A General Analysis Framework for Soft Real-Time Tasks

Zheng Dong, Cong Liu, *Member, IEEE*, Soroush Bateni, Zelun Kong, Liang He, *Senior Member, IEEE*, Lingming Zhang, Ravi Prakash, and Yuqun Zhang

Abstract—Much recent work has been conducted on supporting soft real-time tasks on multiprocessors due to the multicore revolution. While most earlier works focus on the traditional sporadic task model with deterministic worst-case specification, several recent works investigate the stochastic nature of many workloads seen in practice, specifying task execution times using average-case provisioning instead of the worst case. Unfortunately, all the existing work on supporting soft real-time workloads ignores a simple practical fact that the job inter-arrival time (or task period) is also stochastic for many real-world applications. Adopting a fixed worst-case period to model all the arriving pattern is rather pessimistic and may result in significant capacity loss in practice. Based on these observations, we present a general soft real-time multiprocessor schedulability analysis framework in this paper for practical sporadic task systems specified by stochastic period and execution demand, following probability distributions. Our analysis can be generally applied to global tunable priority-based schedulers, which allow any job's priority to be changed dynamically at runtime within a priority window of constant length. We have extensively evaluated the analysis framework using a MPEG video decoding case study and simulation-based experiments. Experimental results demonstrate significant advantages of our analysis, which yields over 200 and 50 percent improvements compared to existing analysis assuming worst-case task periods in terms of schedulability and magnitude of the derived tardiness bound, respectively.

Index Terms—Real-time scheduling, stochastic tasks, schedulability test, tardiness bound, probability distribution

1 Introduction

The growing prevalence of multicore platforms has stimulated much recent work on scheduling *hard* and *soft* real-time workloads on multiprocessor platforms. Due to various complexities that arise in multicore architecture such as cache and bus contentions, multicores are arguably better suited to support soft real-time (SRT) tasks than hard real-time (HRT) ones that require hard deadlines to be met. In practice, many applications run on multicore only require SRT constraints where deadlines can be occasionally missed but any such misses must be provably bounded. Examples include multimedia decoding [26], real-time video and image processing [7], and computer vision applications [28], [30] such as colliding face detection [33]. In such applications, providing predictable and short response times for individual video frames is important, which ensures smooth video output.

Traditionally, a worst-case system provisioning is used for modeling HRT applications [12], [23] to ensure timing correctness even in the worst case. The classical sporadic task model represents such a worst-case provisioning [24]: when a task's utilization specifies its long-term required processor share, a deterministic worst-case execution time and a fixed period denoting the minimum job inter-arrival time are assumed. A key difference in supporting SRT applications in real-time systems is the consideration of many such applications' *stochastic nature* [5] observed in practice. That is, the actual execution times of jobs released by a task are often stochastically distributed.[1] For SRT tasks that only require bounded response times, it is thus not worth conservatively dedicating a worst-case amount of processing time. For example, in computer vision [19], [25], [32], object recognition tasks are executed to recognize certain objects, where objects often arrive in a stochastic manner and analyzing different objects incurs different execution times. In light of this observation, several researchers have recently developed scheduling algorithms and schedulability test algorithms for SRT tasks with stochastic execution demand, which use an average-case provisioning to allocate processing capacity based on tasks' average-case execution times [27].

Research Motivation. While placing an average-case provisioning for a task's execution demand helps reduce the

- *Z. Dong is with the Department of Computer Science, University of Texas at Dallas, Richardson, TX 75080, and also with the Department of Computer Science and Engineering, Southern University of Science and Technology during his visit to the University in December, 2017. E-mail: zheng@utdallas.edu.*
- *C. Liu, S. Bateni, Z. Kong, L. Zhang, and R. Prakash are with the Department of Computer Science, University of Texas at Dallas, Richardson, TX 75080. E-mail: {cong, soroush, zelun.kong, lingming.zhang, ravip}@utdallas.edu.*
- *L. He is with the Department of Computer Science and Engineering, University of Colorado Denver, Denver, CO 80124. E-mail: liang.he@ucdenver.edu.*
- *Y. Zhang is with the Department of Computer Science and Engineering, Southern University of Science and Technology, and also with the Shenzhen Key Laboratory of Computational Intelligence, University Key Laboratory of Evolving Intelligent Systems, Shenzhen, Guangdong Province 518055, China. E-mail: zhangyq@sustc.edu.cn.*

