Suspension-Aware Analysis for Hard Real-Time Multiprocessor Scheduling

Cong Liu and James H. Anderson
Department of Computer Science, University of North Carolina at Chapel Hill

Abstract

In many real-time systems, tasks may experience suspension delays when accessing external devices. The problem of analyzing task systems with such suspensions on multiprocessors has been relatively unexplored. The commonly used suspension-oblivious approach of treating all suspensions as computation can be quite pessimistic. As an alternative, this paper presents the first suspension-aware hard real-time multiprocessor schedulability analysis for task systems with suspensions, under both global fixed-priority and global EDF scheduling. In experiments presented herein, the proposed schedulability tests proved to be superior to suspension-oblivious tests. Moreover, when applied to ordinary arbitrary-deadline sporadic task systems with no suspensions, the proposed analysis for fixed-priority scheduling improves upon prior analysis.

I. Introduction

In many embedded systems, suspension delays may occur when tasks block to access shared resources or perform operations on external devices. Such delays can be quite lengthy, in which case schedulability in embedded systems is negatively impacted. For example, delays introduced by disk I/O range from $15\mu s$ (for NAND flash) to $15ms$ (for magnetic disks) per read [9]. Delays introduced by accessing devices such as GPUs could range from a few milliseconds to several seconds [7], [15].

It has been shown that precisely analyzing hard real-time (HRT) systems with suspensions is difficult, even for very restricted self-suspending task models on uniprocessors [17]. However, uniprocessor analysis that is correct in a sufficiency sense (although pessimistic) has been proposed (see [16] for an overview). Such analysis can be applied on a per-processor basis to deal with suspensions under partitioning approaches where tasks are statically bound to processors. In contrast, for global scheduling where tasks are scheduled from a single run queue and may migrate across processors, other than the *suspension-oblivious* approach, which simply integrates suspensions into per-task worst-case-execution-time requirements, no known global HRT schedulability analysis exists for self-suspending task systems. Such approaches are the main focus of this paper.

Work supported by NSF grants CNS 1239135, CNS 0834270, CNS 0834132, and CNS 1016954; ARO grant W911NF-09-1-0535; AFOSR grant FA9550-09-1-0549; and AFRL grant FA8750-11-1-0033.

Unfortunately, unless the number of self-suspending tasks is small and suspension delays are short, the suspension-oblivious approach may cause significant capacity loss. A potentially better alternative is to explicitly consider suspensions in the task model and corresponding schedulability analysis; this is known as *suspension-aware* analysis. In this paper, we present the first suspension-aware analysis for globally scheduled HRT self-suspending tasks for multiprocessor embedded systems. We focus specifically on two widely used global schedulers: global fixed-priority (GFP) and global EDF (GEDF) scheduling. We analyze these schedulers assuming the scheduled workload is an arbitrary-deadline sporadic self-suspending (SSS) task system (formerly defined in Sec. 2). To enable more general results, we allow soft real-time (SRT) tasks to be supported in addition to HRT ones. Under our definition of SRT, a task can miss a deadline, provided the extent of violation is constrained by a user-specified predefined *deadline tardiness threshold*. A summary of prior related work and our specific contributions is given next.

Overview of related work. Analysis approaches have been proposed in [13], [14] that can check whether bounded response times can be ensured for any given SSS task system. However, these approaches cannot be used to determine whether hard deadlines or predefined deadline tardiness thresholds (as defined in Sec. II) can be met. An overview of work on scheduling SSS task systems on uniprocessors (which we omit here due to space constraints) can be found in [11], [12], [14]. While (as noted earlier) such work can be applied under partitioning approaches, such approaches suffer from bin-packing-related capacity loss, which limits their usefulness, as we shall see in experiments presented later.

Our contributions. The common suspension-oblivious approach of treating all suspensions as computation for analyzing SSS task systems on multiprocessors (or uniprocessors) is pessimistic. In order to support SSS task systems in a more efficient way, we present in this paper the first suspension-aware global HRT multiprocessor schedulability analysis for general SSS task models. Our specific contributions are as follows. First, we present HRT multiprocessor schedulability tests for arbitrary-deadline SSS task systems under both GFP and GEDF scheduling (note that these tests can also be applied on uniprocessors). Second, we show that our analysis is general enough to also support SRT (with predefined tardiness thresholds) SSS tasks. Third, we demonstrate that, for ordinary arbitrary-deadline task systems with no suspensions, our fixed-priority test has lower time complexity than the best prior test [8]. This is because our test is based on an interval analysis framework tailored to the arbitrary-deadline case, while the test in [8] is based on a framework that encompasses both constrained- and arbitrary-deadline task systems and thus is less exact. Fourth, we present experimental results that show that our suspension-aware analysis is much superior to the prior suspension-oblivious approach. Also, when applied to ordinary arbitrary-deadline task systems with no suspensions, our fixed-priority analysis has better runtime performance than that proposed in [8].

The rest of this paper is organized as follows. In Sec. 2, we present the SSS task model. Then, in Secs. 3 and 4, we present the aforementioned suspension-aware GFP and GEDF schedulability tests, respectively. In Sec. 5, we experimentally compare them with other methods. In these experiments, our tests exhibited superior performance, typically by a wide margin. We conclude in Sec. 6.

II. System Model

We consider the problem of scheduling an SSS task system $\tau = \{\tau_1, ..., \tau_n\}$ of n independent SSS tasks on $m \geq 1$ identical processors. Each task is released repeatedly, with each such invocation called a *job*. Jobs alternate between computation and suspension phases. We assume that each job of any task τ_i executes for at most e_i time units (across all of its computation 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). Note that if $s_i = 0$, then τ_i is an ordinary sporadic task. Associated with each task τ_i are a *period* p_i, which specifies the minimum time between two consecutive job releases of τ_i[1], and a *relative deadline* d_i. For any task τ_i, we require $e_i + s_i \leq min(d_i, p_i)$. The k^{th} job of τ_i, denoted $\tau_{i,k}$, is released at time $r_{i,k}$ and has a deadline at time $d_{i,k} = r_{i,k} + d_i$. 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 $u_{sum} = \sum_{\tau_i \in \tau} u_i$. An SSS task system τ is said to be an *arbitrary-deadline* system if, for each τ_i, the relation between d_i and p_i is not constrained (e.g., $d_i > p_i$ is possible).[2] In this paper, we consider arbitrary-deadline SSS task systems.

[1]In a *periodic* task system, each task τ_i's period specifies the exact time between two consecutive job releases of τ_i.

[2]τ is said to be a *constrained* system if, for each task $\tau_i \in \tau$, $d_i \leq p_i$.

Successive jobs of the same task are required to execute in sequence. If a job $\tau_{i,k}$ completes at time t (that is, its last phase, be it a computation or suspension phase, completes at t), then its *response time* is $t - r_{i,k}$ and its *tardiness* is $max(0, t - d_{i,k})$. A task's response time (tardiness) is the maximum of the response time (tardiness) of any of its jobs. Note that, when a job of a task misses its deadline, the release time of the next job of that task is not altered.

In this paper, we establish HRT schedulability tests for arbitrary-deadline SSS task systems under both GFP (Sec. III) and GEDF (Sec. IV). Under GFP, tasks are assigned fixed priorities; a job has the same priority as the task to which it belongs. Under GEDF, released jobs are prioritized by their deadlines and any ties are broken by task ID (lower IDs are favored).

To enable more general results, for any task τ_i, we allow a predefined tardiness threshold to be set, denoted λ_i. (Task τ_i is an ordinary HRT task if $\lambda_i = 0$.) Throughout the paper, we assume that e_i, s_i, d_i, p_i, and λ_i for any task $\tau_i \in \tau$ are non-negative integers and all time values are integral. Thus, a job that executes at time point t executes during the entire time interval $[t, t+1)$.

For simplicity, we henceforth assume that each job of any task τ_i executes for *exactly* e_i time units. As shown in [14], any response-time bound derived for an SSS task system by considering only schedules meeting this assumption applies to other schedules as well. (This property was shown in [14] for GEDF, but it applies to GFP as well.) For any job $\tau_{i,k}$, we let $s_{i,k}$ denote its total suspension time, where $s_{i,k} \leq s_i$.

Real-time workloads often have both self-suspending tasks and computational tasks (which do not suspend) co-exist. To reflect this, we let τ^s (τ^e) denote the set of self-suspending (computational) tasks in τ. Also, we let n_s (n_e) denote the number of self-suspending (computational) tasks in τ.

III. GFP Schedulability Test

In this section, based upon response-time analysis (RTA), we derive a fixed-priority multiprocessor schedulability test for HRT and SRT (i.e., each task τ_i can have a predefined tardiness threshold λ_i) arbitrary-deadline SSS task systems.

Under GFP, a task cannot be interfered with by tasks with lower priorities. Assume that tasks are ordered by decreasing priority, i.e., $i < k$ iff τ_i has a higher priority than τ_k.

