Bursty-Interference Analysis Techniques for Analyzing Complex Real-Time Task Models

Cong Liu
Department of Computer Science
The University of Texas at Dallas

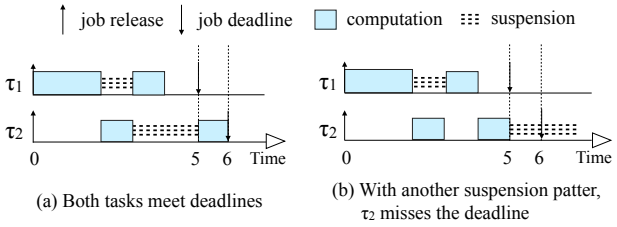
Jian-Jia Chen
Department of Informatics
TU Dortmund University, Germany

***Abstract*—Due to the recent trend towards building complex real-time cyber-physical systems, system designers need to develop and choose expressive formal models for representing such systems, as the model should be adequately expressive such that it can accurately convey the relevant characteristics of the system being modeled. Compared to the classical sporadic task model, there exist a number of real-time task models that are more expressive. However, such models are often complex and thus are rather difficult to be analyzed efficiently. Due to this reason, prior analysis methods for dealing with such complex task models are pessimistic. In this paper, a novel analysis technique, namely the bursty-interference analysis, is presented for analyzing two common expressive real-time task models, the general self-suspending task model and the deferrable server task model. This technique is used to derive new uniprocessor utilization-based schedulability tests and rate-monotonic utilization bounds for the two considered task models scheduled under rate-monotonic scheduling. Extensive experiments presented herein show that our proposed tests improve upon prior tests in all scenarios, in many cases by a wide margin. To the best of our knowledge, these are the first techniques that can efficiently analyze the general self-suspending and deferrable server task models on uniprocessors.**

1 Introduction

The emerging trend towards building complex real-time cyber-physical systems that often integrate external and physical devices and are expected to handle a variety of workloads is posing many challenging problems to the real-time systems community. When developing and choosing a formal model for representing a real-time system, system designers need to design a model that is expressive enough to accurately convey the relevant system characteristics. Meanwhile, the model must be efficiently analyzable for it to be useful in system design and analysis. The classical sporadic task model [22], which has received significant amount of attention over the past many years, lies at one extreme of this tradeoff: many efficient algorithms have been designed to analyze this model, unfortunately, the expressiveness of this model is quite limited since it cannot express many of the complex runtime behaviors in practice. Compared to the sporadic task model, there exist a number of real-time task models that are more expressive. However, such models are often more complex and thus are rather difficult to be analyzed efficiently.

Among such expressive models, the self-suspending task model [12] and the server-base task model [27] are two useful models that can accurately convey the characteristics of many real-time embedded systems that are often seen in practice. The self-suspending task model can be used to represent systems where tasks may experience suspension delays when being blocked to access external devices and shared resources [13]–[15], [20]. For example, suspension delays introduced by accessing devices such as GPUs could range from a few milliseconds to several seconds [7]. The server-based task model is often used to provide improved aperiodic response time performance when a system contains both periodic/sporadic tasks and aperiodic jobs. For example, it can be used to model a typical multimedia component deployed in an avionics system that consists of both hard real-time (HRT) periodic control tasks and soft real-time aperiodic media activities [1], [6]. Specifically, a deferrable server (DS) is defined by its budget and replenishment period [11] to serve incoming requests/jobs when the budget is still positive.



(a) Both tasks meet deadlines

(b) With another suspension patter, τ_2 misses the deadline

Fig. 1: Example RM schedules illustrating a different suspension pattern may cause the same task system to miss deadlines.

Unfortunately, the state-of-the-art schedulability analysis techniques are not able to analyze these two models efficiently and effectively. Regarding the self-suspending task model, negative results have been reported indicating that the problem of analyzing HRT task systems with suspensions on a uniprocessor is hard [25]. For instance, if tasks are allowed to self-suspend just once, then the earliest-deadline-first (EDF), which is optimal for scheduling sporadic task systems on a uniprocessor, cannot define a feasible schedule even under a k-speed processor, where k can be an arbitrarily large constant, while there exists a feasible schedule under a unit speed processor. Moreover, for a system as proposed in [11] that consists of multiple DSs and multiple sporadic real-time tasks, there exists no efficient utilization-based analysis to provide sufficient schedulability tests [5], [19], [27].

An intuitive reason behind the cumbersomeness of analyzing these two complex task models is because it is often hard to accurately analyze the worst-case amount of interference due to higher-priority tasks on the task that is being analyzed. This is due to the fact that such task models introduce execution behaviors that are relatively undeterministic (e.g., compared to the sporadic task model) and may heavily depend on the actual runtime execution behavior, i.e., the suspension patterns in the self-suspending task model and the arrival patterns of aperiodic events to be served by the deferrable servers. Fig. 1 shows

two rate-monotonic (RM) schedules of the same task system containing two suspending tasks: $\tau_1(3, 1, 5)$ and $\tau_2(2, 2, 6)$, where the notation $\tau_i(a, b, c)$ denotes that task τ_i has a total computation time of a time units, a total suspension time of b time units, and a period as well as a relative deadline of c time units. As seen in Figs. 1(a) and (b), different inputs of computation and suspension patterns may result in different schedules where deadlines may or may not be met for the same task system. The DS task model is similarly difficult to be analyzed in the sense that it is hard to predict a DS's execution pattern because it is hard to predict the aperiodic job arrival pattern, particularly when there exist multiple DSs with different assigned priorities.

Upon observing these difficulties, we show in this paper that the situation is not nearly so bleak. We consider the problem of deriving uniprocessor schedulability tests for HRT self-suspending task systems and DS task systems. We focus specifically on the RM scheduling algorithm, but out analysis could potentially be extended to apply to other task-level fixed priority scheduling algorithms as well. We first present a general analysis technique, namely the bursty-interference analysis, that can be used to analyze the worst-case scenario due to suspensions and server-based execution and derive the resulting schedulability tests. We then present novel transformation techniques that transform any self-suspending task system and any DS task system into a corresponding bursty-interference task system. We consider the general self-suspending task model where no restrictions are placed on the number of per-job computation and suspension phases and the phases interleaving pattern, and the DS task model where a DS task system may contain multiple DS tasks.

Overview of related work. For dealing with the general HRT self-suspending task model, the most commonly used approach is to transform a self-suspending task system into an ordinary sporadic task system by converting all tasks' suspensions into computation, called "SC" for brevity. Several approaches have been proposed to analyze a periodic self-suspending task model on a uniprocessor where each task is allowed to suspend for at most once and a fixed computation and suspension pattern is assumed [8], [10], [23], [24], [28]. 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 [19]). 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 analysis with pseudo-polynomial time complexity for periodic self-suspending task systems scheduled under global schedulers. [17] presents a uniprocessor utilization-base test under RM and a multiprocessor utilization-based test under partitioned approach where RM is applied as the per-processor scheduler. However, the analysis techniques and the tests presented in [17] only applies to *synchronous periodic* self-suspending task systems with *harmonic* periods.

In another recent work [3], we consider a special (yet common) case of the self-suspension task model, where any job of a self-suspending task is allowed to suspend for at most *once*. For such a special case, we show that the category of fixed-relative-deadline (FRD) scheduling algorithms may yield non-trivial resource-augmentation performance guarantees. We derive pseudo-polynomial-time and linear-time schedulability tests for a simple FRD scheduler and provide their resource-augmentation bounds by referring to different policies. This paper is completely different from the above-mentioned work [3], w.r.t. the targeted problem, the proposed analysis technique, and the format of the solution.

The existing schedulability analysis related to DSs can be categorized into two cases: (1) the feasibility and the schedulability of a DS task system [11], [27], to provide schedulability guarantees for the ordinary sporadic real-time tasks that are executed together with the DSs and the progressiveness of the deferrable servers, and (2) the worst-case response time analysis by using a deferrable server to serve aperiodic/sporadic tasks [4], [5], [9], [26]. When considering DS execution behaviors, we have to account for the *back-to-back execution* phenomena, in which the budget may be used at the very end before the next replenishment period. The worst-case response time analysis [4], [5], [9], [26] assumes that the budget under the given replenishment period of a DS is feasible and analyzes the worst-case behavior for serving the events by considering the interference imposed by other tasks/servers.

For the feasibility and the schedulability of a DS task system, Strosnider et al. [27] provide the necessary and sufficient schedulability conditions and a simple utilization-based test. However, the utilization-based sufficient test only considers a special case, in which there is only one DS with the highest priority [27]. To our best knowledge, there is no utilization-based sufficient schedulability test to validate the feasibility and the schedulability of the deferrable server system when there are more than one DS.