Manuscript received 7 Apr. 2018; revised 19 Nov. 2018; accepted 21 Nov. 2018. Date of publication 6 Dec. 2018; date of current version 15 May 2019. (Corresponding author: Yuqun Zhang.)
Recommended for acceptance by P. Balaji.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2018.2884980

1. It is actually extremely hard (if possible) to derive accurate timing analysis tools that determine the worst-case execution times for multicore-based platforms [34].

unnecessary conservativeness in specifying an SRT task's required processor share (i.e., task utilization), an equally paramount parameter in defining task utilization has been largely ignored in the existing SRT systems research, i.e., the job inter-arrival time (or the task period). Assuming a minimum job inter-arrival time for an HRT task is often necessary for guaranteeing timing correctness in the worst case, but can be a serious impediment for modeling SRT tasks. Existing works ignore a simple fact in practice, i.e., job arrival patterns are stochastic in nature, which has been widely validated using real traces found in many application systems [14], [16]. For example, in many event-driven control systems, job releasing is triggered by stochastically occurring events [3], thus resulting in highly variable job inter-arrival patterns.

In many such application scenarios [6], [8], [11], [37], a pre-fixed minimum job inter-arrival time (i.e., a fixed task period parameter) could be particularly smaller than the job's average period. Clearly, similar to the worst-case execution time assumption for SRT tasks, assuming a minimum job inter-arrival time for SRT tasks is questionable in practice. Such an assumption may cause rather conservative task utilization provisioning and thus significant capacity loss for SRT systems in practice.

Inspired by those observations, we develop a multiprocessor scheduling analysis framework for supporting SRT workloads with stochastic periods and execution demand. We adopt common probability distributions as a natural model for describing execution times and periods that vary among different jobs. This general framework can be directly applied to analyze a wide set of global scheduling methods such as global earliest-deadline-first (EDF), global first-in-first-out (FIFO), and global least-laxity-first (LLF), or precisely any global tunable priority scheduler (TPS) that allows jobs' priorities to be dynamically changed within a priority range of constant length at runtime. We believe that this work closes the loop for supporting practical SRT tasks with stochastic characteristics on multiprocessors, generalizing previous work on deriving response time bounds assuming deterministic task periods and/or execution times [9], [18].

Related Work. Our work is built based upon the tardiness bound analysis in window constrained priority schedule by Leontyev and Anderson [18] and the expected tardiness bound analysis with stochastic execution times by Mills and Anderson [27]. Some of the presented results have also been similarly proved by Devi and Anderson [9], and Liu and Anderson [21], [22] for different task models assuming deterministic periods.

In [18], a tardiness bound analysis has been presented for scheduling SRT tasks under any window-constrained global scheduling algorithms. However, it assumes worst case task parameters. In [27], a tardiness bound analysis is presented under the global EDF scheduler for SRT tasks with deterministic periods and stochastic execution times. The derived expected (mean) tardiness bound in [27] thus cannot be applied to tasks with stochastic periods and any scheduler other than global EDF. The key differences between our analysis framework and the ones presented in [18] and [27] work are listed in Fig. 1.

Our Contribution. As discussed above, none of the existing techniques can be applied to derive expected (mean) tardiness bounds for truly stochastic SRT task systems with stochastic periods and execution times under a general set of global scheduling algorithms. This lack of support is the key motivation behind this work. Specifically, in this paper we propose a tardiness abound analysis under global scheduling for SRT task systems whose periods and execution times are defined stochastically. The resulting SRT schedulability condition only requires that the *expectation* of the task system's total utilization is no greater than the multiprocessor's capacity. Thus, the task system is allowed to be over-utilized under our analysis in the worst case scenario. Our analysis allows both tasks' periods and execution times to be stochastically specified, following probability distributions. Our analysis can be generally applied to any global tunable priority-based scheduler (TPS) that allows any job's priority to be changed dynamically at runtime within a priority window of constant length. TPS includes a large set of global schedulers such as EDF, FIFO, LLF, earliest deadline zero laxity (EDZL), etc. Finally, we have conducted evaluations to check the applicability of the proposed schedulability test and tardiness bounds. Experimental results demonstrate that our analysis framework can achieve over 500 and 90 percent improvements over the prior deterministic tardiness bound analysis [18], in terms of schedulability and magnitude of the derived tardiness bound. Similarly, compared with the experimental results yielded by the techniques given in [27], our analysis framework can achieve over 200 and 50 percent improvements over the tardiness bound analysis assuming stochastic execution times but deterministic periods, in terms of schedulability and magnitude of the derived tardiness bound.