Definition 1. Let $\tau_{l,j}$ be the *maximal* job of τ_l, i.e., $\tau_{l,j}$ either has the largest response time among all jobs of τ_l or it is the first job of τ_l that has a response time exceeding $d_l + \lambda_l$.

We assume $l > m$ since under GFP, any task τ_i where

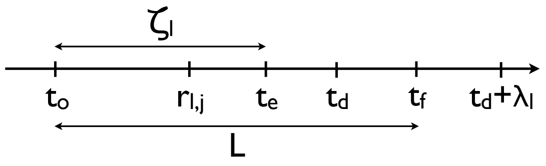


Fig. 1: The maximal job $\tau_{l,j}$ of task τ_l becomes eligible at t_e. t_o is the earliest time instant before t_e such that at any time instant $t \in [t_o, t_e)$ all processors are occupied by tasks with equal or higher priority than $\tau_{l,j}$.

$i \leq m$ has a response time bound of $e_i + s_i$. We further assume that for any task τ_i where $i < l$, its largest response time does not exceed $d_i + \lambda_i$. Our analysis focuses on the job $\tau_{l,j}$, as defined above. To avoid distracting "boundary cases," we also assume that the schedule being analyzed is prepended with a schedule in which no deadlines are missed that is long enough to ensure that all predecessor jobs referenced in the proof exist (this applies to Sec. IV as well).

Definition 2. A job is said to be *completed* if it has finished its last phase (be it suspension or computation). $f_{i,k}$ denotes the completion time of job $\tau_{i,k}$. The *eligible time* of job $\tau_{i,k}$ is defined to be $max(r_{i,k}, f_{i,k-1})$. A task τ_i is *active* at time t if there exists a job $\tau_{i,k}$ such that $r_{i,k} \leq t < f_{i,k}$.

Definition 3. Let t_f denote the completion time of our job of interest $\tau_{l,j}$, t_e denote its eligible time (i.e., $t_e = max(r_{l,j}, f_{l,j-1})$), and t_d denote its deadline.

As in [2], we extend the analyzed interval from t_e to an earlier time instant t_o as defined below.

Definition 4. t_o denotes the earliest time instant at or before t_e such that at any time instant $t \in [t_o, t_e)$ all processors are occupied by tasks with equal[3] or higher priority than τ_l, as illustrated in Fig. 1.

For conciseness, let $\tau_{hp} \subseteq \tau$ denote the set of tasks that have equal or higher priority than the analyzed task τ_l, and let

$$L = t_f - t_o \tag{1}$$

and

$$\zeta_l = t_e - t_o. \tag{2}$$

Definition 5. A task τ_i has a *carry-in* job if there is a job of τ_i that is released before t_o that has not completed by t_o.

Two parameters are important to RTA: the *workload* and the *interference*, as defined below.

Workload. The workload of an SSS task τ_i in the interval $[t_o, t_f)$ is the amount of computation that τ_i requires to execute in $[t_o, t_f)$. Note that suspensions do not contribute to the workload since they do not occupy any processor. Let $\omega(\tau_i, L)$ denote an upper bound of the workload of each task $\tau_i \in \tau_{hp}$ in the interval $[t_o, t_f)$ of length L. Let $\omega^{nc}(\tau_i, L)$ denote the workload bound if τ_i does not have a carry-in job (see Def. 5), and let $\omega^{c}(\tau_i, L)$ denote the workload bound if τ_i has a carry-in job. $\omega^{nc}(\tau_i, L)$ and $\omega^{c}(\tau_i, L)$ can be computed as shown in the following lemmas.

[3]Note that any job $\tau_{l,k}$ of τ_l where $k < j$ may delay $\tau_{l,j}$ from executing and thus can be considered to have higher priority than $\tau_{l,j}$.

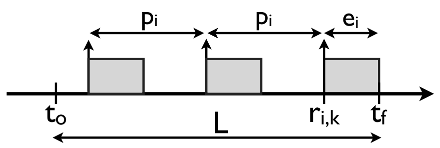


Fig. 2: Computing $\omega^{nc}(\tau_i, L)$.

Lemma 1.

$$\omega^{nc}(\tau_i, L) = \Big(\Big\lfloor \frac{L - e_i}{p_i} \Big\rfloor + 1\Big) \cdot e_i. \tag{3}$$

Proof: Since τ_i does not have a carry-in job, only jobs that are released within $[t_o, t_f)$ can contribute to $\omega^{nc}(\tau_i, L)$. The scenario for the worst-case workload to happen is shown in Fig. 2, where job $\tau_{i,k}$, which is the last job of τ_i that is released before t_f, executes continuously within $[r_{i,k}, r_{i,k} + e_i)$ such that $r_{i,k} + e_i = t_f$ (according to our task model, each suspension of $\tau_{i,k}$ within $[r_{i,k}, t_f)$ may be of length 0), and jobs of τ_i are released periodically. (Note that if $i = l$, then this worst-case scenario still gives a safe upper bound on the workload since in this case $\tau_{l,j}$ could be the job $\tau_{i,k}$.) Besides $\tau_{i,k}$, there are at most $\lfloor \frac{L-e_i}{p_i} \rfloor$ jobs of τ_i released within $[t_o, t_f)$. ■

Definition 6. For any interval of length t, let $\Delta(\tau_i, t) = \big(\lceil \frac{t}{p_i} \rceil - 1\big) \cdot e_i + min\big(e_i, t - \lceil \frac{t}{p_i} \rceil \cdot p_i + p_i\big)$.

$\Delta(\tau_i, t)$ is defined for computing carry-in workloads. Def. 6 improves upon a similar definition for computing carry-in workloads proposed in [10] by deriving a more precise upper bound of the workload.

The following lemma, which computes $\omega^{c}(\tau_i, L)$, is proved similarly to Lemma 1.

Lemma 2.

$$\omega^{c}(\tau_i, L) \quad = \Delta(\tau_i, L - e_i + d_i + \lambda_i) \tag{4}$$

Proof: The scenario for the worst-case workload to happen is shown in Fig. 3, where job $\tau_{i,k}$, which is the last job of τ_i that is released before t_f, executes continuously within $[r_{i,k}, r_{i,k} + e_i)$ such that

$$r_{i,k} + e_i = t_f \tag{5}$$

(recall that $\tau_{i,k}$ may suspend for zero time within $[r_{i,k}, t_f)$), and jobs are released periodically. (Note that if $i = l$, then

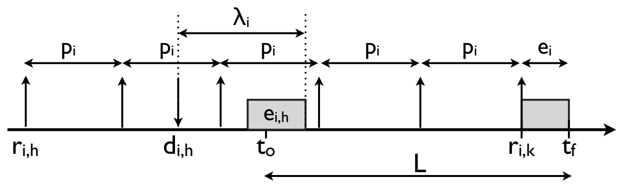


Fig. 3: Computing $\omega^c(\tau_i, L)$.

this worst-case scenario still gives a safe upper bound on the workload since $\tau_{l,j}$ could be the job $\tau_{i,k}$.)

Let $\tau_{i,h}$ be the first job such that $r_{i,h} < t_o$ and

$$d_{i,h} + \lambda_i > t_o, \tag{6}$$

i.e., $\tau_{i,h}$ is the first job of τ_i (potentially tardy) that may execute during $[t_o, t_f)$ and is released before t_o (note that if $\tau_{i,h}$ does not exist, then τ_i would not have a carry-in job). Besides $\tau_{i,k}$ and $\tau_{i,h}$, jobs of τ_i that are released within $[r_{i,h} + p_i, r_{i,k})$ can contribute to $\omega^c(\tau_i, L)$. Let y denote the number of jobs of τ_i released in $[r_{i,h}, r_{i,k})$. There are thus $y-1$ jobs of τ_i that are released within $[r_{i,h}+p_i, r_{i,k})$. Since jobs of τ_i are released periodically, we have

$$r_{i,k} - r_{i,h} = y \cdot p_i. \tag{7}$$

Moreover, work contributed by $\tau_{i,h}$ cannot exceed the smaller of e_i and the length of the interval $[t_o, d_{i,h} + \lambda_i)$. The length of the interval $[t_o, d_{i,h} + \lambda_i)$ is given by $d_{i,h} + \lambda_i - t_o = r_{i,h} + d_i + \lambda_i - t_o \overset{\{\text{by (7)}\}}{=} r_{i,k} - y \cdot p_i + d_i + \lambda_i - t_o \overset{\{\text{by (1) and (5)}\}}{=} (L - e_i + d_i + \lambda_i) - y \cdot p_i$. Thus, the work contributed by $\tau_{i,h}$ is given by $min(e_i, (L - e_i + d_i + \lambda_i) - y \cdot p_i)$.