Contributions. In this paper, we propose the general bursty-interference analysis technique, and derive sufficient uniprocessor schedulability tests and an RM utilization bound for any bursty-intereference task system scheduled under RM (Section 3). We then present transformation techniques that transform any given self-suspending task system (Section 4) and any DS task system (Section 5) into a corresponding bursty-interference task system. We prove that if the transformed bursty-interference task system is schedulable, then the original task system is also schedulable. This transformation technique enables us to apply the schedulability tests and the RM utilization bound derived for bursty-interference task systems to self-suspending and DS task systems. To further reduce the pessimism of the derived schedulability tests, in Section 6, we present a more precise analysis that explores individual task parameters. As demonstrated by our experimental results in Section 7, our proposed tests improve upon prior methods with respect to schedulability, in many cases by a wide margin. Moreover, the precise analysis exploring individual task parameters is shown to be most effective in improving schedulability. We expect that our proposed bursty-interference analysis framework could be applied to efficiently analyze other complex task models under which undeterministic task execution behaviors and bursty interference may occur.

2 System Model

We model a given real-time system as a set $\Gamma = \{\tau_1, ..., \tau_n\}$ of n real-time tasks on a uniprocessor. In this paper, we

consider several real-time task models including the bursty-interference sporadic task model, the general self-suspending sporadic task model, and the DS task model. Since all these task models are based on the classical sporadic task model [22], we describe this model first.

The sporadic task model. A sporadic task τ_i is released repeatedly, with each such an invocation called a job. The j^{th} job of τ_i, denoted $\tau_{i,j}$, is released at time $r_{i,j}$ and has a deadline at time $d_{i,j}$. Each job of any task τ_i is assumed to have a worst-case execution time C_i. Successive jobs of the same task are required to execute in sequence. Associated with each task τ_i are a period T_i, which specifies the minimum time between two consecutive job releases of τ_i,[1] 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 $U_i = C_i/T_i$, and the utilization of the task system τ as $U_{sum} = \sum_{\tau_i \in \tau} U_i$. We assume that $U_i \leq 1$ and $U_{sum} \leq 1$; otherwise, deadlines would be missed. A sporadic task system τ is said to be an implicit-deadline system if $D_i = T_i$ holds for each τ_i. We limit attention to implicit-deadline real-time task systems in this paper.

We focus on RM scheduling, where tasks are prioritized by their periods. We index tasks such that $T_i \leq T_{i+1}$ for $1 \leq i \leq n-1$. We assume that ties are broken arbitrarily.

Bursty-intereference sporadic task set. A sporadic task set τ is a *bursty-interference sporadic task set* if the following property holds: when checking the schedulability of each task $\tau_k \in \tau$ under RM, the worst-case execution time of *at most one job* in each task τ_i in $\{\tau_1, ..., \tau_{k-1}\}$ is allowed to increase from C_i to $\alpha_i \cdot C_i$ where $\alpha_i \geq 1$. Such a job is called a *bursty job* and α_i is called the *bursty ratio* of τ_i. Note that, during the analysis of task τ_k, the value $\alpha_i \cdot C_i$ for each τ_i in $\{\tau_1, ..., \tau_{k-1}\}$ can be even larger than T_i for accounting the workload due to the bursty interference. Fig. 2(a) illustrates an example, where a higher-priority (w.r.t. the task that is being analyzed) task τ_i releases a bursty job $\tau_{i,j}$ with execution time $\alpha_i \cdot C_i$.

The general self-suspending sporadic task model. The general self-suspending sporadic task model extends the sporadic task model by allowing tasks to suspend themselves. Similar to sporadic tasks, a self-suspending sporadic task releases jobs sporadically. Jobs alternate between computation and suspension phases. We assume that each job of τ_i executes for at most C_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 assume that $C_i + S_i \leq T_i$ for any task $\tau_i \in \tau$; for otherwise deadlines would be missed. Our suspension model is general: we place no restrictions on the number of phases per-job and 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. Fig. 2(b) illustrates an example self-suspending task τ_i. As seen in the figure, jobs of τ_i may have completely different phase interleaving patterns.

The DS task model. A deferrable server (DS) σ_i is defined by its budget C_{σ_i} and replenishment period T_{σ_i} [11]. That is, if the j-th replenishment time is t, then the next replenishment time is $t + T_{S_i}$. Each DS is used to provide services, typically, for jobs with unknown arrival patterns or to isolate the interference from the unknown arrival patterns by providing progressiveness guarantees. The budget of a DS is consumed while it is used to serve some jobs. When the budget becomes 0, the DS cannot serve any further jobs until the next replenishment time. At the time instant to replenish the budget of DS σ_i, the budget is set to C_{σ_i}, i.e., any unused execution time budget at the end of each period is lost.

[1] τ_i becomes a periodic task if T_i specifies the exact time between two consecutive job releases.

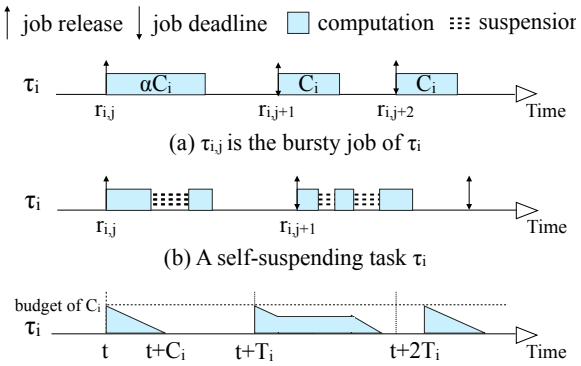


Fig. 2: Task models considered in this paper.

A task in a DS task system is either a sporadic real-time task or a DS. For notational brevity, we also denote a DS σ_i as τ_i, in which the budget is denoted by C_i and the replenishment period is T_i. To distinguish whether a task is a deferrable server or not, we define $\delta(\tau_i) = 1$ if task τ_i is a deferrable server and $\delta(\tau_i) = 0$ if task τ_i is an ordinary sporadic task. Fig. 2(c) illustrates an example DS task τ_i. As seen in the figure, a DS task may have different execution and budget consumption patterns within different replenishment periods, depending on the runtime aperiodic job arrival pattern.

3 Analysis under Bursty-Intereference

We derive in this section schedulability tests and a resulting RM utilization bound for bursty-interference sporadic task sets. Our analysis draws inspiration from Liu and Layland's seminal work [18] by quantifying the interference from higher-priority tasks on each analyzed task. For sporadic task systems, the worst-case releasing pattern is well-defined according to the well-known critical instant theorem [18]. The following theorem draws the corresponding critical instant theorem when considering bursty-interference sporadic task sets.

Theorem 1. *Consider a task τ_k in a bursty-interference sporadic task set scheduled under a fixed-priority scheduling policy and its job $\tau_{k,j}$. Suppose that every higher-priority task releases its bursty job at the same time as job $\tau_{k,j}$, and releases its subsequent jobs as early as possible. Let $R_{k,j}$ be the response time of job $\tau_{k,j}$. If the worst-case response time of task τ_k under this fixed-priority scheduling is no more than T_k, then its worst-case response time is $R_{k,j}$.*

Proof: This theorem is proved similarly to the well-known critical instant theorem [18]. It can be proved by applying the same "minimum phase" argument used in proving the

critical instant theorem (given on Pages 132-133 in [19]). For completeness, the detailed proof is given in Appendix. ■

Let τ_k denote the task that is being analyzed. Let $\Gamma(k-1)$ denote the higher-priority task set $\{\tau_1, ..., \tau_{k-1}\}$. Let α_{max} denote the maximum α_i values among tasks in $\Gamma(k-1)$. The following lemma provides the schedulability analysis based only on given utilizations and the bursty ratio α_{max}, by enumerating all *effective* settings of periods $T_1, T_2, \ldots, T_{k-1}, T_k$ under the given utilizations and α_{max}. A setting of periods $T_1, T_2, \ldots, T_{k-1}, T_k$ is called effective if

$$\forall 1 \leq j \leq k-1, \quad \sum_{i=1}^{k-1} \alpha_{max} \cdot C_i + C_k + \sum_{i=1}^{j-1} C_i > T_j, \quad (1)$$

in which C_i is defined as $U_i \cdot T_i$.

Lemma 1. *Given a constant α_{max}, T_k, and a set of task utilizations $U_1, ..., U_{k-1}$, task τ_k in a bursty-interference sporadic task set is schedulable under RM (i.e., the worst-case response time of τ_k is at most T_k), if for all effective settings of periods under Eq.* (1)*, the following condition holds*

$$\sum_{i=1}^{k-1} \alpha_{max} \cdot C_i + C_k + \sum_{i=1}^{k-1} C_i \leq T_k, \quad (2)$$

in which C_i is defined as $U_i \cdot T_i$.

Proof: We prove this lemma by contraposition. Suppose that the worst-case response time of τ_k is greater than T_k. Then there exists an invocation sequence of tasks in $\Gamma(k-1)$ with jobs released according to the releasing pattern given in Theorem 1, where only part of the execution time C_k is granted in time interval $[t, t+T_k)$.

