED
12    assign τ_i to M_{k+1}
13    k = k + 1
14 endfor
15 return PARTITIONING SUCCEEDED
```

Figure 8: SSPartition pseudocode.

Analysis of SSPartition. The following theorem gives a condition for *SSPartition* to successfully partition any given task system onto m processors. Before stating the theorem, we first prove the following lemma.

Definition 4. Let U_{m-1} denote the sum of $m-1$ largest task utilizations. Let V_m denote the sum of m largest task suspension ratios.

Lemma 1. *Under Algorithm SSPartition, if a task τ_i is the first task that cannot be assigned to any processor, then $\sum_{k=1}^{i} u_k > m - U_{m-1} - V_m$.*

Proof. Due to the fact that some task τ_i cannot be assigned to any processor, the last processor M_m does not have enough capacity to accommodate τ_i. That is, if τ_i is assigned to M_m, then either the total utilization on M_m becomes greater than 1 or Eq. (1) does not hold. Moreover, for each previous processor M_j, where $j \leq m-1$, there exists a task, denoted by τ^j (for the last processor, we know that τ^m is τ_i), that could not be assigned to M_j, and thus another processor was considered to accommodate τ^j. Let $u(\tau^j)$ denote the utilization of τ^j. Hence, we know that for processor M_j, either

$$u(M_j) + u(\tau^j) > 1, \tag{2}$$

or, after assigning τ^j to M_j, on M_j,[4] we have

$$\begin{aligned} & \max_k \left\{ \sum_{i=1}^{k} u_i + \frac{s_k}{p_k} \right\} > 1 \\ \Rightarrow \quad & u(M_j) + u(\tau^j) + max_k \left(\frac{s_k}{p_k} \right) \\ & \geq \max_k \left\{ \sum_{i=1}^{k} u_i + \frac{s_k}{p_k} \right\} \\ & > 1. \end{aligned} \tag{3}$$

Thus, according to Eqs. (2) and (3), for any processor M_j, its allocated capacity before being assigned τ^j must be strictly greater than $1 - u(\tau^j) - max_k \left(\frac{s_k}{p_k} \right)$. Since tasks $\{\tau_1, \tau_2, ..., \tau_{i-1}\}$ have been successfully assigned, the total utilization of these tasks is equal to the total allocated capacity of processors, which is given by $\sum_{k=1}^{i-1} u_k$. Hence, we have

$$\begin{aligned} & \textstyle\sum_{k=1}^{i-1} u_k > \sum_{j=1}^{m} \left(1 - u(\tau^j) - max_k \left(\frac{s_k}{p_k} \right) \right) \\ \Leftrightarrow \quad & \{\text{adding } u_i \text{ on both sides}\} \\ & \textstyle\sum_{k=1}^{i} u_k > \sum_{j=1}^{m} \left(1 - u(\tau^j) - max_k \left(\frac{s_k}{p_k} \right) \right) \\ & \qquad + u_i \\ \Leftrightarrow \quad & \{\text{because } u(\tau^m) = u_i \text{ and by the definition of } V_m\} \\ & \textstyle\sum_{k=1}^{i} u_k > m - \sum_{j=1}^{m-1} u(\tau^j) - V_m \\ \Rightarrow \quad & \{\text{by the definition of } U_{m-1}\} \\ & \textstyle\sum_{k=1}^{i} u_k > m - U_{m-1} - V_m. \end{aligned}$$

□

Theorem 2. *Algorithm SSPartition successfully partitions any synchronous periodic harmonic suspending task system τ on m processors for which*

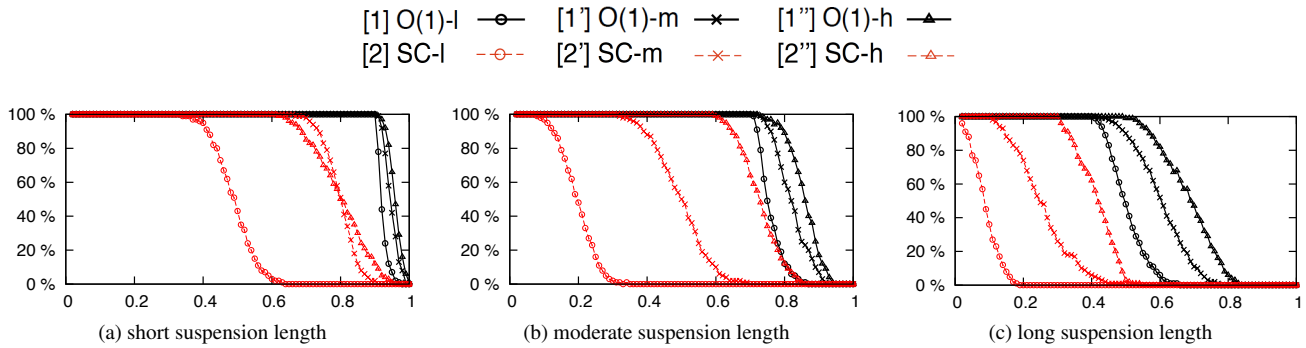
$$U_{sum} \leq m - U_{m-1} - V_m. \tag{4}$$