By summing the contributions of $\tau_{i,h}$, $\tau_{i,k}$, and jobs of τ_i that are released within $[r_{i,h} + p_i, r_{i,k})$, we have

$$\begin{aligned} & \omega^c(\tau_i, L) \\ = \; & min(e_i, (L - e_i + d_i + \lambda_i) - y \cdot p_i) \\ & + e_i + (y-1) \cdot e_i \\ = \; & min(e_i, (L - e_i + d_i + \lambda_i) - y \cdot p_i) \\ & + y \cdot e_i \end{aligned} \tag{8}$$

To find y, by (7), we have $y = \frac{r_{i,k} - r_{i,h}}{p_i} = \frac{r_{i,k} - d_{i,h} + d_i}{p_i} \overset{\{\text{by (6)}\}}{<} \frac{r_{i,k} - t_o + \lambda_i + d_i}{p_i} \overset{\{\text{by (1)and(5)}\}}{=} \frac{L - e_i + \lambda_i + d_i}{p_i}$. For conciseness, let $\sigma = L - e_i + \lambda_i + d_i$. Thus, $y < \frac{\sigma}{p_i}$ holds. If σ mod $p_i = 0$, then $y \le \frac{\sigma}{p_i} - 1 = \lceil \frac{\sigma}{p_i} \rceil - 1$, otherwise, $y \le \lfloor \frac{\sigma}{p_i} \rfloor = \lceil \frac{\sigma}{p_i} \rceil - 1$. Thus, a general expression for y can be given by $y \le \lceil \frac{\sigma}{p_i} \rceil - 1$.

By (8), the maximum value for $\omega^c(\tau_i, L)$ can be obtained when $y = \lceil \frac{\sigma}{p_i} \rceil - 1$. Setting this expression for y into (8), we get $\omega^c(\tau_i, L) = min\big(e_i, \sigma - \lceil \frac{\sigma}{p_i} \rceil \cdot p_i + p_i\big) + \big(\lceil \frac{\sigma}{p_i} \rceil - 1\big) \cdot e_i \overset{\{\text{by Def. 6}\}}{=} \Delta(\tau_i, \sigma) \overset{\{\text{by the def. of } \sigma\}}{=} \Delta(\tau_i, L - e_i + d_i + \lambda_i)$. ■

It is important to point out that neither $\omega^{nc}(\tau_i, L)$ nor $\omega^c(\tau_i, L)$ depends on ζ_l (as defined in (2)). For any given interval $[t_o, t_f)$ of length L, we get the same result of $\omega^{nc}(\tau_i, L)$ and $\omega^c(\tau_i, L)$, regardless of the value of ζ_l. This observation enables us to greatly reduce the time complexity to derive the response time bound, as shown later.

Interference. The interference $I_l(\tau_i, L)$ of a specific task τ_i on τ_l over $[t_o, t_f)$ is the part of the workload of τ_i that has higher priority than $\tau_{l,j}$ and can delay $\tau_{l,j}$ from executing its computation phases. Note that if $i \neq l$, then τ_i cannot interfere with τ_l while τ_i or τ_l is suspending. If $i = l$, then suspensions of job $\tau_{l,k}$ where $k < j$, may delay $\tau_{l,j}$ from executing. However, by Def. 4, all processors are occupied by tasks with equal or higher priority than τ_l at any time instant $t \in [t_o, t_e)$. Thus, whenever suspensions of any such job $\tau_{l,k}$ delay $\tau_{l,j}$ from executing within $[t_o, t_e)$, such suspensions must be overlapped with computation from some other task with higher priority than τ_l. Therefore, it suffices for us to compute the interference using workload as derived in (3) and (4). (Intuitively, this portion of the schedule, i.e., the schedule within $[t_o, t_e)$, would be the same even if τ_l did not suspend, since τ_l has the lowest priority among the tasks being considered.)

As we did for the workload, we also define two expressions for $I_l(\tau_i, L)$. We use $I_l^{nc}(\tau_i, L)$ to denote a bound on the interference of τ_i to τ_l during $[t_o, t_f)$ if τ_i does not have a carry-in job, and use $I_l^c(\tau_i, L)$ if τ_i has a carry-in job.

By the definitions of workload and interference, within $[t_o, t_f)$, if $i \neq l$, then task τ_i cannot interfere with τ_l by more than τ_i's workload in this interval. Thus, we have $I_l^{nc}(\tau_i, L) \le \omega^{nc}(\tau_i, L)$ and $I_l^c(\tau_i, L) \le \omega^c(\tau_i, L)$. The other case is $i = l$. In this case, since $\tau_{l,j}$ cannot interfere with itself, we have $I_l^{nc}(\tau_i, L) \le \omega^{nc}(\tau_l, L) - e_l$ and $I_l^c(\tau_i, L) \le \omega^c(\tau_l, L) - e_l$. Moreover, because τ_i cannot interfere with τ_l while $\tau_{l,j}$ is executing and suspending for a total of $e_l + s_{l,j}$ time units in $[t_o, t_f)$, $I_l(\tau_i, L)$ cannot exceed $L - e_l - s_{l,j}$. Therefore, we have[4]

$$I_l^{nc}(\tau_i, L) = \begin{cases} min(\omega^{nc}(\tau_i, L), L - e_l - s_{l,j} + 1), & \text{if } i \neq l \\ min(\omega^{nc}(\tau_l, L) - e_l, L - e_l - s_{l,j} + 1), & \text{if } i = l \end{cases} \tag{9}$$

and

$$I_l^c(\tau_i, L) = \begin{cases} min(\omega^c(\tau_i, L), L - e_l - s_{l,j} + 1), & \text{if } i \neq l \\ min(\omega^c(\tau_l, L) - e_l, L - e_l - s_{l,j} + 1), & \text{if } i = l. \end{cases} \tag{10}$$

[4]The upper bounds of $I_l^{nc}(\tau_i, L)$ and $I_l^c(\tau_i, L)$ (as shown next) are set to be $L - e_l - s_{l,j} + 1$ instead of $L - e_l - s_{l,j}$ in order to guarantee that the response time bound we get from the schedulability test presented later is valid. A formal explanation of this issue can be found in [5].

Now we define the *total interference bound* on τ_l within any interval $[t_o, t_o + Z)$ of arbitrary length Z, denoted $\Omega_l(Z)$, which is given by $\sum_{\tau_i \in \tau_{hp}} I_l(\tau_i, Z)$. The total interference bound on τ_l within the interval $[t_o, t_f)$ is thus given by $\Omega_l(L)$.

Upper-bounding $\Omega_l(L)$. By Def. 4, either $t_o = 0$, in which case no task has a carry-in job, or some processor is idle in $[t_o - 1, t_o)$, in which at most $m - 1$ computational tasks are active at $t_o - 1$. Thus, at most $min(m-1, n^e_{hp})$ computational tasks in τ_{hp} have carry-in jobs, where n^e_{hp} denotes the number of computational tasks in τ_{hp}. Due to suspensions, however, all self-suspending tasks in τ_{hp} may have carry-in jobs that suspend at t_o. Let τ^s_{hp} denote the set of self-suspending tasks in τ_{hp}. Thus, self-suspending tasks can contribute at most $\sum_{\tau_i \in \tau^s_{hp}} max(I^c_l(\tau_i, L), I^{nc}_l(\tau_i, L))$ work to $\Omega_l(L)$. Let τ^e_{hp} denote the set of computational tasks in τ_{hp} and $\beta^{min(m-1, n^e_{hp})}_{\tau_i \in \tau^e_{hp}}$ denote the $min(m - 1, n^e_{hp})$ greatest values of $max(0, I^c_l(\tau_i, L) - I^{nc}_l(\tau_i, L))$ for any computational task $\tau_i \in \tau^e_{hp}$. Then computational tasks can contribute at most $\sum_{\tau_i \in \tau^e_{hp}} I^{nc}_l(\tau_i, L) + \beta^{min(m-1, n^e_{hp})}_{\tau_i \in \tau^e_{hp}}$ work to $\Omega_l(L)$. Therefore, by summing up the work contributed by both self-suspending tasks and computational tasks, we can bound $\Omega_l(L)$ by

$$\begin{aligned}\Omega_l(L) &= \sum_{\tau_i \in \tau^s_{hp}} max(I^c_l(\tau_i, L), I^{nc}_l(\tau_i, L)) \\ &\quad + \sum_{\tau_i \in \tau^e_{hp}} I^{nc}_l(\tau_i, L) + \beta^{min(m-1, n^e_{hp})}_{\tau_i \in \tau^e_{hp}}. \quad (11)\end{aligned}$$

The time complexity for computing $\sum_{\tau_i \in \tau^s_{hp}} max(I^c_l(\tau_i, L), I^{nc}_l(\tau_i, L))$ and $\sum_{\tau_i \in \tau^e_{hp}} I^{nc}_l(\tau_i, L)$ is $O(n)$. Also, as noted in [2], by using a linear-time selection technique from [6], the time complexity for computing $\beta^{min(m-1, n^e_{hp})}_{\tau_i \in \tau^e_{hp}}$ is $O(n)$. Thus, the time complexity to upper-bound $\Omega_l(L)$ as above is $O(n)$.