Suppose that task τ_i releases more than 2 jobs in such an invocation sequence. Therefore, based on Theorem 1, $T_k/T_i > 2$ in such a case. Suppose F_i is defined as $\lfloor T_k/T_i \rfloor$. The workload of the jobs released by task τ_i in time interval $[t, t+F_i \cdot T_i)$ is $\alpha_{max} \cdot C_i + (\lfloor T_k/T_i \rfloor - 1) \cdot C_i$, and the job of task τ_k released at t cannot finish before $t + F_i \cdot T_i$. We can easily convert this invocation sequence by setting a new period T_i' to $\lfloor T_k/T_i \rfloor \cdot T_i$. In this new invocation sequence for task τ_i under the new period,

- due to the fact that $\alpha_{max} \geq 1$, the job released at time t has execution time $\alpha_{max} \cdot \lfloor T_k/T_i \rfloor \cdot C_i$, which is no less than the workload $\alpha_{max} \cdot C_i + (\lfloor T_k/T_i \rfloor - 1) \cdot C_i$ released by the original setting in time interval $[t, t+F_i \cdot T_i)$, and
- the job released at time $t + F_i \cdot T_i$ has execution time $\lfloor T_k/T_i \rfloor \cdot C_i$, which is larger than C_i.

Therefore, if $F_i \geq 2$, we can easily convert the sequence to another one with $F_i = 1$ such that task τ_k remains unschedulable under RM. As a result, we only have to focus on cases in which $F_i = 1$; i.e., $\forall 1 \leq j \leq k-1, T_k \leq 2 \cdot T_j \leq 2 \cdot T_k$ holds. Moreover, we also reindex the tasks such that after the above transformation, $T_i \leq T_{i+1}$ for each $i \leq k$. Note that this does not affect the unschedulability of τ_k.

By Theorem 1 and the above arguments to consider only the case $\forall 1 \leq j \leq k-1, T_k \leq 2 \cdot T_j \leq 2 \cdot T_k$, we know that each task in $\Gamma(k-1)$ releases two jobs in $[t, t+T_k)$ and the first job is a bursty job. Let $\tau_{k,h}$ be the job released by τ_k at t. Since $\tau_{k,h}$ does not complete by $t+T_k$, we know that Eq. (1) must hold (for otherwise $\tau_{k,h}$ would have completed before $t+T_k$). Likewise, we know that

$$\sum_{i=1}^{k-1} \alpha_{max} \cdot C_i + C_k + \sum_{i=1}^{k-1} C_i > T_k,$$

holds, for otherwise $\tau_{k,h}$ would have completed by $t+T_k$ under this effective setting of periods. Thus, Eq. (2) does not hold and the lemma follows by the contrapositive statement. ■

Lemma 1 indicates that task τ_k in a bursty-interference sporadic task set is schedulable under RM if the conditions in the lemma are verified. Based upon this lemma, the following lemma gives a utilization-based condition for checking the schedulability of any task τ_k in the system.

Lemma 2. *Any task τ_k in a bursty-interference sporadic task set τ is schedulable under RM if the following condition holds:*

$$U_k \leq 1 - (\alpha_{max} + 1) \cdot \left(1 - \frac{1}{\prod_{i=1}^{k-1} (U_i + 1)} \right). \quad (3)$$

Proof: We prove this lemma by showing that the condition in Eq. (3) leads to the satisfactions of the schedulability conditions listed in Lemma 1 under a given T_k, a given α_{max} and a given set of task utilizations $\{U_1, ..., U_{k-1}\}$.

By Lemma 1 and Eq. (2), we know that task τ_k in a bursty-interference sporadic task set τ is schedulable under RM if

$$U_k \leq 1 - \frac{(\alpha_{max} + 1) \cdot \sum_{i=1}^{k-1} C_i}{T_k} \quad (4)$$

holds for any effective settings of periods $T_1, T_2, \ldots, T_k$ under Eq. (1).

Therefore, we need to obtain the infimum (minimum) value of $1 - \frac{(\alpha_{max}+1) \cdot \sum_{i=1}^{k-1} C_i}{T_k}$ among all effective settings of periods under Eq. (1). In other words, we are looking for the supremum (maximum) value for $\frac{(\alpha_{max}+1) \cdot \sum_{i=1}^{k-1} C_i}{T_k}$ under Eq. (1). That is, if the utilization of task τ_k is less than or equal to this supremum value of $\frac{(\alpha_{max}+1) \cdot \sum_{i=1}^{k-1} C_i}{T_k}$, there does not exist any effective setting of periods $T_1, T_2, \ldots, T_{k-1}$ to make task τ_k unschedulable under RM.

For the rest of the proof, we replace $>$ with $\geq$ in Eq. (1), as the infimum and the minimum are the same when presenting the inequality with $\geq$. By reorganizing Eq. (1) and using the fact $C_i = U_i \cdot T_i$ for every task τ_i in $\Gamma(k-1)$, we have

$$\forall 1 \leq j \leq k-1, T_j \leq \sum_{i=1}^{k-1} \alpha_{max} \cdot U_i \cdot T_i + C_k + \sum_{i=1}^{j-1} U_i \cdot T_i. \quad (5)$$

Moreover, by reorganizing Eq. (2), we have

$$\sum_{i=1}^{k-1} \alpha_{max} \cdot U_i \cdot T_i \leq T_k - C_k - \sum_{i=1}^{k-1} U_i \cdot T_i. \quad (6)$$

By substituting $\sum_{i=1}^{k-1} \alpha_{max} \cdot U_i \cdot T_i$ in Eq. (5) by Eq. (6), we have

$$\forall 1 \leq j \leq k-1, T_j \leq T_k - \sum_{i=j}^{k-1} U_i \cdot T_i. \quad (7)$$

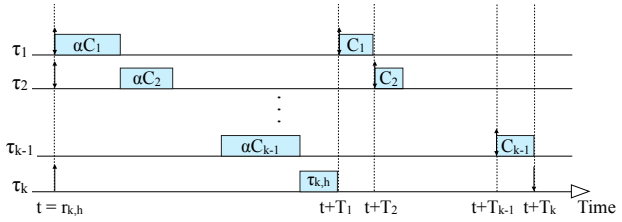


Fig. 3: The releasing pattern and task parameters that yield the maximum U_k such that τ_k is guarantee to be schedulable.

Since $C_i = U_i \cdot T_i$ holds for $1 \leq i \leq k$ and $U_1, ..., U_{k-1}$, T_k, and α_{max} are fixed, maximizing $\sum_{i=1}^{k-1} T_i$ implies that $\frac{(\alpha_{max}+1)\cdot\sum_{i=1}^{k-1} C_i}{T_k}$ is maximized. According to Eq. (7), the linear objective term $\sum_{i=1}^{k-1} T_i$ contains $k-1$ variables each of which has a corresponding linear constraint (given in Eq. (7)). Thus, according to the extreme point theorem for linear programming [21], $\sum_{i=1}^{k-1} T_i$ reaches its maximum when all the inequalities in Eqs. (7) become equalities.[2]

Thus, an effective setting of periods under Eq. (1) can maximize its interference when the inequalities in Eqs. (1) and (2) become equality. By treating the inequalities in Eqs. (1) and (2) as equalities and rearrangements, we have

$$\forall 1 \leq i \leq k-1, \quad T_{i+1} - T_i = C_i, \tag{8}$$

which implies that

$$\frac{T_1}{T_k} = \frac{1}{\prod_{i=1}^{k-1}(U_i+1)}. \tag{9}$$

These relationships among task periods result in a releasing pattern as illustrated in Fig. 3.

By Eq. (4), task τ_k in a bursty-interference sporadic task set τ is schedulable under RM if

$$\begin{aligned} U_k &\leq 1 - \frac{(\alpha_{max}+1)\cdot\sum_{i=1}^{k-1} C_i}{T_k} \\ &\overset{\{\text{By Eq. (8)}\}}{=} 1 - (\alpha_{max}+1)\cdot\left(1 - \frac{T_1}{T_k}\right) \\ &\overset{\{\text{By Eq. (9)}\}}{=} 1 - (\alpha_{max}+1)\cdot\left(1 - \frac{1}{\prod_{i=1}^{k-1}(U_i+1)}\right), \end{aligned} \tag{10}$$

which concludes the proof. ■

Based upon Lemma 2, the following schedulability test for bursty-interference sporadic task sets immediately follows.

Theorem 2. *Any bursty-interference sporadic task set τ is schedulable under RM if the following conditions hold:*

$$\forall 1 \leq k \leq n, U_k \leq 1 - (\alpha_{max}+1)\cdot\left(1 - \frac{1}{\prod_{i=1}^{k-1}(U_i+1)}\right). \tag{11}$$

The following theorem provides a sufficient RM utilization bound for any bursty-interference sporadic task set.