Parameter	Parameter type	[18]	[27]	This Paper
Job inter-arrival pattern	Sporadic	✓	✓	✓
	Stochastic			✓
Execution time	Worst-case	✓		✓
	Stochastic		✓	✓
Priority assignment	EDF	✓	✓	✓
	FIFO	✓		✓
	Priority window	✓		✓
Preemptivity	Preemptive	✓	✓	✓
	Non-preemptive	✓		✓

Fig. 1. Key differences compared to previous work.

The rest of this paper is organized as follows. Section 2 introduces the formal task system model. Sections 3, 4, and 5 explain our main results. Section 6 shows a case study. Section 7 presents the evaluation results and Section 8 concludes.

2 System Model

In this paper, we study the problem of scheduling a set $\tau = \tau_1, \tau_2, \ldots, \tau_n$ of n independent real-time stochastic tasks on a multi-processor platform, which consists of m identical processors. A *task* may release an infinite sequence of jobs. The jth job of τ_l, denoted as $\tau_{l,j}$, has three parameters: release time $r_{l,j}$, execution time $e_{l,j}$, and priority value $\Upsilon_{l,j}(t)$ at time t. Jobs from the same task must be processed sequentially.

We consider the above system under the following assumptions:

TABLE 1
Notation Summary

Parameters	Descriptions
$e_{l,j}$	the execution time of $\tau_{l,j}$.
$p_{l,j}$	the period of $\tau_{l,j}$.
$\bar{e}_l$	the average execution time of $\tau_{l,j}$.
$\bar{p}_l$	the average period of $\tau_{l,j}$.
σ_l^2	the variance of τ_l's execution time.
$\sigma_l{}'^2$	the variance of τ_l's period.
m	the number of identical processors.
$\bar{u}_l$	τ_l's expected utilization.
$\bar{U}$	expected total utilization of all tasks.
$r_{l,j}$	the release time of $\tau_{l,j}$.
$\Upsilon_{l,j}(t)$	job prioritization function.
$f_{l,j}^*$	the completion time of $\tau_{l,j}$ in TPS.
$\hat{f}_{l,j}$	the completion time of $\tau_{l,j}$ in PS.
$\hat{u}_l$	the execution rate of τ_l in PS.
$\hat{e}_{l,j}$	$\max_{1\le j'\le j}\{e_{l,j'}\}$.
ρ	$\max_{\tau_i\in\tau}\phi_i + \max_{\tau_l\in\tau}\varphi_l$.
U_L	$\sum_{i=1}^{m-1}\hat{u}^i$.
$\hat{u}^i$	the ith largest value of $\{\hat{u}_i\}$.

Assumption 1. *The $e_{l,j}$ values are independent random variables with identical probability distributions. The mean is $\bar{e}_l$ and the variance is σ_l^2.*

Assumption 2. *Let $p_{l,j}$ be the time interval between the release times of two consecutive jobs $\tau_{l,j}$ and $\tau_{l,j+1}$ from the same task τ_l. The $p_{l,j}$ values are independent random variables with identical probability distributions. The mean is $\bar{p}_l$ and the variance is $\sigma_l'^2$.*

Note that we only require such distributions to have finite mean and variance, and the above assumed properties hold for any standard probability distributions, such as the exponential [4], Weibull [29] distribution. Moreover, we assume the expected value of the maximum execution time is upper bounded by ϵ_l and the consecutive inter release time is lower bounded by ζ_l—a maximum execution time and a minimum release time interval will suffice, where $\epsilon_l \ge e_{l,j}$ and $\zeta_l \le p_{l,j}$ for $j = 1, 2, \ldots$. These upper/lower bounds are needed in order to guarantee the correctness of our analysis. Based on the above definitions, $\bar{u}_l = \frac{\bar{e}_l}{\bar{p}_l}$ denotes τ_l's expected utilization. Therefore, the expected total utilization of the task system is denoted by $\bar{U} = \sum_{l=1}^{n}\bar{u}_l$.