Schedulability test. We now derive an upper bound on the response time of task τ_l in an SSS task system τ scheduled using fixed priorities, as stated in Theorem 1. Before stating the theorem, we first present two lemmas, which are used to prove the theorem. Lemma 3 is intuitive since it states that the total interference of tasks with equal or higher priority than τ_l must be large enough to prevent $\tau_{l,j}$ from being finished at $t_o + H$ if $t_o + H < t_f$ holds (recall that t_f is defined to be the completion time of $\tau_{l,j}$).

Lemma 3. *For job $\tau_{l,j}$ and any interval $[t_o, t_o + H)$ of length H, if $H < t_f - t_o$, then*

$$\left\lfloor \frac{\Omega_l(H)}{m} \right\rfloor > H - e_l - s_{l,j}. \quad (12)$$

Proof: $\Omega_l(H)$ denotes the total interference bound on τ_l within the interval $[t_o, t_o + H)$.

If $t_e \geq t_o + H$, then by Def. 4, all processors must be occupied by tasks in τ_{hp} during the interval $[t_o, t_o + H)$, which implies that tasks in τ_{hp} generate a total workload of at least $m \cdot H$ within $[t_o, t_o + H)$ that can interfere with τ_l. Thus, (12) holds since $\Omega_l(H) \geq m \cdot H \geq m \cdot (H - e_l - s_{l,j} + 1)$.

The other possibility is $t_e < t_o + H$. In this case, given (from the statement of the lemma) that $H < t_f - t_o$, job $\tau_{l,j}$ is not yet completed at time $t_o + H$. Thus, only at strictly fewer than $e_l + s_{l,j}$ time points within the interval $[t_o, t_o + H)$ was $\tau_{l,j}$ able to execute its computation and suspension phases (for otherwise it would have completed by $t_o + H$). In order for $\tau_{l,j}$ to execute its computation and suspension phases for strictly fewer than $e_l + s_{l,j}$ time points within $[t_o, t_o + H)$, tasks in τ_{hp} must generate a total workload of at least $m \cdot (H - e_l - s_{l,j} + 1)$ within $[t_o, t_o + H)$ that can interfere with τ_l. Thus, $\Omega_l(H) \geq m \cdot (H - e_l - s_{l,j} + 1)$ holds. ■

Lemma 4. *$t_e - r_{l,j} \leq \kappa_l$, where $\kappa_l = \lambda_l - p_l + d_l$ if $\lambda_l > p_l - d_l$, and $\kappa_l = 0$, otherwise.*

Proof: By Def. 1, we have

$$f_{l,j-1} \leq d_{l,j-1} + \lambda_l. \quad (13)$$

If $\lambda_l > p_l - d_l$, then we have $t_e - r_{l,j} \overset{\{\text{by Def. 3}\}}{=} max(r_{l,j},\ f_{l,j-1}) - r_{l,j} = max(0, f_{l,j-1} - r_{l,j}) \overset{\{\text{by (13)}\}}{\leq} max(0, d_{l,j-1} + \lambda_l - r_{l,j}) = max(0, r_{l,j-1} + d_l + \lambda_l - r_{l,j}) \leq max(0, d_l + \lambda_l - p_l) = \lambda_l - p_l + d_l$.

If $\lambda_l \leq p_l - d_l$, then $f_{l,j-1} \overset{\{\text{by (13)}\}}{\leq} d_{l,j-1} + \lambda_l = r_{l,j-1} + d_l + \lambda_l \leq r_{l,j} - p_l + d_l + \lambda_l \leq r_{l,j}$, which implies that job $\tau_{l,j-1}$ completes by $r_{l,j}$. Thus, by Def. 3, we have $t_e - r_{l,j} = 0$. ■

Theorem 1. *Let ψ_l be the set of minimum solutions of (14) for L below for each value of $s_{l,j} \in \{0, 1, 2, ..., s_l\}$ by performing a fixed-point iteration on the RHS of (14) starting with $L = e_l + s_{l,j}$:*

$$L = \left\lfloor \frac{\Omega_l(L)}{m} \right\rfloor + e_l + s_{l,j}. \quad (14)$$

Then $\psi^{max}_l + \kappa_l$ upper-bounds τ_l's response time, where ψ^{max}_l is the maximum value in ψ_l.

Proof: We first prove by contradiction that $\psi^{max}_l + \kappa_l - \zeta_l$ is an upper bound of τ_l's response time. Assume that the actual worst-case response time of τ_l is given by R, where

$$R > \psi^{max}_l + \kappa_l - \zeta_l. \quad (15)$$

By Defs. 1 and 3, we have

$$R = t_f - r_{l,j}. \tag{16}$$

Thus, we have $\psi_l^{max} \overset{\{\text{by (15)}\}}{<} R + \zeta_l - \kappa_l \overset{\{\text{by (16)}\}}{=} t_f - r_{l,j} + \zeta_l - \kappa_l \overset{\{\text{by (2)}\}}{=} t_f - t_o + t_e - r_{l,j} - \kappa_l \overset{\{\text{by Lemma 4}\}}{\leq} t_f - t_o + \kappa_l - \kappa_l = t_f - t_o$. Hence, by Lemma 3, (12) holds with $H = \psi_l^{max}$, which contradicts the assumption of ψ_l^{max} being a solution of (14). Therefore, $\psi_l^{max} + \kappa_l - \zeta_l$ is an upper bound of τ_l's response time.

By (2) and Def. 4, $\zeta_l \geq 0$ holds. Moreover, by (3) and (4)-(11), $\Omega_l(L)$ is *independent* of ζ_l, which implies that ψ_l^{max} is independent of ζ_l. Thus, the maximum value for the term $\psi_l^{max} + \kappa_l - \zeta_l$, which is given by $\psi_l^{max} + \kappa_l$ when setting $\zeta_l = 0$, is an upper bound of τ_l's response time. ■

Note that (14) depends on $s_{l,j}$. Thus, it is necessary to test each possible value of $s_{l,j} \in \{0, 1, 2, ..., s_l\}$ to find a corresponding minimum solution of (14). By the definition of ψ_l^{max}, ψ_l^{max} can then be safely used to upper-bound τ_l's response time. Moreover, for every task $\tau_i \in \tau$, $\psi_i^{max} \leq d_i + \lambda_i - \kappa_i$ must hold in order for τ to be schedulable; otherwise, some jobs of τ_i may have missed their deadlines by more than the corresponding tardiness thresholds. The following corollary immediately follows.

Corollary 1. *Task system τ is GFP-schedulable upon m processors if, by repeating the iteration stated in Theorem 1 for all tasks $\tau_i \in \tau$, $\psi_i^{max} \leq d_i + \lambda_i - \kappa_i$ holds.*

Comparing with [8]. In [8], an RTA technique, which we refer to as "GY" for short, was proposed to handle ordinary arbitrary-deadline sporadic task systems (without suspensions). (Note that GY is the only prior work that considers multiprocessor RTA techniques for arbitrary-deadline task systems.) In GY, the methodology used for the constrained-deadline case is extended for dealing with the arbitrary-deadline case by recursively solving an RTA equation. This recursive process could iterate many times depending on task parameters, and may not terminate in some rare cases. On the other hand, due to the fact that our analysis used a more suitable interval analysis framework for arbitrary-deadline task systems[5] (which applies to constrained-deadline task systems as well), for any task in an ordinary sporadic task systems (without suspensions), its response-time bound can be found by solving the RTA equation (14) only once and our RTA process always terminates. As shown by experiments presented in Sec. V, our analysis has better runtime performance than GY.

[5]Specifically, we analyzed the eligible time t_e of our job of interest $\tau_{l,j}$, instead of its release time $r_{l,j}$ as done in [8]. In this way, when computing workload and interference, we already considered the case where job $\tau_{l,k}$ where $k < j$ might complete beyond $r_{l,j}$ if $d_l > p_l$ holds. On the contrary, in GY, the analyzed interval is extended to include all prior jobs that may complete beyond $r_{l,j}$ and an RTA equation is recursively solved to find the response-time bound.

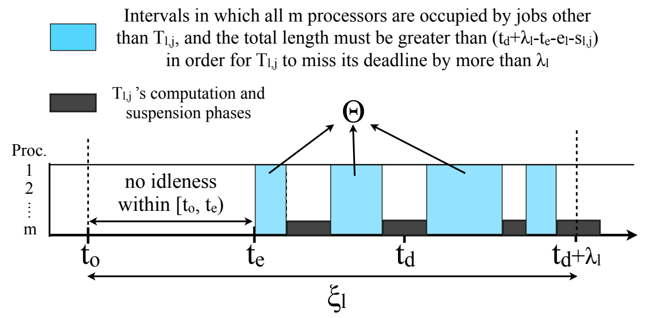

Intervals in which all m processors are occupied by jobs other than $T_{l,j}$, and the total length must be greater than $(t_d+\lambda_l-t_e-e_l-s_{l,j})$ in order for $T_{l,j}$ to miss its deadline by more than λ_l
$T_{l,j}$'s computation and suspension phases
Θ
Proc.
1
2
m
no idleness within $[t_o, t_e)$
t_o
t_e
t_d
$t_d+\lambda_l$
ξ_l

Fig. 4: A job $\tau_{l,j}$ of task τ_l becomes eligible at t_e and misses its deadline at t_d by more than λ_l. t_o is the earliest time instant at or before t_e such that there is no idleness in $[t_o, t_e)$.