Proof. Let us suppose that Algorithm *SSPartition* fails to assign the i^{th} task τ_i to any processor for contradiction. Then by Lemma 1, $\sum_{k=1}^{i} u_k > m - U_{m-1} - V_m$ holds. Therefore, we have

$$\begin{aligned} & \textstyle\sum_{k=1}^{i} u_k > m - U_{m-1} - V_m \\ \Rightarrow \quad & U_{sum} \geq \textstyle\sum_{k=1}^{i} u_k > m - U_{m-1} - V_m. \end{aligned}$$

Hence, any system that Algorithm *SSPartition* fails to partition must have $U_{sum} > m - U_{m-1} - V_m$, in which we reach the contradiction. □

[4] Recall that tasks that have been assigned to M_j are indexed again by periods.



(a) short suspension length (b) moderate suspension length (c) long suspension length

Figure 9: Uniprocessor schedulability results. In all three graphs, the x-axis denotes the task set utilization cap and the y-axis denotes the fraction of generated task sets that were schedulable. In inset (a) (respectively, (b) and (c)), short (respectively, moderate and long) per-task suspension lengths are assumed. Each graph gives six curves per tested approach for the cases of light, medium, and heavy per-task utilizations, respectively. As seen at the top of the figure, the label "Θ**(1)-l(m/h)**" indicates the approach of $\Theta(1)$ assuming light (medium/heavy) utilizations. Similar "**SC**" labels are used for SC.

6 Experiments

In this section, we describe experiments conducted using randomly-generated task sets to evaluate the applicability of our uniprocessor schedulability test (Theorem 1) and partitioning-based multiprocessor schedulability test (Theorem 2).

In our experiments, harmonic suspending task sets were generated similar to the methodology used in [15, 16]. Harmonic task periods were distributed uniformly over [$2ms$, $1024ms$] (e.g., $\{2ms, 4ms, 8ms, ..., 1024ms\}$). Task utilizations were distributed differently for each experiment using three uniform distributions. The ranges for the uniform distributions were [0.005, 0.1] (light), [0.1, 0.3] (medium), and [0.3, 0.5] (heavy). Task execution costs were calculated from periods and utilizations. Suspensions lengths of tasks were also distributed using three uniform distributions: $[0.005 \cdot (1-u_i) \cdot p_i, 0.1 \cdot (1-u_i) \cdot p_i]$ (suspensions are short), $[0.1 \cdot (1-u_i) \cdot p_i, 0.3 \cdot (1-u_i) \cdot p_i]$ (suspensions are moderate), $[0.3 \cdot (1-u_i) \cdot p_i, 0.6 \cdot (1-u_i) \cdot p_i]$ (suspensions are long).[5] Suspension lengths generated by these parameters range from $5\mu s$ - $611ms$, which we believe is wide enough to cover a large set of workloads that incur suspensions in practice. We varied the total system utilization U_{sum} within $0.1, 0.2, ..., m$. For each combination of task utilization distribution, suspension length distribution, and U_{sum}, 10,000 task sets were generated for system with $m = 1$, $m = 4$ and $m = 8$ processors. Each such task set was generated by creating tasks until total utilization exceeded the corresponding utilization cap, and by then reducing the last task's utilization so that the total utilization equaled the utilization cap.