Definition 1 (tardiness). *The first job of any task can be released at any time $t \ge 0$. $\tau_{l,j}$ is released at $r_{l,j}$ and its successive job $\tau_{l,j+1}$'s release time is $r_{l,j+1}$. If $\tau_{l,j}$ completes execution at $f_{l,j}^*$, then $\tau_{l,j}$'s* tardiness *is defined as* $\max(0, f_{l,j}^* - r_{l,j+1})$.

A task's tardiness is defined to be the maximum tardiness of all its jobs. We require that

$$\bar{U} < m, \tag{1}$$

and

$$\bar{u}_l \le 1. \tag{2}$$

Otherwise, tasks' tardiness will grow infinitely.

The release time of $\tau_{l,j+1}$ will not be altered when $\tau_{l,j}$ does not complete by $r_{l,j+1}$. Thus, at any time instant, a task may have at most one job execute, even if multiple jobs have been released by that task.

For readability, the notation used throughout this paper is summarized in Table 1.

2.1 Tunable Priority-Based Scheduler

We define the *Tunable Priority-based Scheduler* in this section. Note that the scheduling of jobs does not affect the release time and the execution time of any job $\tau_{l,j}$.

Definition 2 (Job prioritization functions). *Each released job has an associated prioritization function $\Upsilon_{l,j}(t)$. For any jobs $\tau_{l,j}$ and $\tau_{i,k}$, $\tau_{l,j}$ has higher priority than $\tau_{i,k}$ at t, if $\Upsilon_{l,j}(t) < \Upsilon_{i,k}(t)$, or $\Upsilon_{l,j}(t) = \Upsilon_{i,k}(t)$ and $l < i$.*

Since jobs' priorities are functions of time, they can be dynamically tuned at runtime within a time window of constant length, as defined below.

Definition 3 (Tunable window of priorities). *To guarantee each task has bounded tardiness, the priority of job $\tau_{l,j}$ can be tuned within a specific window:*

$$r_{l,j} - \phi_l \le \Upsilon_{l,j}(t) \le r_{l,j+1} + \varphi_l, \tag{3}$$

where $\phi_l \ge 0$ and $\varphi_l \ge 0$ are arbitrary constant values.

To illustrate the tunable property of jobs' priorities, we show an example to convert a preemptive EDF schedule into a non-preemptive schedule by tuning the jobs' priorities dynamically.

Example 1. We show how to convert a preemptive EDF schedule into a non-preemptive EDF schedule by tuning each job's priority in two steps: (i) when jobs are released, they are prioritized by their deadlines, and (ii) at any time instant t when a job is scheduled to execute on a processor, its priority is tuned to t. Then, the preemptive GEDF schedule is converted into a non-preemptive GEDF schedule.

Compare to traditional real-time schedulers, TPS is more flexible. TPS can dynamically simulate a set of schedules at runtime, and guarantee bounded tardiness for each task, for instance under EDF, FIFO, LLF, etc. Thus, TPS inherits all the practical benefits of these schedulers. Furthermore, TPS also can simulate non-preemptive scheduling: when $\tau_{l,j}$ gets into a non-preemptive region, set $\Upsilon_{l,j}(t)$ to be $r_{l,j} - \phi_l$, where ϕ_l is large enough to ensure that $\tau_{l,j}$'s priority is higher than that of any unscheduled or newly-released job.

Definition 4 (Pending Jobs). *In any schedule S, if $\tau_{l,j}$ does not complete execution by time instant t and $r_{l,j} \le t$, $\tau_{l,j}$ is pending at t. Task τ_l is pending at time instant t.*

Definition 5 (Pending Tasks). *Task τ_l is pending at time instant t if a job released by τ_l is pending at time instant t.*

Definition 6 (Ready Jobs). *In a schedule S, $\tau_{l,j}$ is ready at time instant t if all previous jobs released by τ_l have completed execution by time instant t and $r_{l,j} \le t$.*

Definition 7. *A TPS schedule has the following properties: (i) a single global priority queue is constructed to accommodate the released jobs in the priority decreasing order; (ii) a job's priority can be tuned to any value within its priority window (Definition 3) at any time after its release and before its completion; (iii) at each time instant where more than m jobs are ready, the m jobs, which have the highest priorities, are scheduled.*