IV. GEDF Schedulability Test

In this section, we present a GEDF schedulability test for SSS task systems. Our goal is to identify sufficient conditions for ensuring that each task τ_i cannot miss any deadlines by more than its predefined tardiness threshold, λ_i. These conditions must be checked for each of the n tasks in τ.

Let S be a GEDF 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 $t_d = d_{l,j}$ by more than its predefined tardiness threshold λ_l, as shown in Fig. 4. Under GEDF, jobs with lower priorities than $\tau_{l,j}$ do not affect the scheduling of $\tau_{l,j}$ and jobs with higher priorities than $\tau_{l,j}$, so we will henceforth discard from S all jobs with priorities lower than $\tau_{l,j}$.

Similar to Sec. III, we extend the analyzed interval from $\tau_{l,j}$'s eligible time t_e (see Def. 3) to an earlier time instant t_o as defined below.

Definition 7. t_o denotes the earliest time instant at or before t_e such that there is no idleness in $[t_o, t_e)$.

Our goal now is to identify conditions necessary for $\tau_{l,j}$ to miss its deadline by more than λ_l; i.e., for $\tau_{l,j}$ to execute its computation and suspension phases for strictly fewer than $e_l + s_{l,j}$ time units over $[t_e, t_d + \lambda_l)$. This can happen only if all m processors execute jobs other than $\tau_{l,j}$ for strictly more than $(t_d + \lambda_l - t_e) - (e_l + s_{l,j})$ time units (i.e., at least $t_d + \lambda_l - t_e - e_l - s_{l,j} + 1$ time units) over $[t_e, t_d + \lambda_l)$ (for otherwise, $\tau_{l,j}$ would complete by $t_d + \lambda_l$), as illustrated in Fig. 4. For conciseness, let

$$\xi_l = t_d + \lambda_l - t_o. \tag{17}$$

Definition 8. Let Θ denote a subset of the set of intervals within $[t_e, t_d + \lambda_l)$, where $\tau_{l,j}$ does not execute or suspend, such that the cumulative length of Θ is exactly $t_d + \lambda_l - t_e - e_l - s_{l,j} + 1$ over $[t_e, t_d + \lambda_l)$. As seen in Fig. 4, Θ may not be contiguous.

By Def. 8, the length of the intervals in $[t_o, t_e) \cup \Theta$ is given by $t_e - t_o + t_d + \lambda_l - t_e - e_l - s_{l,j} + 1 = t_d + \lambda_l - t_o - e_l - s_{l,j} + 1 \overset{\{\text{by (17)}\}}{=} \xi_l - e_l - s_{l,j} + 1$.

For each task τ_i, let $W(\tau_i)$ denote the contribution of τ_i to the work done in S during $[t_o, t_e) \cup \Theta$. In order for $\tau_{l,j}$ to miss its deadline, it is necessary that the total amount of work that executes over $[t_o, t_e) \cup \Theta$ satisfies

$$\sum_{\tau_i \in \tau} W(\tau_i) > m \cdot (\xi_l - e_l - s_{l,j}). \qquad (18)$$

This follows from the observation that all m processors are, by Defs. 7 and 8, completely busy executing work over the $\xi_l - e_l - s_{l,j} + 1$ time units in the interval $[t_o, t_e) \cup \Theta$.

Condition (18) is a necessary condition for $\tau_{l,j}$ to miss its deadline by more than λ_l. Thus, in order to show that τ is GEDF-schedulable, it suffices to demonstrate that Condition (18) cannot be satisfied for any task τ_l for any possible values of ξ_l and $s_{l,j}$.

We now construct a schedulability test using Condition (18) as follows. In Sec. IV-A, we first derive an upper bound for the term $\sum_{\tau_i \in \tau} W(\tau_i)$ in the LHS of Condition (18). Then, in Sec. IV-B, we compute possible values of the term $m \cdot (\xi_l - e_l - s_{l,j})$ in the RHS of Condition (18). Later, in Sec. IV-C, a schedulability test is derived based on these results.

A. Upper-Bounding $\sum_{\tau_i \in \tau} W(\tau_i)$

In this section, we derive an upper bound on $\sum_{\tau_i \in \tau} W(\tau_i)$, by first upper-bounding $W(\tau_i)$ for each task τ_i and then summing these per-task upper bounds.

In the following, we compute upper bounds on $W(\tau_i)$. If τ_i has no carry-in job (defined in Def. 5), then let $W_{nc}(\tau_i)$ denote this upper bound; otherwise, let $W_c(\tau_i)$ denote the upper bound. Since $\tau_{l,j}$ is the first job that misses its deadline at t_d by more than its corresponding tardiness threshold, we have

$$f_{l,j-1} \leq d_{l,j-1} + \lambda_l \leq t_d - p_l + \lambda_l. \qquad (19)$$

The following lemma bounds the length of time interval $[t_o, t_e)$.

Lemma 5. $t_e - t_o \leq max(\xi_l - \lambda_l - d_l, \xi_l - p_l)$.

Proof: $t_e - t_o \overset{\{\text{by Def. 3}\}}{=} max(r_{l,j}, f_{l,j-1}) - t_o \overset{\{\text{by (19)}\}}{\leq} max(t_d - d_l, t_d - p_l + \lambda_l) - t_o = max(t_d - d_l - t_o, t_d - p_l + \lambda_l - t_o) \overset{\{\text{by (17)}\}}{=} max(\xi_l - \lambda_l - d_l, \xi_l - p_l)$. ■

If a task τ_i has no carry-in job, then the total amount of work that must execute over $[t_o, t_e) \cup \Theta$ is generated by jobs of τ_i arriving in, and having deadlines within, the interval $[t_o, t_d]$. The following lemma, which was originally proved for ordinary sporadic task systems [4], applies to SSS task systems as well.

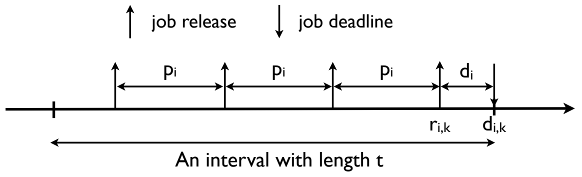


Fig. 5: DBF for self-suspending tasks.

Lemma 6. *The maximum cumulative execution requirement by jobs of an SSS task τ_i that both arrive in, and have deadlines within, any interval of length t is given by demand bound function*

$$DBF(\tau_i, t) = max(0, (\lfloor \tfrac{t-d_i}{p_i} \rfloor + 1) \cdot e_i).$$

Proof: Because we restrict attention to jobs of τ_i that have releases and deadlines within the considered interval of length t, and suspensions do not occupy any processor, the required total work of τ_i can be bounded by considering the scenario in which some job $\tau_{i,k}$ of τ_i has a deadline at the end of the interval and jobs are released periodically. This scenario is illustrated in Fig. 5. There are at most $\lfloor \frac{t-d_i}{p_i} \rfloor$ jobs that are released and have deadlines within the interval other than $\tau_{i,k}$. Thus, the maximum cumulative execution requirement by jobs of τ_i is given by $DBF(\tau_i, t) = max(0, (\lfloor \frac{t-d_i}{p_i} \rfloor + 1) \cdot e_i)$, which is the same as the DBF for ordinary sporadic tasks (i.e., without suspensions). This is due to the fact that suspensions do not contribute to the execution requirement. ■

The lemma below computes $W_{nc}(\tau_i)$ using DBF.

Lemma 7.

$$W_{nc}(\tau_i) = \begin{cases} min\big(DBF(\tau_i, \xi_l - \lambda_l), \\ \quad \xi_l - e_l - s_{l,j} + 1\big) & \textit{if } i \neq l \\ min\big(DBF(\tau_l, \xi_l - \lambda_l) - e_l, \\ \quad max(\xi_l - \lambda_l - d_l, \xi_l - p_l)\big) & \textit{if } i = l \end{cases}$$

Proof: Depending on the relationship between i and l, there are two cases to consider.

Case $i \neq l$**.** The total amount of work contributed by τ_i that must execute over $[t_o, t_e) \cup \Theta$ cannot exceed the total length of the intervals in $[t_o, t_e) \cup \Theta$, which is $\xi_l - e_l - s_{l,j} + 1$. Furthermore, the total work that needs to be bounded must have releases and deadlines within the interval $[t_o, t_d]$, which by (17) is of length $\xi_l - \lambda_l$. By Lemma 6, this total work is at most $DBF(\tau_i, \xi_l - \lambda_l)$.