[2]For a linear programming, the linear constraints form a polyhedron of feasible solutions. The extreme point theorem states that either there is no feasible solution or one of the extreme points in the polyhedron is an optimal solution when the objective of the linear programming is finite. In our linear programming, there is only one extreme point in the polyhedron by setting all the inequalities in Eqs. (7) to equalities.

Theorem 3. *A bursty-interference sporadic task set τ is schedulable on a uniprocessor under RM if the following utilization-based condition holds for any $1 \leq k \leq n$:*

$$U(k) \leq k \cdot \left(\left(\frac{\alpha_{max}+1}{\alpha_{max}}\right)^{1/k} - 1\right) \tag{12}$$

Proof: Our objective is to find the minimum $U(k)$ such that Eq. (3) always holds. This is equivalent to the following non-linear programming:

$$\begin{aligned} &\text{minimize} \quad \sum_{i=1}^{k} U_k \\ &\text{such that} \quad 0 \leq U_k \leq 1 - (\alpha_{max}+1)\cdot\left(1 - \frac{1}{\prod_{i=1}^{k-1}(U_i+1)}\right). \end{aligned} \tag{13}$$

The above non-linear programming can be solved with Lagrange Multiplier method. We omit the details due to space constraints. Note that the solving procedure is the same as Step 4 in the proof of Theorem 1 in [18]. ■

By observing Eq. (12), when $\alpha_{max} = 1$ (i.e., Γ becomes an ordinary sporadic task set), the utilization bound shown in Theorem 3 indeed becomes identical to the RM utilization bound for ordinary sporadic task systems [18].

4 Suspension-to-Bursty System Transformation

In this section, we derive a utilization-based schedulability test and a sufficient RM utilization bound for sporadic self-suspending task systems. Our method is to transform a given self-suspending sporadic task system Γ into a corresponding bursty-intereference sporadic task system Γ^B, and then apply Theorems 2 and 3.

This transformation process consists of three stages, where we show that if a task τ_k is schedulable in one stage, then it is also schedulable in the previous stage. We prove these steps via contraposition, as illustrated in Fig. 4. Thus, by applying Theorems 2 and 3 to derive schedulability tests for Γ^B, we also obtain sufficient schedulability tests for the original self-suspending task system Γ.

Our technique checks schedulability of each task in Γ in order. Let τ_k denote the task that is being analyzed. Since tasks τ_j where $j > k$ do not affect the scheduling of τ_k, we ignore such tasks in the schedule. Before defining the corresponding Γ^B, we first convert τ_k's suspensions into computation and then analyze the resulting RM schedule. That is, we treats τ_k as an ordinary sporadic task by factoring its suspension length into the worst-case execution time parameter. Thus, τ_k executes just like an ordinary sporadic task (without suspensions) in the corresponding RM schedule, with an execution time of $C_k + S_k$. Note that τ_k's computation (both the original computation and the computation converted from suspensions) will be preempted by higher-priority tasks. Let Γ^C denote the task system after converting τ_k's suspension into computation. For clarity, we do not modify the value of U_k in Γ^C, i.e., τ_k in Γ^C has a utilization of $U_k + S_k/T_k$.

Lemma 3. *If τ_k in Γ is not schedulable under a fixed-priority scheduling algorithm, then τ_k in Γ^C is also not schedulable under the algorithm.*

Proof: The proof is given in Appendix. ■

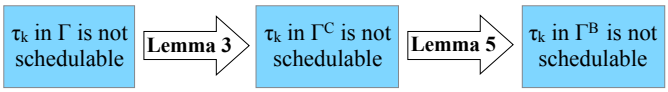


Fig. 4: The proof logic behind Theorem 4.

We now provide a worst-case releasing pattern for τ_k in Γ^C, where τ_k may experience the maximum interference from each of its higher-priority tasks. Let $\Gamma^C(k-1)$ denote the set of tasks in Γ^C with higher priorities than τ_k in Γ^C.

Lemma 4. *For any $0 \leq t_o \leq T_k$, the interference due to a higher-priority task $\tau_i \in \Gamma^C(k-1)$ within interval $[t, t+t_o]$ on any job of τ_k in Γ^C is at most $C_i + \lceil t_o/T_i \rceil \cdot C_i$.*

Proof: The proof is given in Appendix. ■

In the transformation process herein, the goal is to transform a self-suspending task system into a corresponding bursty task system and apply the schedulability test presented in Sec. 3. We now define the bursty-interference task system Γ^B that corresponds to Γ^C.

Task τ_k in Γ^B is identical to the corresponding task τ_k in Γ^C. Each task τ_i with higher priority than τ_k in Γ^B corresponds to each task τ_i in Γ^C with modified parameters. Let C_i^B and T_i^B denote the execution time and period of τ_i in Γ^B, respectively. We define these parameters as follows. If τ_i is a self-suspending task (i.e., $S_i > 0$), then we have

$$C_i^B = \lfloor T_k/T_i \rfloor \cdot C_i, \tag{14}$$

$$T_i^B = \lfloor T_k/T_i \rfloor \cdot T_i, \tag{15}$$

and

$$\alpha_i^B = \frac{C_i + C_i^B}{C_i^B} = 1 + \frac{1}{\lfloor T_k/T_i \rfloor}. \tag{16}$$

On the other hand, if τ_i is an ordinary sporadic task (i.e., $S_i = 0$), then we have

$$\alpha_i^B = 1. \tag{17}$$

Note that τ_k in Γ^B has an execution cost of $C_k^B = C_k + S_k$ given that τ_k's suspensions have been converted to computation in Γ^C. Also note that according to Eqs. (14) and (15), each higher-priority task τ_i in Γ^B has the same utilization as the corresponding τ_i in Γ^C. Moreover, we define

$$\alpha_{max}^B = max(\alpha_i^B), 1 \leq i \leq k-1. \tag{18}$$

Note that by Eqs. (16) and (17) and the fact that $T_k \geq T_i$ $(1 \leq i \leq k-1)$, $1 \leq \alpha_{max}^B \leq 2$ holds.

Now we prove the second transformation step of our proof logic as shown in Fig. 4.

Lemma 5. *If τ_k in Γ^C is not schedulable, then τ_k in Γ^B is also not schedulable.*

Proof: We prove by contraposition. Suppose that τ_k in Γ^B is schedulable. According to Theorem 1, τ_k must be schedulable under its worst-case releasing pattern (as defined in Theorem 1). According to this releasing pattern defined in Theorem 1, the total interference on τ_k in Γ^B due to *any* higher-priority task τ_i within interval $[t, t+t_o]$ $(0 \leq t_o \leq T_k)$ is:

$$\begin{aligned}
&(\alpha_{max}^B - 1) \cdot C_i^B + \lceil t_o/T_i^B \rceil \cdot C_i^B \\
&\{\text{By Eqs. (14)-(16) and (18)}\} \\
\geq &C_i + \left\lceil \frac{t_o}{\lfloor T_k/T_i \rfloor \cdot T_i} \right\rceil \cdot \left\lfloor \frac{T_k}{T_i} \right\rfloor \cdot C_i = C_i + \lceil t_o/T_i \rceil \cdot C_i.
\end{aligned}$$

Note that the last equality of the above equation holds due to the nested division property of ceiling functions and the fact that $\lfloor T_k/T_i \rfloor$ is a positive integer.

Thus, this total interference is no less than the maximum interference due to the corresponding task τ_i in Γ^C that τ_k in Γ^C could experience, which is at most $C_i + \lceil t_o/T_i \rceil \cdot C_i$ as given in Lemma 4. Therefore, if τ_k in Γ^B is schedulable, then τ_k in Γ^C is also schedulable. ■

According to our proof logic shown in Fig. 4, we prove the following schedulability condition on the analyzed task τ_k.

Theorem 4. *Any task τ_k in the original self-suspending task set Γ is schedulable under RM if the following condition holds:*

$$U_k + \frac{S_k}{T_k} \leq 1 - (\alpha_{max}^B + 1) \cdot \left(1 - \frac{1}{\prod_{i=1}^{k-1}(U_i + 1)} \right). \tag{19}$$

Proof: By Theorem 2, τ_k in Γ^B is schedulable if Eq. (19) holds. Thus, by Lemmas 3 and 5, the theorem follows. ■

If we apply the above test (Theorem 4) to every task in the system, then we have the following theorem.

Theorem 5. *Any sporadic self-suspending task set is schedulable under RM if the following conditions hold:*

$$\forall 1 \leq k \leq n, U_k + \frac{S_k}{T_k} \leq 1 - (\alpha_{max}^B + 1) \cdot \left(1 - \frac{1}{\prod_{i=1}^{k-1}(U_i + 1)} \right). \tag{20}$$

By applying Theorem 3 to Γ^B, the following utilization bound on the original self-suspending task system Γ follows.