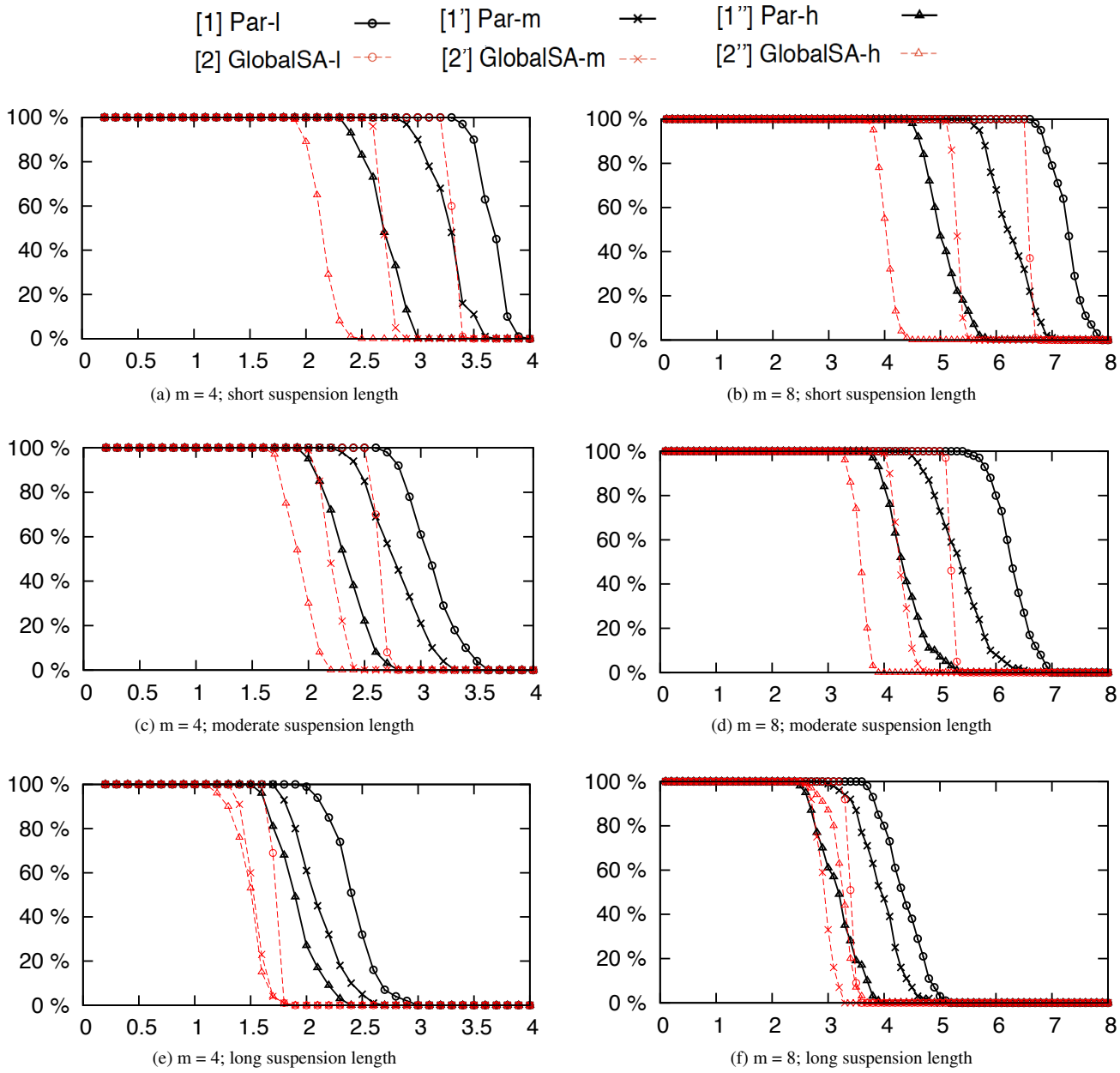
Uniprocessor schedulability. We evaluated the effectiveness of the proposed uniprocessor schedulability test by comparing Theorem 1, denoted by "$\Theta(1)$", to the suspension-oblivious approach denoted by "SC".[6] That is, after transforming all suspending tasks into ordinary periodic tasks (no suspensions) using SC, the original task system is schedulable if the total utilization of the transformed task system is not greater than 1.