Case $i = l$**.** As in the previous case, the total work is at most $DBF(\tau_l, \xi_l - \lambda_l)$. However, in this case, since $\tau_{l,j}$ does not execute within $[t_o, t_e) \cup \Theta$, we can subtract its execution requirement, which is e_l, from $DBF(\tau_l, \xi_l -$

$\lambda_l)$. Also, this contribution cannot exceed the length of the interval $[t_o, t_e)$, which by Lemma 5 is at most $max(\xi_l - \lambda_l - d_l, \xi_l - p_l)$. ■

We now consider the case where τ_i has a carry-in job. The following lemma, which computes $W_c(\tau_i)$, is proved similarly to Lemma 7.

Lemma 8.

$$W_c(\tau_i) = \begin{cases} min(\Delta(\tau_i, \xi_l - \lambda_l + \lambda_i), \\ \quad \xi_l - e_l - s_{l,j} + 1) & \textit{if } i \neq l \\ min(\Delta(\tau_l, \xi_l) - e_l, \\ \quad max(\xi_l - \lambda_l - d_l, \xi_l - p_l). & \textit{if } i = l \end{cases}$$

Proof: The total work of τ_i in this case can be upper-bounded by considering the scenario in which some job of τ_i has a deadline at t_d and jobs of τ_i are released periodically, as illustrated in Fig. 6. Depending on the relationship between i and l, we have two cases to consider.

Case $i \neq l$**.** Let $\tau_{i,k}$ be the first job such that $r_{i,k} < t_o$ and

$$d_{i,k} + \lambda_i > t_o, \tag{20}$$

i.e., $\tau_{i,k}$ is the first job of τ_i (potentially tardy) that may execute during $[t_o, t_d)$ and is released before t_o (note that if $\tau_{i,k}$ does not exist then τ_i would have no carry-in job). Since jobs are released periodically,

$$t_d - d_{i,k} = x \cdot p_i \tag{21}$$

holds for some integer x.

The demand for jobs of τ_i in this case is thus bounded by the demand due to x jobs that have deadlines at or before t_d and are released at or after $r_{i,k} + p_i$, plus the demand imposed by the job $\tau_{i,k}$, which cannot exceed the smaller of e_i and the length of the interval $[t_o, d_{i,k} + \lambda_i)$, which by (21) is $t_d - x \cdot p_i + \lambda_i - t_o \overset{\{\text{by (17)}\}}{=} \xi_l - \lambda_l + \lambda_i - x \cdot p_i$. Thus, we have

$$W_c(\tau_i) = x \cdot e_i + min(e_i, \xi_l - \lambda_l + \lambda_i - x \cdot p_i). \tag{22}$$

To find x, by (21), we have $x = \frac{t_d - d_{i,k}}{p_i} \overset{\{\text{by (20)}\}}{<} \frac{t_d - t_o + \lambda_i}{p_i} \overset{\{\text{by (17)}\}}{=} \frac{\xi_l - \lambda_l + \lambda_i}{p_i}$. For conciseness, let $\pi = \xi_l - \lambda_l + \lambda_i$. Thus, $x < \frac{\pi}{p_i}$ holds. If $\pi \bmod p_i = 0$, then $x \leq \frac{\pi}{p_i} - 1 = \lceil \frac{\pi}{p_i} \rceil - 1$, otherwise, $x \leq \lfloor \frac{\pi}{p_i} \rfloor = \lceil \frac{\pi}{p_i} \rceil - 1$. Thus, a general expression for x can be given by $x \leq \lceil \frac{\pi}{p_i} \rceil - 1$.

By (22), the maximum value for $W_c(\tau_i)$ can be obtained when $x = \lceil \frac{\pi}{p_i} \rceil - 1$. Setting this expression for x into (22), we have $W_c(\tau_i) = \big(\lceil \frac{\pi}{p_i} \rceil - 1\big) \cdot e_i + min\big(e_i, \pi - \lceil \frac{\pi}{p_i} \rceil \cdot p_i + p_i\big) \overset{\{\text{by Def. 6}\}}{=} \Delta(\tau_i, \pi) \overset{\{\text{by the def. of } \pi\}}{=} \Delta(\tau_i, \xi_l - \lambda_l + \lambda_i)$.

Moreover, this total demand cannot exceed the total length of the intervals in $[t_o, t_e) \cup \Theta$, which is $\xi_l - e_l - s_{l,j} + 1$.

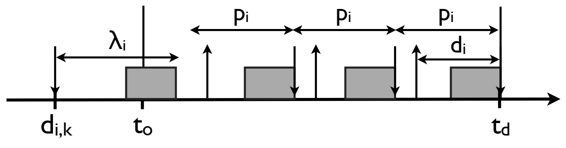


Fig. 6: Computing $W_c(\tau_i)$.

Case $i = l$**.** Repeating the reasoning from the previous case, we find that the total demand of jobs of τ_l with deadlines at most t_d is at most $\Delta(i, \xi_l - \lambda_l + \lambda_i) = \Delta(\tau_i, \xi_l)$. Since $\tau_{l,j}$ does not execute within $[t_o, t_e) \cup \Theta$, we subtract its execution requirement, which is e_l, from $\Delta(\tau_i, \xi_l)$. Also, this contribution cannot exceed the length of the interval $[t_o, t_e)$, which by Lemma 5 is $max(\xi_l - \lambda_l - d_l, \xi_l - p_l)$. ■

Upper-bounding $\sum_{\tau_i \in \tau} W(\tau_i)$**.** Similar to the discussion in Sec. III, by Def. 7, either $t_o = 0$, in which case no task has a carry-in job, or some processor is idle in $[t_o - 1, t_o)$, in which at most $m - 1$ computational tasks are active at $t_o - 1$. Thus, at most $min(m-1, n_e)$ computational tasks can have a carry-in job. However, since suspensions do not occupy any processor, each self-suspending task may be active at $t_o - 1$ and have a job that is suspended at t_o. Thus, in the worst case, all n_s self-suspending tasks can have carry-in jobs. Consequently, there are at most n_s self-suspending tasks and $min(m-1, n_e)$ computational tasks that contribute $W_c(\tau_i)$ work, and the remaining $max(0, n_e - m + 1)$ computational tasks must contribute to $W_{nc}(\tau_i)$. Thus, self-suspending tasks can contribute at most $\sum_{\tau_i \in \tau^s} max\big(W_{nc}(\tau_i), W_c(\tau_i)\big)$ work to $\sum_{\tau_i \in \tau} W(\tau_i)$. Let $\delta_{\tau_i \in \tau^e}^{min(m-1, n_e)}$ denote the $min(m-1, n_e)$ greatest values of $max(0, W_c(\tau_i) - W_{nc}(\tau_i))$ for any computational task τ_i. Then computational tasks can contribute at most $\sum_{\tau_j \in \tau^e} W_{nc}(\tau_j) + \delta_{\tau_k \in \tau^e}^{min(m-1, n_e)}$ work to $\sum_{\tau_i \in \tau} W(\tau_i)$. Therefore, by summing up the work contributed by both self-suspending tasks and computational tasks, we can bound $\sum_{\tau_i \in \tau} W(\tau_i)$ by $\sum_{\tau_i \in \tau^s} max(W_{nc}(\tau_i), W_c(\tau_i)) + \sum_{\tau_j \in \tau^e} W_{nc}(\tau_j) + \delta_{\tau_k \in \tau^e}^{min(m-1, n_e)}$.

Similar to the discussion in Sec. III, the time complexity for computing $W_c(\tau_i)$, $W_{nc}(\tau_i)$, and $W_c(\tau_i) - W_{nc}(\tau_i)$ is $O(n)$. Also, by using a linear-time selection technique from [6], the time complexity for computing $\delta_{\tau_k \in \tau^e}^{min(m-1, n_e)}$ is $O(n)$. Thus, the time complexity to upper-bound $\sum_{\tau_i \in \tau} W(\tau_i)$ as above is $O(n)$.

B. Finding Values of ξ_l and $s_{l,j}$

So far we have upper-bounded the LHS of Condition (18). Recall that our goal is to test Condition (18) for a violation for all possible values of ξ_l and $s_{l,j}$. The following theorem shows that the range of possible values of ξ_l that need to be tested can be limited. Let e_{sum} be the sum of the execution costs for all tasks in τ. For conciseness, let $\phi =$

$m \cdot (e_l + s_{l,j}) - \lambda_l \cdot u_{sum} + \sum_{\tau_i \in \tau} \lambda_i \cdot u_i + e_{sum}$.