2.2 Processor-Sharing

In this section, we introduce a processor-sharing (PS) schedule. Jobs released by different tasks will not preempt each

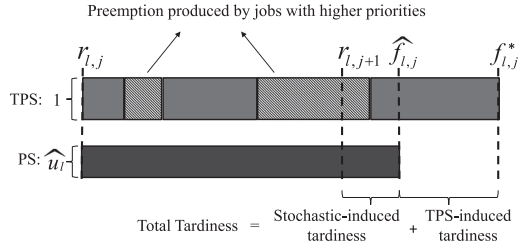


Fig. 2. $\tau_{l,j}$ executes in TPS schedule and PS schedule.

other in a PS schedule. PS is considered to be an ideal schedule where for each $\tau_l \in \tau$, at every time instant that τ_l has pending jobs, a fraction $\hat{u}_l$ of the processing capacity of one processor is allocated to τ_l, where

$$\sum_{l=1}^{n} \hat{u}_l \leq m, \tag{4}$$

$$\bar{u}_l \ < \ \hat{u}_l \leq 1, \forall \tau_l \in \tau. \tag{5}$$

Different choices of $\hat{u}_l$ will lead to different derived tardiness bounds (we will optimize the choice of $\hat{u}_l$ in Section 4). Sufficient processing capacity must be allocated to each task, which satisfies (4) and (5), in order to guarantee that each task has bounded tardiness in PS on m processors. Although PS cannot be implemented in practice, it provides an ideal allocation strategy that can be used as a reference for our tardiness analysis.

Note that our definition of PS is different from the PS schedule defined in [9]. In our defined PS, the processing capacity obtained by τ_l is strictly greater than the processing capacity it needed to end its execution on time in the average case (i.e. Eqs. (4) and (5) are satisfied), but some jobs of τ_l may not complete execution by the release time of the successive jobs. This is because jobs' execution times and periods (the minimum length of the time interval between the release times of two consecutive jobs) are stochastic in our model. When jobs of τ_l execute in PS at a constant rate $\hat{u}_l$, they may not complete execution by the release time of their successive jobs.

Example 2. Suppose we have a task system where three stochastic tasks are scheduled on two identical processors, which have the following specifications: $(\bar{p}_1, \bar{e}_1) = (1, 0.5)$, $(\bar{p}_2, \bar{e}_2) = (2.3, 1.1), (\bar{p}_3, \bar{e}_3) = (1.6, 1.3)$. One feasible choice for $\{\hat{u}_1, \hat{u}_2, \hat{u}_3\}$ is $\{0.5, 0.5, 1\}$.

2.3 Analysis Overview: Combining TPS-Induced and Stochastic-Induced Tardiness

We derive tardiness bounds for tasks with stochastic periods and execution times under TPS in this paper. For the deterministic sporadic task model [9], jobs complete with no tardiness under PS. Thus, tardiness under the deterministic sporadic task model is merely due to the use of non-optimal schedulers (such as GEDF).

Due to the stochastic nature of jobs' execution times and arrival pattern, there may be stochastic-induced tardiness under the PS schedule. Our result generalizes [9] for the stochastic case. Fig. 2 intuitively illustrates the analysis flow. In the TPS schedule, $\tau_{l,j}$ may be preempted by jobs with higher priorities, and occupies an entire processor when it executes. Suppose $\tau_{l,j}$ completes execution at $f^*_{l,j}$ in TPS. In the PS schedule, $\tau_{l,j}$ will not be preempted by other jobs and executes at rate $\hat{u}_l$. Suppose $\tau_{l,j}$ completes execution at $\hat{f}_{l,j}$ in PS. From this example, we can see a job's tardiness under TPS has two contributors: (i) stochastic-induced tardiness of $(\hat{f}_{l,j} - r_{l,j+1})$, which is due to the stochastic periods and execution times,[2] and (ii) scheduler-induced tardiness of $(f^*_{l,j} - \hat{f}_{l,j})$, due to using a specific non-optimal schedule TPS.

We first derive TPS-induced tardiness and stochastic-induced tardiness individually in Sections 3 and 4, respectively. Then in Section 5, we combine these two parts to obtain the final tardiness bound for any task τ_l scheduled under a TPS scheduler.