The obtained schedulability results are shown in Fig. 9 (the organization of which is explained in the figure's caption). Each curve plots the fraction of the generated task sets the corresponding approach successfully scheduled, as a function of total utilization. As seen, in all tested scenarios, $\Theta(1)$ improves upon SC by a substantial margin. For example, as seen in Fig. 9(a), when suspensions are short, $\Theta(1)$ can achieve 100% schedulability under all three task suspension length distributions when $U_{sum} \leq 0.9$; while SC fails to do so when U_{sum} merely exceeds 0.4, 0.6, and 0.7 with light, medium, and heavy task utilizations, respectively. Moreover, as seen in all insets of Fig 9, when task suspension lengths and task utilizations increase, the utilization loss suffered by SC becomes much more significant. For example, when suspension lengths are long and task utilizations are heavy, SC fails for task sets even with $U_{sum} < 0.2$, while $\Theta(1)$ can still guarantee 100% schedulability when $U_{sum} \leq 0.4$. The negative impact due to long task suspension lengths on $\Theta(1)$ is limited because only one suspending task can cause additional suspension-related utilization loss.

Multiprocessor schedulability. We also evaluated the effectiveness of the proposed partitioning algorithm by comparing Theorem 2, denoted by "Par", to an existing schedulability test [16] for suspending tasks, denoted "GlobalSA", which is the only available global suspension-aware analy-

[5] Note that any s_i is upper-bounded by $(1-u_i) \cdot p_i$

[6] We choose to compare against the suspension-oblivious approach because it can be used to support the general suspension model considered in this paper; whereas most existing uniprocessor suspension-aware analysis techniques assume a restricted suspension model (e.g., tasks that can suspend at most once [8, 24]).



(a) m = 4; short suspension length

(b) m = 8; short suspension length

(c) m = 4; moderate suspension length

(d) m = 8; moderate suspension length

(e) m = 4; long suspension length

(f) m = 8; long suspension length

Figure 10: Multiprocessor schedulability results. In all six graphs, the x-axis denotes the task set utilization cap and the y-axis denotes the fraction of generated task sets that were schedulable. In the first (respectively, second) column of graphs, $m = 4$ (respectively, $m = 8$) is assumed. In the first (respectively, second and third) row of graphs, short (respectively, moderate and long) per-task suspension length are assumed. Each graph gives three curves per tested approach for the cases of light, medium, and heavy per-task utilizations, respectively. As seen at the top of the figure, the label "**Par-l(m/h)**" indicates the approach of Par assuming light (medium/heavy) per-task utilizations. Similar "**GlobalSA**" labels are used for GlobalSA.

sis.[7] Also, it has been shown in [16] that this suspension-aware approach achieves better performance w.r.t. HRT schedulability compared to other existing approaches.

[7]Note that the analysis presented in [16] is able to handle more general periodic suspending task systems.

The obtained multiprocessor schedulability results are shown in Fig. 10 (the organization of which is explained in the figure's caption). As seen, in most scenarios, Par improves upon GlobalSA. For example, as seen in Fig. 10(a), when task utilizations are heavy and suspension lengths are short, Par is able to achieve 100% schedulability when

$U_{sum} \leq 2.3$; while GlobalSA fails to do so when U_{sum} exceeds 1.9. This suggests the effectiveness of Par even if it suffers from the bin-packing-related utilization loss. Another observation is that when suspension lengths are long and utilizations are heavy, Par yields weaker performance, which becomes comparable to GlobalSA in one case (as shown in Fig. 10(f)). This is because the utilization constraint (as seen in Eq. (4) becomes more severe in this case, as U_{m-1} increases when task utilizations become heavier, and V_m also increases when suspension lengths become longer.

7 Conclusion and Future Work

In this paper, we have presented new techniques for analyzing HRT synchronous periodic suspending tasks with harmonic periods. The resulting uniprocessor schedulability test yields only an $\Theta(1)$ suspension-related utilization loss, which theoretically dominates the suspension-oblivious approach [17]. Then based upon this uniprocessor schedulability test, we further present a partitioning scheme for handling such task systems on multiprocessors. The resulting multiprocessor schedulability test yields only an $\Theta(m)$ utilization loss. As demonstrated by experiments presented herein, our proposed tests significantly improves upon prior methods with respect to schedulability, and are often able to guarantee schedulability with little or no utilization loss.

In future work, we plan to design new analysis techniques that can efficiently handle general sporadic suspending task systems. Moreover, new scheduling algorithms that may better deal with suspensions need to be designed and analyzed, as classical schedulers such as EDF and RM might not be the best choices in this case.

Acknowledgment: This research was supported in part by a start-up grant from the University of Texas at Dallas, DFG, as part of the Collaborative Research Center SFB876, the priority program "Dependable Embedded Systems", iTrust IGDSi1301013.