Theorem 2. *If Condition (18) is satisfied for τ_l, then it is satisfied for some ξ_l satisfying*

$$min(d_l + \lambda_l, p_l) \leq \xi_l < \frac{\phi}{m - u_{sum}}, \qquad (23)$$

provided $u_{sum} < m$

Proof: By Lemmas 6 and 7, $W_{nc}(\tau_i) \leq \lfloor \frac{\xi_l - \lambda_l}{p_i} \rfloor \cdot e_i + e_i$ holds. By Lemma 8 and Def. 6, $W_c(\tau_i) \leq \lfloor \frac{\xi_l - \lambda_l + \lambda_i}{p_i} \rfloor \cdot e_i + e_i$ holds. Thus, the LHS of Condition (18) is no greater than $\sum_{\tau_i \in \tau} \left(\lfloor \frac{\xi_l - \lambda_l + \lambda_i}{p_i} \rfloor \cdot e_i + e_i \right)$. Assuming Condition (18) is satisfied, we have

$$\begin{aligned}
& \sum_{\tau_i \in \tau} W(\tau_i) > m \cdot (\xi_l - e_l - s_{l,j}) \\
\Rightarrow \quad & \{\text{upper-bounding } \textstyle\sum_{\tau_i \in \tau} W(\tau_i) \text{ as above}\} \\
& \sum_{\tau_i \in \tau} \left(\lfloor \tfrac{\xi_l - \lambda_l + \lambda_i}{p_i} \rfloor \cdot e_i + e_i \right) > m \cdot (\xi_l - e_l - s_{l,j}) \\
\Rightarrow \quad & \{\text{removing the floor}\} \\
& \sum_{\tau_i \in \tau} \left((\xi_l - \lambda_l + \lambda_i) \cdot u_i + e_i \right) > m \cdot (\xi_l - e_l - s_{l,j}) \\
\Rightarrow \quad & \{\text{rearranging}\} \\
& \xi_l \cdot u_{sum} - \lambda_l \cdot u_{sum} + \sum_{\tau_i \in \tau} \lambda_i \cdot u_i + e_{sum} \\
& \quad > m \cdot \xi_l - m \cdot e_l - m \cdot s_{l,j} \\
\Rightarrow \quad & \xi_l < \frac{\phi}{m - u_{sum}},
\end{aligned}$$

provided $u_{sum} < m$.

Moreover, we have $\xi_l \overset{\{\text{by (17)}\}}{=} t_d - t_o + \lambda_l \overset{\{\text{by Def. 7}\}}{\geq} t_d - t_e + \lambda_l \overset{\{\text{by Def. 3}\}}{=} t_d - max(r_{l,j}, f_{l,j-1}) + \lambda_l \overset{\{\text{by (19)}\}}{\geq} t_d - max(t_d - d_l, t_d - p_l + \lambda_l) + \lambda_l = min(d_l, p_l - \lambda_l) + \lambda_l = min(d_l + \lambda_l, p_l)$. ■

Possible values for $s_{l,j}$. By Lemmas 7 and 8, $\sum_{\tau_i \in \tau} W(\tau_i)$, which is the LHS of Condition (18), *depends on* the value of $s_{l,j}$ non-monotonically. Moreover, by Theorem 2, ξ_l also depends on the value of $s_{l,j}$ (recall ϕ). Thus, it is necessary to test all possible values of $s_{l,j}$, which are $\{0, 1, 2, ..., s_l\}$.

C. Schedulability Test

Theorem 3. *Task system τ is GEDF-schedulable on m processors if for all tasks τ_l and all values of ξ_l satisfying (23),*

$$\sum_{\tau_i \in \tau} max\Big(W_{nc}((\tau_i), W_c(\tau_i)) + \sum_{\tau_j \in \tau^e} W_{nc}(\tau_j) + \delta_{\tau_k \in \tau^e}^{min(m-1, n_e)}\Big) \leq m \cdot (\xi_l - e_l - s_{l,j}) \qquad (24)$$

holds for every value of $s_{l,j} \in \{0, 1, 2, ..., s_l\}$.

By Theorem 2, we can test Condition (24) in time pseudo-polynomial in the task parameters.

V. Experiments

In this section, we describe experiments conducted using randomly-generated task sets to evaluate the performance of the proposed schedulability tests. In these experiments, several aspects of our analysis were investigated. In the following, we denote our GEDF and GFP schedulability tests as "Our-EDF" and "Our-FP," respectively.

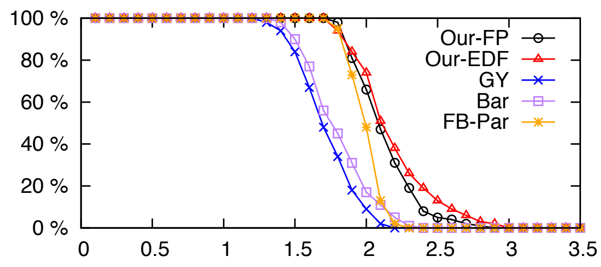
HRT effectiveness. We evaluated the effectiveness of the proposed techniques for HRT SSS task systems by comparing Our-EDF and Our-FP to the suspension-oblivious approach denoted "SC" combined with the tests in [2] and [8], which we denote "Bar" and (as noted earlier) "GY," respectively. That is, after transforming all SSS tasks into ordinary sporadic tasks (no suspensions) using SC, we applied Bar and GY, which are the best known schedulability tests for GEDF and GFP, respectively. In [2], Bar was shown to overcome a major deficiency (i.e., the $O(n)$ carry-in work) of prior GEDF analysis. In [8], GY was shown to be superior to all prior analysis for ordinary task systems available at that time. Moreover, since partitioning approaches have been shown to be generally superior to global approaches on multiprocessors [1], we compared our test to SC combined with the partitioning approach proposed in [3], which we denoted "FB-Par". FB-Par is considered to be the best partitioning approach for constrained-deadline sporadic task systems.

SRT effectiveness. We evaluated the effectiveness of the proposed techniques for SRT SSS task systems with predefined tardiness threshold by comparing them to SC combined with the test proposed in [10], which we denote "LA." LA is the only prior schedulability test for SRT ordinary task systems with predefined tardiness thresholds.

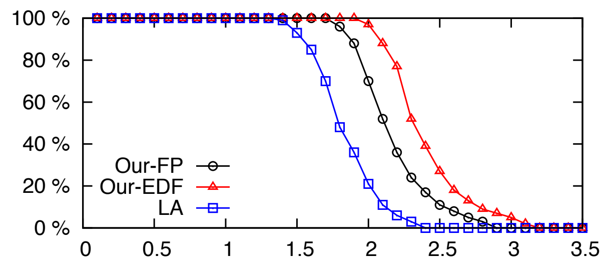
Impact of carry-in work. To evaluate the impact brought by $O(n)$ carry-in work on our analysis, we compared the HRT schedulability for SSS task systems using our analysis to that obtained by applying the analysis proposed in [5], which we denoted "BC," to an otherwise equivalent task system with no suspensions. In [5], BC was shown to be superior to all prior analysis assuming $O(n)$ carry-in work available at that time.

Runtime performance. Finally, we evaluated the effectiveness and the runtime performance of Our-FP for ordinary arbitrary-deadline sporadic task systems (with no suspensions) by comparing it to GY.

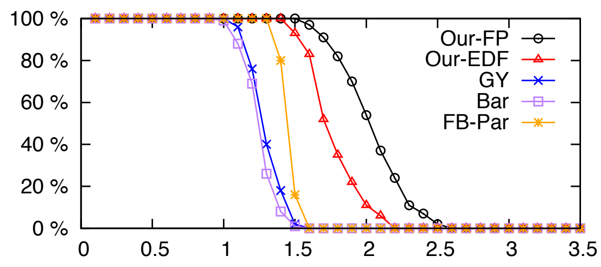
In our experiments, SSS task sets were generated based upon the methodology proposed by Baker in [1]. Integral task periods were distributed uniformly over [10ms,100ms]. Per-task utilizations were uniformly distributed in [0.01, 0.3]. Task execution costs were calculated from periods and utilizations. For any task τ_i in any generated task set, d_i/p_i was varied within [1,2] for the arbitrary-deadline case and within $[max(0.7, \frac{e_i+s_i}{p_i}), 1]$ for the constrained-deadline case, and the tardiness threshold λ_i was varied uniformly within $[0, 2 \cdot p_i]$ for SRT tasks. The suspension



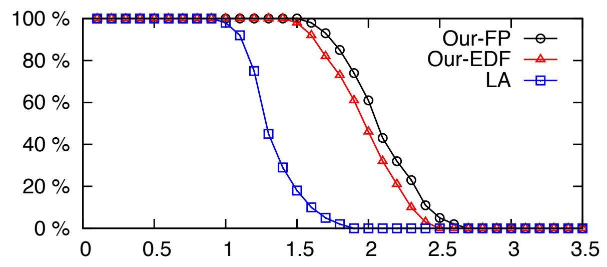
(a) HRT: short suspensions



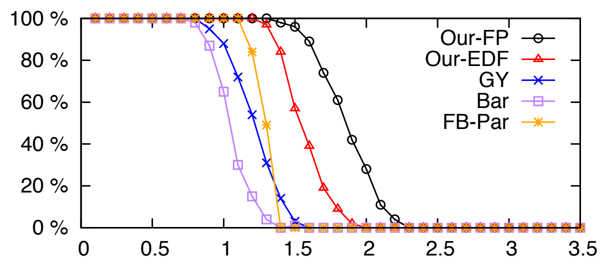
(b) SRT: short suspensions



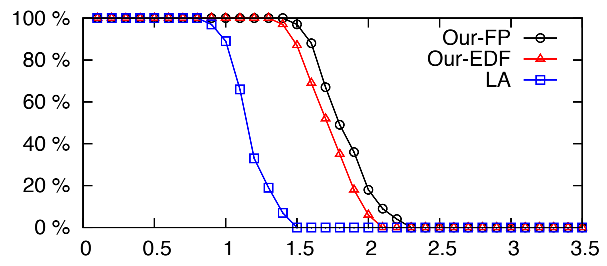
(c) HRT: moderate suspensions



(d) SRT: moderate suspensions



(e) HRT: long suspensions



(f) SRT: long suspensions

Fig. 7: HRT and SRT results. $u_i \in [0.01, 0.3]$, $d_i \in [max(0.7 \cdot p_i, e_i + s_i), p_i]$.