Theorem 6. *A self-suspending sporadic task set Γ is schedulable under RM if the following utilization-based condition holds for any $1 \leq k \leq n$:*

$$U(k) + \frac{S_k}{T_k} \leq k \cdot \left(\left(\frac{\alpha_{max}^B + 1}{\alpha_{max}^B} \right)^{1/k} - 1 \right) \tag{21}$$

Corollary 1. *A self-suspending sporadic task set Γ is schedulable under RM if the following utilization-based condition holds for any $1 \leq k \leq n$:*

$$U(k) + \frac{S_k}{T_k} \leq \ln \frac{\alpha_{max}^B + 1}{\alpha_{max}^B}, \tag{22}$$

in which $\ln \frac{\alpha_{max}^B + 1}{\alpha_{max}^B} \geq \ln \frac{3}{2} \approx 0.405$ due to the fact that $\alpha_{max}^B \leq 2$.

Proof: This comes from the fact that $\left(\left(\frac{\alpha_{max}^B + 1}{\alpha_{max}^B} \right)^{1/k} - 1 \right)$ converges to $\ln \frac{\alpha_{max}^B + 1}{\alpha_{max}^B}$ when $k \to \infty$. ■

5 DS-to-Bursty System Transformation

We now apply the bursty-interference analysis to analyze DS task systems and obtain a utilization-based schedulability test and a sufficient RM utilization bound for such systems. Before describing the analysis, we first elaborate on the definition of schedulability for any given DS task system.

If task τ_i is a sporadic real-time task, i.e.,$\delta(\tau_i) = 0$, we have to provide the schedulability test to verify whether task τ_i is going to meet its deadline in the DS task system. However, if task τ_i is a deferrable server, i.e., $\delta(\tau_i) = 1$, the terminology of schedulability is a bit blur. That is, the deferrable server does not need to guarantee that its budget will always be set to 0 before the replenishment period. Therefore, even if the budget is still positive and there are jobs to be served by the deferrable server, these jobs may not be served before the next replenishment period (because this deferrable server may be preempted by other higher-priority deferrable servers or tasks).

However, deferrable servers are usually provided to ensure the *progressiveness*. We say that a deferrable server provides *progressiveness* guarantee if $\min\{C_k, X_{k,t}\}$ amount of execution time is successfully executed by the DS within $t + T_k$, where t is *any replenishment time* of the DS and $X_{k,t}$ is the execution time of the unfinished jobs to be served by the DS at time t. That is, if $X_{k,t} \geq C_k$, then the progressiveness guarantee (defined above) ensures that the budget C_k of the deferrable server is consumed; if $X_{k,t} < C_k$, all of these jobs will be finished before $t + T_k$. A DS task system τ is said schedulable under RM when

- if task τ_k is a sporadic real-time task, i.e., $\delta(\tau_k) = 0$, the worst-case response time of task τ_k is no more than its period T_k, and
- if task τ_k is a deferrable server, i.e., $\delta(\tau_k) = 1$, the progressiveness guarantee (defined above) to consume $\min\{C_k, X_{k,t}\}$ budget is ensured.

The above definition provides a unified view while testing whether task τ_k is schedulable or with progressiveness guarantee. When $X_{k,t} < C_k$, this can be considered as early completion in sporadic real-time tasks. When $X_{k,t} \geq C_k$, only C_k is considered to be executed before the next replenishment period. Therefore, according to the above definitions, we do not have to distinguish the test to verify the schedulability guarantee or progressiveness guarantee for testing τ_k.

We now provide the analysis of the workload from the higher-priority tasks $\tau_1, \tau_2, \ldots, \tau_{k-1}$. When a deferrable sever τ_i, with $i < k$, has a positive budget but does not have any job to serve, this is equivalent to the self-suspension behavior. One can imagine that a DS *suspends* itself for such cases. Therefore, *the execution behavior of a DS τ_i is a self-suspending task with $S_i \leq T_i$*. Note that the above observation is only used for analyzing the maximum workload of task τ_i while analyzing task τ_k. This is independent upon the schedulability or progressiveness guarantee of task τ_i.

Therefore, similar to the argument given in Section 4, we can consider task τ_i with a bursty ratio $\alpha_i^B = 1 + \frac{1}{\lfloor T_k/T_i \rfloor}$ (like in Eq. (16)) if $\delta(\tau_i) = 1$ or $\alpha_i^B = 1$ if $\delta(\tau_i) = 0$ (like in Eq. (17)). Moreover, α_{max}^B is defined as in Eq. (18). As a result, (with the same arguments in Lemma 5 and Theorem 4), we know that τ_k is schedulable if $\delta(\tau_k) = 0$ or has progressiveness guarantee if $\delta(\tau_k) = 1$ under RM if

$$U_k \leq 1 - (\alpha_{max}^B + 1) \cdot \left(1 - \frac{1}{\prod_{i=1}^{k-1}(U_i + 1)}\right). \quad (23)$$

Similar to Theorems 5 and 6, we conclude this section with the following theorems.

Theorem 7. *A DS task system is schedulable for its sporadic real-time tasks and has progressiveness guarantee for its DSs under RM if the following condition holds:*

$$U_k \leq 1 - (\alpha_{max}^B + 1) \cdot \left(1 - \frac{1}{\prod_{i=1}^{k-1}(U_i + 1)}\right). \quad (24)$$

Theorem 8. *A DS system is schedulable for its sporadic real-time tasks and has progressiveness guarantee for its DSs under RM if the following utilization-based condition holds for any $1 \leq k \leq n$:*

$$U(k) \leq k \cdot \left(\left(\frac{\alpha_{max}^B + 1}{\alpha_{max}^B}\right)^{1/k} - 1\right) \quad (25)$$

6 Analysis by Using Different Bursty Ratios

In Sec. 3, we derive the schedulability tests for bursty-interference task systems using a single α_{max} to replace all the individual α_i values. This may result in unnecessary pessimism. In particular, such pessimism may become significant when we test schedulability of self-suspending task systems and DS task systems (as described in Secs. 4 and 5, respectively). In practice, a self-suspending task system often contains (sometimes a large number of) ordinary sporadic tasks with no suspensions. In this case, assuming a large α_{max} for any non-suspending task is pessimistic because $\alpha_i = 1$ for any of such tasks. Therefore, in this section, we intend to derive more precise analysis using individual α_i values, which results in improved schedulability tests.

Similar to Lemma 1, the following lemma provides the schedulability analysis based only on given utilizations and individual bursty ratios α_i ($1 \leq i \leq k-1$), by enumerating all *effective* settings of periods $T_1, T_2, \ldots, T_{k-1}, T_k$ under the given utilizations and α_i values ($1 \leq i \leq k-1$). A setting of periods $T_1, T_2, \ldots, T_{k-1}, T_k$ is called effective with individual bursty ratios if

$$\forall 1 \leq j \leq k-1, \quad \sum_{i=1}^{k-1} \alpha_i \cdot C_i + C_k + \sum_{i=1}^{j-1} C_i > T_j, \quad (26)$$

in which C_i is defined as $U_i \cdot T_i$ and the $k-1$ tasks are indexed such that $T_1 \leq T_2 \leq \cdots \leq T_{k-1} \leq T_k$.

Lemma 6. *We are given constants α_i ($1 \leq i \leq k-1$), T_k, and a set of task utilizations $U_1, ..., U_{k-1}$. The worst-case response time of τ_k is at most T_k, and τ_k in a bursty-interference sporadic task set is schedulable under RM if for all effective settings of periods under Eq.* (26)*, the following condition holds*

$$\sum_{i=1}^{k-1} \alpha_i \cdot C_i + C_k + \sum_{i=1}^{k-1} C_i \leq T_k, \quad (27)$$

in which C_i is defined as $U_i \cdot T_i$.

Proof: The proof is identical to the proof of Lemma 1 by replacing the α_{max} term with individual α_i terms. ■

For the schedulability analysis of task τ_k, the indexing of the first $k-1$ tasks does not matter if we do not constrain their periods. However, as we have focused our discussions on setting $T_1 \leq T_2 \leq ... \leq T_{k-1}$, the indexing will further constrain the possible assignments of the periods. To evaluate all possible cases, we have to evaluate all permutations of indexing the first k-1 tasks. Fortunately, the following lemma gives a utilization-based condition for checking the schedulability of any task τ_k in the system, and shows that a specific indexing of the first $k-1$ tasks yields the maximum interference that affects the schedulability of τ_k.