References

[1] Avionics application software standard interface: Part 1 - required services (arinc specication 653-2). Technical report, Avionics Electronic Engineering Committee (ARINC), 2006.

[2] Green hills software, arinc 653 partition scheduler. www.ghs.com/products/safety critical/arinc653.html, 2013.

[3] Windriver, platform for arinc 653. www.windriver.com/ products/platforms/safetycritical/, 2013.

[4] Riccardo Bettati and Jane W-S Liu. End-to-end scheduling to meet deadlines in distributed systems. In *Distributed Computing Systems, 1992., Proceedings of the 12th International Conference on*, pages 452–459. IEEE, 1992.

[5] B. Brandenburg. *Scheduling and Locking in Multiprocessor Real-Time Operating Systems*. PhD thesis, University of North Carolina at Chapel Hill, Chapel Hill, NC, 2011.

[6] A. Easwaran, L. Insup, O. Sokolsky, and S. Vestal. A compositional scheduling framework for digital avionics systems. In *Proceedings of the 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 371–380, 2009.

[7] J. Goossens, S. Funk, and S. Baruah. Priority-driven scheduling of periodic task systems on multiprocessors. *Real-Time Systems*, 25(2-3):187–205, 2003.

[8] I. Kim, K. Choi, S. Park, D. Kim, and M. Hong. Real-time scheduling of tasks that contain the external blocking intervals. In *Proceedings of the 2nd International Workshop on Real-Time Computing Systems and Applications*, pages 54–59, 1995.

[9] T. Kuo and A. Mok. Load adjustment in adaptive real-time systems. In *Proceedings of the 12th Real-Time Systems Symposium*, pages 160–170, 1991.

[10] K. Lakshmanan and R. Rajkumar. Scheduling self-suspend- ing real-time tasks with rate-monotonic priorities. In *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 3–12, 2010.

[11] J. Lee, S. Xi, S. Chen, L. Phan, C. Gill, I. Lee, C. Lu, and O. Sokolsky. Realizing compositional scheduling through virtualization. In *Proceedings of the 18th IEEE Real Time and Embedded Technology and Applications Symposium*, pages 13–22, 2012.

[12] C. Liu and J. Anderson. Task scheduling with self-suspensions in soft real-time multiprocessor systems. In *Proceedings of the 30th Real-Time Systems Symposium*, pages 425–436, 2009.

[13] C. Liu and J. Anderson. Improving the schedulability of sporadic self-suspending soft real-time multiprocessor task systems. In *Proceedings of the 16th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 14–23, 2010.

[14] C. Liu and J. Anderson. Scheduling suspendable, pipelined tasks with non-preemptive sections in soft real-time multiprocessor systems. In *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 23–32, 2010.

[15] C. Liu and J. Anderson. An O(m) analysis technique for supporting real-time self-suspending task systems. In *Proceedings of the 33th IEEE Real-Time Systems Symposium*, pages 373–382, 2012.

[16] C. Liu and J. Anderson. Suspension-aware analysis for hard real-time multiprocessor scheduling. In *Proceedings of the 25th EuroMicro Conference on Real-Time Systems*, pages 271–281, 2013.

[17] J. Liu. *Real-Time Systems*. Prentice Hall, 2000.

[18] M. Mollison, J. Erickson, J. Anderson, S. Baruah, and J. Scoredos. Mixed criticality real-time scheduling for multicore systems. In *Proceedings of the 7th IEEE International Conference on Embedded Software and Systems*, pages 1864–1871, 2010.

[19] J. C. Palencia and M. Gonzlez Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *Proceedings of the 19th IEEE Real-Time Systems Symposium*, pages 26–37, 1998.

[20] J. C. Palencia and M. Gonzlez Harbour. Response time analysis of EDF distributed real-time systems. *Journal of Embedded Computing*, 1(2):225–237, 2005.

[21] R. Rajkumar. Dealing with Suspending Periodic Tasks. Technical report, IBM T. J. Watson Research Center, 1991.

[22] F. Ridouard, P. Richard, and F. Cottet. Negative results for scheduling independent hard real-time tasks with self-suspensions. In *Proceedings of the 25th IEEE Real-Time Systems Symposium*, pages 47–56, 2004.

[23] L. Sha and J. Goodenough. Real-time scheduling theory and Ada. *IEEE Computer*, 23(4):53–62, 1990.

[24] K. Tindell. Adding time-offsets to schedulability analysis. Technical Report 221, University of York, 1994.