length for any task τ_i was generated by varying s_i/e_i as follows: 0.5 (short suspension length), 1 (moderate suspension length), and 1.5 (long suspension length). Task sets were generated for $m = 4$ processors, as follows. A cap on overall utilization was systematically varied within $[1, 1.1, 1.2, ..., 3.9, 4]$. For each combination of utilization cap and suspension length, we generated 1,000 SSS task sets. Each such SSS task set was generated by creating SSS tasks until total utilization exceeded the corresponding utilization cap, and by then reducing the last task's utilization so that the total utilization equalled the utilization cap. For GFP scheduling, priorities were assigned on a global deadline-monotonic basis. In all figures presented in this section, the x-axis denotes the utilization cap and the y-axis denotes the fraction of generated task sets that were schedulable.

Fig. 7 shows HRT and SRT schedulability results for constrained-deadline SSS task sets achieved by using Our-EDF, Our-FP, Bar, GY, and FB-Par. As seen, for both the HRT and the SRT cases, Our-EDF and Our-FP improve upon the other tested alternatives. Notably, Our-EDF and Our-FP consistently yield better schedulability results than the partitioning approach FB-Par. This is due to the fact that, after treating suspensions as computation, FB-Par suffers from bin-packing-related utilization loss. Moreover, as the suspension length increases, such performance improvement also increases. This is because treating suspension as computation becomes more pessimistic as the suspension length increases. This result suggests that a task's suspensions do not negatively impact the schedulability of other tasks as much as computation does.

Fig. 8 shows HRT schedulability results for constrained-deadline SSS task sets achieved by using Our-FP and by applying BC to otherwise equivalent task sets with no suspensions. In Fig. 8, "Our-FP-S" (respectively, "Our-FP-M") represents schedulability results achieved by Our-FP for the task sets (which are originally generated for BC with no suspensions) after adding suspensions by setting

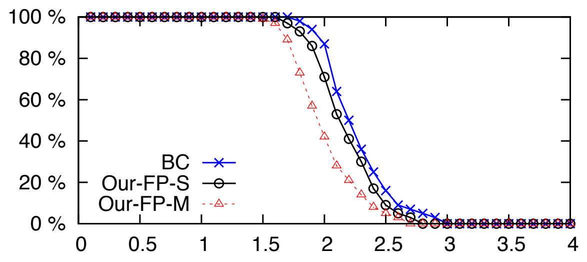


Fig. 8: HRT results compared with BC.

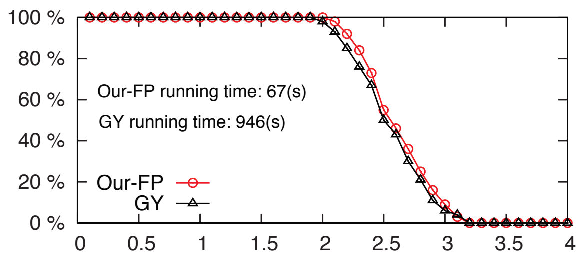


Fig. 9: HRT results compared with GY.

$\frac{s_i}{e_i} = 0.2$ (respectively, $\frac{s_i}{e_i} = 0.5$). It can be seen that Our-FP yields schedulability results that are very close to that achieved by BC. For task sets with $\frac{s_i}{e_i} = 0.2$, Our-FP and BC achieved almost identical schedulability results. This shows that the negative impact brought by suspensions is mainly caused by forcing $O(n)$ carry-in work.

Fig. 9 shows HRT schedulability results for arbitrary-deadline ordinary task systems (with no suspensions) achieved by using Our-FP and GY. In this experiment, for each choice of the utilization cap, 10,000 task sets are generated. As seen, Our-FP slightly improves upon GY. Moreover, Fig. 9 also shows the total time for running this entire experiment for Our-FP and GY. As seen, Our-FP runs much faster ($> 10\times$) than GY, due to the fact that Our-FP can find any task's response time by solving the RTA equation only once.

VI. Conclusion

We have presented HRT and SRT (with predefined tardiness thresholds) multiprocessor schedulability tests for arbitrary-deadline SSS task systems under both GFP and GEDF scheduling. In experiments presented herein, our suspension-aware analysis results significantly improve upon the suspension-oblivious approach SC. Moreover, our analysis shows that by applying an interval analysis framework tailored to the arbitrary-deadline case, arbitrary-deadline task systems (both ordinary and self-suspending ones) can be analyzed much more efficiently. Experiments indicate that our analysis has much better runtime performance than the prior approach proposed in [8].

In future work, it would be interesting to investigate more precise and practical suspension patterns. That is, instead of assuming that each task's suspensions are simply upper-bounded and will not be interfered with by other tasks' suspensions, it would be interesting to allow a task's suspension lengths to be affected by other tasks' suspensions (e.g., due to contention when multiple tasks simultaneously access the same shared resource).

References

[1] T. Baker. A comparison of global and partitioned EDF schedulability tests for multiprocessors. In *Technical Report TR-051101, Florida State University*, 2005.
[2] S. Baruah. Techniques for multiprocessor global schedulability analysis. In *Proc. of the 28th RTSS*, pp. 119-128, 2007.
[3] S. Baruah and N. Fisher. The partitioned multiprocessor scheduling of sporadic task systems. In *Proc. of the 26th RTSS*, pp. 321-329, 2005.
[4] S. Baruah, A. Mok, and L. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proc. of the 11th RTSS*, pp. 182-190, 1990.
[5] M. Bertogna and M. Cirinei. Response-time analysis for globally scheduled symmetric multiprocessor platforms. In *Proc. of the 28th RTSS*, pp. 149-160, 2007.
[6] M. Blum, R. W. Floyd, V. Pratt, R. L. Rivest, and R. E. Trarjan. *Time bounds for selection*. JCSS, (4): 448-461, 1973.
[7] G. Elliott and J. Anderson. Real-World Constraints of GPUs in Real-Time Systems. In *Proceedings of the First International Workshop on Cyber-Physical Systems, Networks, and Applications*, pp. 48-54, 2011.
[8] N. Guan, M. Stigge, W. Yi, and G. Yu. New response time bounds for fixed priority multiprocessor scheduling. In *Proc. of the 30th RTSS*, pp. 387-397, 2009.
[9] S. Lee and B. Moon. Design of flash-based DBMS: an in-page logging approach. In *Proc. of the 2007 ACM SIGMOD Conf. on Management of Data*, pp. 55-66, 2007.
[10] H. Leontyev and J. Anderson. A unified hard/soft real-time schedulability test for global EDF multiprocessor scheduling. In *Proc. of the 29th RTSS*, pp. 375-384, 2008.
[11] C. Liu and J. Anderson. Improving the schedulability of sporadic self-suspending soft real-time multiprocessor task systems. In *Proc. of the 16th IEEE Int'l Conf. on Embedded and Real-Time Computing Sys. and Apps.*, pp. 14-23, 2010.
[12] C. Liu and J. Anderson. Scheduling suspendable, pipelined tasks with non-preemptive sections in soft real-time multiprocessor systems. In *Proc. of the 16th IEEE Real-Time and Embedded Tech. and Apps. Symp.*, pp. 23-32, 2010.
[13] C. Liu and J. Anderson. An O(m) analysis technique for supporting real-time self-suspending task systems. In *Proc. of the 33rd RTSS*, pp. 373-382, 2012.
[14] C. Liu and J. Anderson. Task scheduling with self-suspensions in soft real-time multiprocessor systems. In *Proc. of the 30th RTSS*, pp. 425-436, 2009.
[15] C. Liu, J. Li, W. Huang, J. Rubio, E. Speight, and X. Lin. Power-efficient time-sensitive mapping in cpu/gpu heterogeneous systems. In *Proc. of the 21st International Conference on Parallel Architectures and Compilation Techniques*, pp. 23-32, 2012.
[16] J. Liu. *Real-time systems*. Prentice Hall, 2000.
[17] F. Ridouard, P. Richard, and F. Cottet. Negative results for scheduling independent hard real-time tasks with self-suspensions. In *Proc. of the 25th RTSS*, pp. 47-56, 2004.