Lemma 7. *Given a set of higher-task utilizations $U_1, ..., U_{k-1}$ with corresponding bursty ratios $\alpha_1, \alpha_2, \ldots, \alpha_{k-1}$, in which $\alpha_1 \leq \alpha_2 \leq \cdots \leq \alpha_{k-1}$, bursty-interference sporadic task τ_k is schedulable under RM if the following conditions hold:*

$$U_k \leq 1 - \sum_{i=1}^{k-1} \left((\alpha_i + 1) \cdot U_i \cdot \frac{1}{\prod_{j=i}^{k-1}(U_j + 1)} \right). \quad (28)$$

Proof: We prove this lemma by showing that the condition in Eq. (28) leads to the satisfactions of the schedulability conditions listed in Lemma 6 under a given T_k, a given set of α_i values ($1 \leq i \leq k-1$) and a given set of task utilizations $\{U_1, ..., U_{k-1}\}$.

First we need to identify the relationships among task periods under which U_k reaches its maximum value that guarantees τ_k to be schedulable. Since this part of the proof is identical to the corresponding part of the proof of Lemma 2 by replacing the α_{max} value with individual α_i values, we move this part of the proof to Appendix.

U_k reaches its maximum value when the following equations hold:

$$\forall 1 \leq i \leq k-1, \quad T_{i+1} - T_i = C_j, \quad (29)$$

which implies that

$$\forall 1 \leq i \leq k-1, \quad T_{i+1} = T_i \cdot (1 + U_i), \quad (30)$$

and

$$\forall 1 \leq i \leq k-1, \quad \frac{T_i}{T_k} = \frac{1}{\prod_{j=i}^{k-1}(U_j + 1)}. \quad (31)$$

These relationships among task periods result in the same releasing pattern as illustrated in Fig. 3.

Our goal now is to identify a maximum value for U_k that guarantees to satisfy the schedulability conditions given in Lemma 6. Similar to the derivation of Eq. (10), by Eq. (33) and rearranging Eq. (27), we have

$$U_k \leq 1 - \sum_{i=1}^{k-1} \left((\alpha_i + 1) \cdot U_i \cdot \frac{1}{\prod_{j=i}^{k-1}(U_j + 1)} \right). \quad (32)$$

We observe from Eq. (32) that the value of the term $\sum_{i=1}^{k-1} \left((\alpha_i + 1) \cdot U_i \cdot \frac{1}{\prod_{j=i}^{k-1}(U_j+1)} \right)$ varies when tasks in $\Gamma(k-1)$ are indexed differently. Note that this is different from the case as seen in Eq. (10), where α_{max} is assumed for all tasks in $\Gamma(k-1)$. Eq. (10) shows that the indexing scheme for tasks in $\Gamma(k-1)$ does not affect the maximum feasible U_k value because the term $\prod_{i=1}^{k-1}(U_i + 1)$ remains the same regardless of the specific task indexing scheme.

Thus, in order to obtain the maximum value for U_k that guarantees to satisfy the schedulability conditions given in Lemma 6, we need to first determine the task indexing rule that yields the maximum value of $\sum_{i=1}^{k-1}(\alpha_i+1) \cdot C_i$ as seen in Eq. (32), which represents the total amount of workload due to tasks in $\Gamma(k-1)$ on $\tau_{k,j}$ in interval $[t, t + T_k)$.

We now prove that this total amount of workload reaches maximum when tasks are indexed by smallest-bursty-ratio-first. That is, τ_i in $\Gamma(k-1)$ has the i^{th} smallest bursty ratio among tasks in $\Gamma(k-1)$. This can be proved by using an index swapping argument. Suppose that there are two bursty tasks τ_i and τ_{i+1} in $\Gamma(k-1)$ with $\alpha_i \leq \alpha_{i+1}$. We calculate the workload difference between two indexing schemes: assigning τ_i a smaller index than τ_{i+1} and vice versa.

If we assign τ_i a smaller index (according to smallest-bursty-ratio-first), then we have $T_{i+1} = T_i \cdot (1+U_i)$ according to Eq. (30). On the other hand, if we assign τ_{i+1} a smaller index, then we obtain the new period $T'_{i+1} = T_i$ for τ_{i+1} and $T'_i = T_i \cdot (1 + U_{i+1})$ for τ_i (again according to Eq. (30)). By accounting the difference of the workload due to these two tasks within $[t, t + T_1)$ under these two indexing rules, we have (note that in the following equation we use the workload obtained under smallest-bursty-ratio-first to minus the workload obtained under the opposite indexing rule)

$$\begin{aligned}
&((\alpha_i + 1) \cdot U_i \cdot T_i + (\alpha_{i+1} + 1) \cdot U_{i+1} \cdot T_{i+1}) \\
&- \left((\alpha_{i+1} + 1) \cdot U_{i+1} \cdot T'_{i+1} + (\alpha_i + 1) \cdot U_i \cdot T'_i\right) \\
=&((\alpha_i + 1) \cdot U_i \cdot T_i + (\alpha_{i+1} + 1) \cdot U_{i+1} \cdot T_i \cdot (1 + U_i)) \\
&- ((\alpha_{i+1} + 1) \cdot U_{i+1} \cdot T_i + (\alpha_i + 1) \cdot U_i \cdot T_i \cdot (1 + U_{i+1})) \\
=&U_i \cdot U_{i+1} \cdot T_i \cdot (\alpha_{i+1} - \alpha_i) \geq 0.
\end{aligned}$$

Thus, by applying this priority swapping argument on all adjacent pairs of tasks in $\Gamma(k-1)$, we know that indexing tasks in $\Gamma(k-1)$ by smallest-bursty-ratio-first yields the maximum higher-priority workload in $[t, t + T_k)$, which is given by $\sum_{i=1}^{k-1}(\alpha_i + 1) \cdot C_i$. According to Eq. (32), U_k is minimized under this task indexing scheme as well. We want to emphasize that this specific indexing scheme is applied only to analytically obtain the maximum workload in $[t, t+T_k)$ due to tasks in $\Gamma(k-1)$, such that the resulting maximum value of U_k guarantees τ_k to be schedulable. It does not imply that we must index tasks according to this scheme. At runtime, tasks can be indexed according to different schemes.

According to the above discussion, we thus re-index tasks $\tau_1, ..., \tau_{k-1}$ such that $\alpha_i \leq \alpha_{i+1}$ for $1 \leq i \leq k-1$. Ties are broken arbitrarily. By Eq. (32), the lemma follows. ■

Lemma 7 presents a tighter test by using individual bursty ratios when checking schedulability for each task. It is clear that we can directly apply this test to both self-suspending task systems and DS task systems (similar to the way of applying Lemma 2 to Theorems 4 and 5). For conciseness, we present the following general corollary instead of rewriting Theorems 4, 5, and 7 in detail.

Corollary 2. *Theorems 4, 5, and 7 are valid by replacing the right-hand side of Eqs. (19), (20), and (24) by the right-hand side of Eq. (28) by indexing the higher priority tasks using smallest-bursty-ratio-first when calculating Eq. (28).*

7 Experiment

In this section, we describe experiments conducted using randomly-generated task sets to evaluate the applicability of our proposed schedulability tests. Since the schedulability tests for DS task systems are derived by transforming DSs to suspending tasks (as discussed in Sec. 5), we focused on evaluating the schedulability tests that handle suspending task systems in the experiments.

Specifically, we evaluated our derived RM utilization bound (Theorem 6), utilization-based test using a single maximum bursty ratio (Theorem 5), and the more precise utilization-based test for suspending tasks using individual bursty ratios (Corollary 2), denoted by "Sum-U-Test", "Hyperbolic-U-Test", and "Individual-Bursty-U-Test", respectively. We compare these three tests against the common approach of treating suspensions as computation combined with the RM and the EDF utilization bound [19], denoted by "SC-RM" and "SC-EDF", respectively. Specifically, a task set is schedulable under SC-RM (SC-EDF) if $\sum_{i=1}^{n}(U_i+S_i/T_i) \leq 0.693$ ($\sum_{i=1}^{n}(U_i + S_i/T_i) \leq 1$). Note that SC-RM and SC-EDF are the only existing approaches that can handle the general HRT suspending task model on uniprocessors.

Experimental setup. In our experiments, suspending task sets were generated in a similar manner to the method used in [2], [12]. Task periods were uniformly distributed over [20*ms*,200*ms*]. Task utilizations were distributed uniformly in [0.005,0.2]. Task execution costs were calculated from periods and utilizations. Suspension lengths of tasks were distributed using three uniform distributions: $[0.005 \cdot T_i, 0.1 \cdot T_i]$ (suspensions are short), $[0.1 \cdot T_i, 0.3 \cdot T_i]$ (suspensions are moderate), and $[0.3 \cdot T_i, 0.5 \cdot T_i]$ (suspensions are long). The percentage of suspending tasks in each generated task set was set using three values: $0.6 \cdot n$ (moderate number of suspending tasks), $0.8 \cdot n$ (large number of suspending tasks), $1 \cdot n$ (all tasks are suspending tasks). We varied the total system utilization U_{sum} within $\{0.01, 0.02, ..., 1\}$. For each combination of suspension length distribution, suspending task percentage, and U_{sum}, 10,000 task sets were generated. 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 equalled the utilization cap. For each generated system, HRT schedulability was checked for SC-RM, SC-EDF, Sum-U-Test, Hyperbolic-U-Test, and Individual-Bursty-U-Test.

Results. The obtained schedulability results are shown in Fig. 5 (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, Individual-Bursty-U-Test achieves the best performance, in many cases improving upon SC-RM and SC-EDF by a substantial margin. For example, as seen in Fig. 5(b), when suspension lengths are moderate and suspending task percentage is moderate, Individual-Bursty-U-Test can achieve 100% schedulability when U_{sum} equals 0.36, while SC-RM and SC-EDF fail to do so when U_{sum} merely exceeds 0.03 and 0.1, respectively. Compared to Hyperbolic-U-Test, Individual-Bursty-U-Test is able to achieve better schedulability because using individual bursty ratios enables tighter tests, particularly when the generated tasks contain ordinary sporadic tasks.

Moreover, in most tested scenarios, Hyperbolic-U-Test achieves better performance than SC-RM and SC-EDF, particularly when suspending task percentage becomes larger and suspension lengths become longer, as seen in Figs. 5(h) for instance. Furthermore, Sum-U-Test achieves better schedulability than SC-RM in many cases, particularly when suspension lengths become longer and/or the suspending task percentage becomes larger, as seen in Figs. 5(e), (f), (h), and (i). However, when suspension lengthes are short or moderate, SC-RM achieves similar and sometimes better performance than SC-RM. This is because when suspension lengths are short, the utilization loss caused by converting suspensions into computation becomes smaller under SC-RM. Due to this same reason and the fact that EDF is an optimal uniprocessor scheduler, in many cases SC-EDF achieves better performance than Sum-U-Test, particularly when suspension lengths are short, as seen in Figs. 5(a), (d), and (g). Another interesting observation is that when suspension lengths increase and/or suspending task percentages increase, the improvement margins by Individual-Bursty-U-Test and Hyperbolic-U-Test over SC-RM and SC-EDF increase. This is because in these cases, the utilization loss caused by treating suspensions as computation increases under the SC approach. On the other hand, the suspension-related utilization loss under Individual-Bursty-U-Test and Hyperbolic-U-Test is due to the single S_k/T_k term when testing each task τ_k in the system (as seen on the left hand side of Eqs. (20) and (28)), and thus is not similarly affected.

8 Conclusion

We have presented a novel bursty-interference analysis technique for analyzing HRT self-suspending task systems and DS task systems scheduled under RM on uniprocessors. The resulting utilization-based schedulability tests improve upon prior methods, in many cases by a wide margin, as demonstrated by experiments presented herein. We expect that this analysis framework could be applied to analyze other complex real-time task models under which undeterministic task execution behaviors and bursty interference may occur.

References

[1] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings of the 19th IEEE Real-Time Systems Symposium*, pages 4–13, 1998.

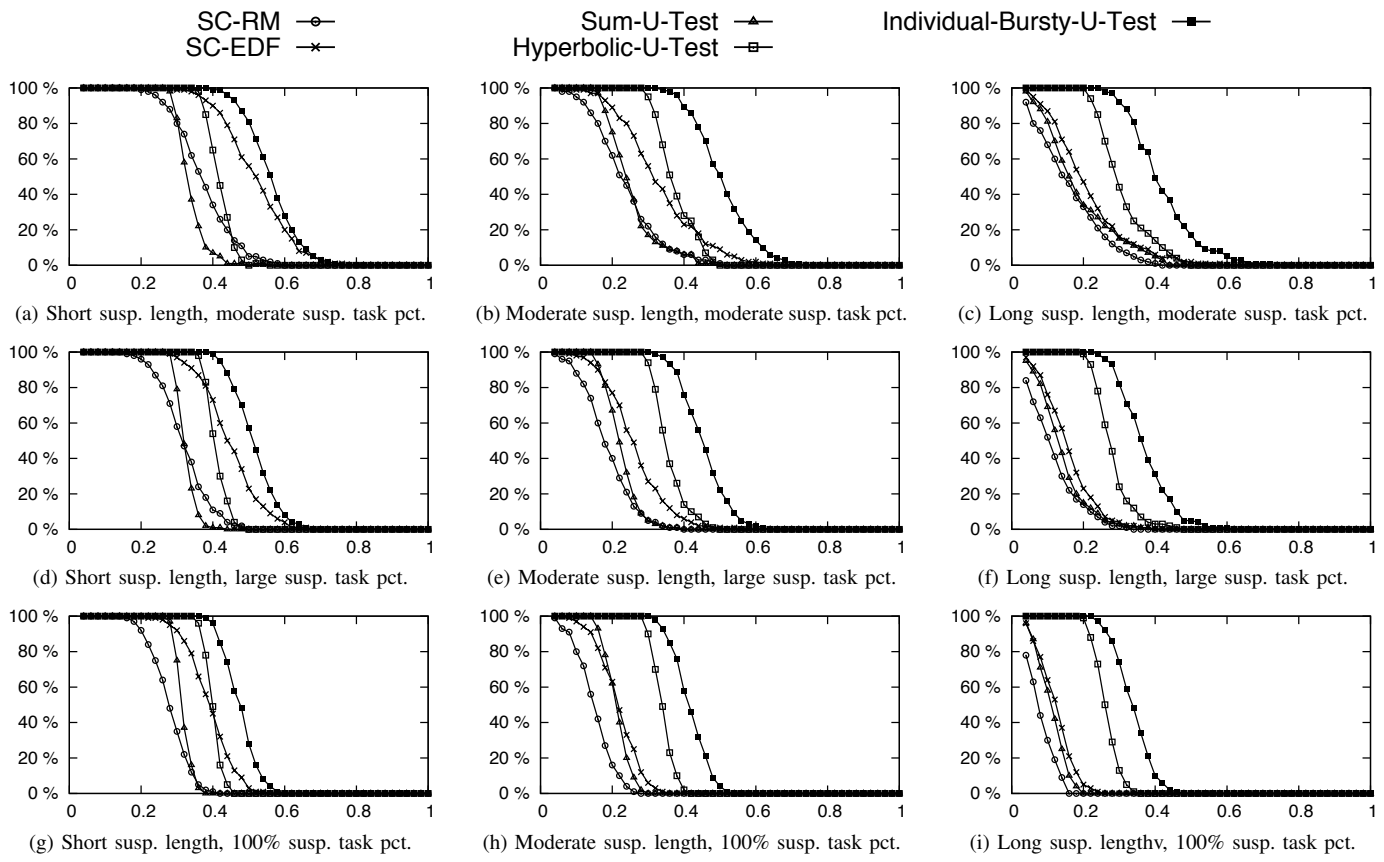
[2] T. P. Baker. A comparison of global and partitioned edf schedulability tests for multiprocessors. In *Proceedings of the International Conference on Real-Time and Network Systems*. Citeseer, 2005.

[3] J. Chen and C. Liu. Fixed-relative-deadline scheduling of hard real-time tasks with self-suspensions. In *Proceedings of the 35th Real-Time Systems Symposium*, to appear, 2014.

[4] P. J. L. Cuijpers and R. J. Bril. Towards budgeting in real-time calculus: deferrable servers. In *FORMATS: International conference on Formal modeling and analysis of timed systems*, pages 98–113, 2007.

[5] R. I. Davis and A. Burns. Hierarchical fixed priority pre-emptive scheduling. In *Proceedings of the 25th IEEE Real-Time Systems Symposium*, pages 389–398, 2005.

[6] G. Fohler, T. Lennvall, and G. Buttazzo. Improved handling of soft aperiodic tasks in offline scheduled real-time systems using total bandwidth server. In *Emerging Technologies and Factory Automation*, pages 151–157. IEEE, 2001.



(a) Short susp. length, moderate susp. task pct. (b) Moderate susp. length, moderate susp. task pct. (c) Long susp. length, moderate susp. task pct.

(d) Short susp. length, large susp. task pct. (e) Moderate susp. length, large susp. task pct. (f) Long susp. length, large susp. task pct.

(g) Short susp. length, 100% susp. task pct. (h) Moderate susp. length, 100% susp. task pct. (i) Long susp. lengthv, 100% susp. task pct.

Fig. 5: Schedulability results. In all nine 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 and third) column of graphs, short (respectively, moderate and long) suspension lengths are assumed. In the first (respectively, second and third) row of graphs, moderate (respectively, large and 100%) suspending task percentages are assumed. Each graph gives five curves representing the considered 5 schedulability tests.

[7] S. Kato, K. Lakshmanan, A. Kumar, M. Kelkar, Y. Ishikawa, and R. Rajkumar. RGEM: A Responsive GPGPU Execution Model for Runtime Engines. In *Proceedings of the 32nd IEEE Real-Time Systems Symposium*, pages 57–66, 2011.

[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] P. Kumar, J.-J. Chen, L. Thiele, A. Schranzhofer, and G. C. Buttazzo. Real-time analysis of servers for general job arrivals. In *RTCSA*, pages 251–258, 2011.

[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. P. Lehoczky, L. Sha, and J. K. Strosnider. Enhanced aperiodic responsiveness in hard real-time environments. In *Proceedings of the 8th IEEE Real-Time Systems Symposium*, pages 261–270, 1987.

[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] C. Liu, J. Chen, L. He, and Y. Gu. Analysis techniques for supporting harmonic real-time tasks with suspensions. In *Proceedings of the 26th Euromicro Conference on Real-Time Systems*. IEEE, 2014.

[18] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

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

[20] W. Liu, J. Chen, A. Toma, T. Kuo, and Q. Deng. Computation offloading by using timing unreliable components in real-time systems. In *Proceedings of the 51st Annual Design Automation Conference*, pages 1–6, 2014.

[21] D. G. Luenberger and Y. Ye. *Linear and nonlinear programming*, volume 116. Springer, 2008.

[22] A. Mok. *Fundamental Design Problems of Distributed Systems for the Hard-real-time environment*. PhD thesis, Massachusetts Institute of Technology, 1983.

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

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

[25] 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.

[26] S. Saewong, R. Rajkumar, J. P. Lehoczky, and M. H. Klein. Analysis of hierarchical fixed-priority scheduling. In *Proceedings of the Euromicro Conference on Real-Time Systems*, pages 173–181, 2002.

[27] J. K. Strosnider, J. P. Lehoczky, and L. Sha. The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments. *IEEE Transactions on Computers*, 44(1):73–91, 1995.

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

Appendix

Proof of Theorem 1.

Proof: We first prove that if a job $\tau_{k,j}$ is released at the same time with a job of every high-priority task and all tasks release jobs as early as possible, then the response time of $\tau_{k,j}$ is the largest among all jobs of τ_k. This can be proved by applying the same "minimum phase" argument used in proving the critical instant theorem (given on Pages 132-133 in [19]).

Consider another job of task τ_k, which arrives the system at time t'. Let $\tau_{k,j'}$ denote this job released at t'. Due to the assumption, that the worst-case response time of task τ_k is no more than T_k, we know that the response time of job $\tau_{k,j'}$ is to finish the workload of job $\tau_{k,j'}$ and the jobs which have higher priority than job $\tau_{k,j'}$.

Let t_o be the latest time instant at or before t' where no task with higher priority than τ_k executes. If we (artificially) redefine the release time of $\tau_{k,j'}$ to be t_o, then the completion time of $\tau_{k,j'}$ remains unchanged but the response time of $\tau_{k,j'}$ may increase. Moreover, for tasks with higher priorities than τ_k (i.e., $\tau_1, ..., \tau_{k-1}$), if starting with τ_1, we left-shift its first job that may be released after t_o such that its first job is released at t_o, then τ_k's response time does not decrease. This is because of the definition of t_o and the fact that the resulting interference from higher priority tasks on $\tau_{k,j}$ could only increase after such left-shiftings. By applying this left-shifting argument to all higher-priority tasks $\tau_1, ..., \tau_{k-1}$, we have constructed a portion of the RM schedule where all τ_1, ..., τ_k release jobs together at time t_o.

Furthermore, in this reconstructed RM schedule, if the job of any higher-priority task released at time t_o is not a bursty job, we switch this job with the bursty job of the corresponding task in the schedule. Clearly this switching step cannot decrease $\tau_{k,j}$'s response time because it can only increase the amount of workload contributed by τ_i within $[r_{k,j}, r_{k,j} + R_{k,j}]$ that may preempt $\tau_{k,j}$.

Therefore, we have successfully constructed a portion of the RM schedule that is identical to that which occurs at the arrival time defined for job $\tau_{k,j}$. Moreover, the response time $R_{k,j}$ of job $\tau_{k,j}$ is at least that of $\tau_{k,j'}$ released at t_o, which concludes the theorem. ■

The proof of Lemma 3

Proof: Since task τ_k in Γ is not schedulable under a fixed-priority scheduling algorithm, there exists a job $\tau_{k,\ell}$ of task τ_k in which its execution cannot be finished before $t + T_k$, where t is the released time $r_{k,\ell}$ of job $\tau_{k,\ell}$. In the interval $(t, t+T_k]$, the system can idle or execute tasks with lower priority than τ_k by at most S_k amount of time; otherwise, job $\tau_{k,\ell}$ has to suspend more than S_k amount of time in this interval. In the setting of Γ^C, we can consider the same pattern for the other jobs, but only convert the self-suspension of task τ_k in Γ to computation time. The additional S_k amount of computation time of $\tau_{k,\ell}$ in Γ^C can only be granted when the processor is idle or executes tasks with lower priority than τ_k, which is in total at most S_k as explained above. Therefore, τ_k in Γ^C is also not schedulable. ■

The proof of Lemma 4

Proof: Within $[t, t + t_o]$, the work done by task τ_i in the worst case can be divided into three parts: (*i*) body jobs: jobs of τ_i with both release time and deadline in $[t, t + t_o]$, (*ii*) carry-in job: a job of τ_i with release time earlier than t and deadline in $[t, t + t_o]$, and (*iii*) carry-out job: a job of τ_i with release time in $[t, t + t_o]$ and deadline after $t + t_o$. Since the carry-in and the carry-out job can each contribute at most C_i workload in $[t, t + t_o]$, a safe upper bound of the interference due to task τ_i in $[t, t + t_o]$ is obtained by assuming that the carry-in and carry-out jobs of τ_i both contribute C_i each in $[t, t + t_o]$. The lemma thus follows. ■

The first part of the proof of Lemma 7.

Proof: For an effective setting of periods $T_1, T_2, \ldots, T_k$ under Eq. (26), by Lemma 6 and Eq. (27), we know that task τ_k in a bursty-interference sporadic task set τ is schedulable under RM if

$$U_k \leq 1 - \frac{(\alpha_i + 1) \cdot \sum_{i=1}^{k-1} C_i}{T_k}. \tag{33}$$

Therefore, we need to obtain the minimum value of $1 - \frac{(\alpha_i+1)\cdot\sum_{i=1}^{k-1} C_i}{T_k}$ among all effective settings of periods under Eq. (26). In other words, we are looking for the maximum value for $\frac{(\alpha_i+1)\cdot\sum_{i=1}^{k-1} C_i}{T_k}$ under Eq. (26).

By reorganizing Eq. (26) and using the fact that $C_i = U_i \cdot T_i$ holds for every task τ_i in $\Gamma(k-1)$, we have

$$\forall 1 \leq j \leq k-1, \frac{C_j}{U_j} \leq \sum_{i=1}^{k-1} \alpha_i \cdot U_i \cdot T_i + C_k + \sum_{i=1}^{j-1} U_i \cdot T_i. \tag{34}$$

Moreover, by reorganizing Eq. (27), we have

$$\sum_{i=1}^{k-1} \alpha_i \cdot U_i \cdot T_i \leq T_k - C_k - \sum_{i=1}^{k-1} U_i \cdot T_i. \tag{35}$$

Thus, by substituting $\sum_{i=1}^{k-1} \alpha_i \cdot U_i \cdot T_i$ in Eq. (34) by Eq. (35), we have

$$\forall 1 \leq j \leq k-1, \frac{C_j}{U_j} \leq T_k - \sum_{i=j}^{k-1} U_i \cdot T_i. \tag{36}$$

Since $C_i = U_i \cdot T_i$ holds for $1 \leq i \leq k$ and $U_1, ..., U_{k-1}$, and T_k are fixed, maximizing $\sum_{i=1}^{k-1}(\alpha_i + 1) \cdot T_i$ implies that $\frac{\sum_{i=1}^{k-1}(\alpha_i+1)\cdot C_i}{T_k}$ is maximized. According to Eq. (36) and the fact that α_i ($1 \leq \alpha_i \leq k-1$) values are constants, the linear objective term $\sum_{i=1}^{k-1}(\alpha_i+1)\cdot T_i$ contains $k-1$ variables each of which has a corresponding constraint (given in Eq. 36). Thus, according to the extreme point theory [21], $\sum_{i=1}^{k-1}(\alpha_i+1)\cdot T_i$ reaches its maximum when all the inequalities in Eqs. (36) become equalities. Since Eq. (36) is derived by reorganizing Eqs. (26) and (27), it implies that in this case all the inequalities inEqs. (26) and (27) become equalities.

Thus, U_k reaches its maximum value that guarantees τ_k to be schedulable when the inequalities in Eqs. (26) and (27) become equality. By treating the inequalities in Eqs. (26) and (27) as equalities and rearrangements, we obtain Eqs. (29)-(31). ■