3 Bounding TPS-Induced Tardiness

In this section, we bound TPS-induced tardiness by comparing the job's completion time in TPS and its completion time in PS.

Definition 8 (Allocation). *Let $A(\tau_{l,j}, t_1, t_2, S)$ represent the processing capacity allocated to $\tau_{l,j}$ during $[t_1, t_2)$ in S, where S can be an arbitrary schedule. $A(\tau_l, t_1, t_2, S)$ represents the total processing capacity allocated to all jobs of τ_l during $[t_1, t_2)$ in S. Since every job executes in TPS at rate 1, $A(\tau_l, t_1, t_2, TPS)$ denotes the amount of processing time that τ_l's jobs receive during $[t_1, t_2)$ in TPS; $A(\tau_l, t_1, t_2, PS)$ denotes $\hat{u}_l$ multiplying the length of execution of τ_l's jobs in PS (as the jobs of τ_l execute at rate $\hat{u}_l$ in PS).*

Similar to the reasoning in [9], [21] (despite different definitions of PS and TPS in this paper), we define the difference between the allocation to job $\tau_{l,j}$ in PS schedule and the allocation in TPS schedule using lag reasoning that is defined as follows:

$$lag(\tau_{l,j}, t, TPS) = A(\tau_{l,j}, 0, t, PS) - A(\tau_{l,j}, 0, t, TPS). \tag{6}$$

Lag represents the difference of processing time allocated to $\tau_{l,j}$ in TPS and PS before t. τ_l's lag is defined as:

$$lag(\tau_l, t, TPS) = \sum_{j \geq 1} A(\tau_{l,j}, 0, t, PS) - \sum_{j \geq 1} A(\tau_{l,j}, 0, t, TPS). \tag{7}$$

Similarly, we define the lag for a job set Ω at time instant t in TPS as LAG

$$\begin{aligned} LAG(\Omega, t, TPS) &= \sum_{\tau_{l,j} \in \Omega} lag(\tau_{l,j}, t, TPS) \\ &= \sum_{\tau_{l,j} \in \Omega} (A(\tau_{l,j}, 0, t, PS) - A(\tau_{l,j}, 0, t, TPS)). \end{aligned} \tag{8}$$

Since $LAG(\Omega, 0, TPS) = 0$, the Eq. (9) holds when $t' \leq t$.

$$\begin{aligned} LAG(\Omega, t, TPS) = LAG(\Omega, t', TPS) + A(\Omega, t', t, PS) \\ - A(\Omega, t', t, TPS). \end{aligned} \tag{9}$$

Lag indicates the difference of the processor capacity allowed to the same jobs between PS and TPS. The key fact behind the lag reasoning is that if a job's tardiness in PS and its lag are both bounded then its total tardiness is also bounded.

2. Note that jobs incur zero tardiness in the PS schedule for ordinary sporadic task systems with deterministic periods and execution times.

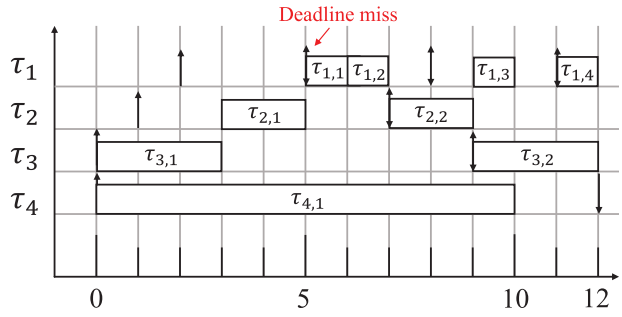


Fig. 3. A schedule for τ in Example 3.

Definition 9 (busy/non-busy intervals). *For a job set Ω, if all processors perform jobs from Ω at each time instant during a time interval $[t_1, t_2)$, we define that $[t_1, t_2)$ is busy for Ω. Note that Ω may contain jobs released by multiple tasks. $[t_1, t_2)$ is non-busy for Ω if there is at least one processor that does not execute jobs from Ω at each instant in $[t_1, t_2)$.*

Lemma 1. *For a job set Ω, if the time interval $[t_1, t_2)$ is busy, $LAG(\Omega, t_2, TPS) \leq LAG(\Omega, t_1, TSP)$.*

Proof.

$$\begin{aligned} &LAG(\Omega, t_2, TPS) \\ &= LAG(\Omega, t_1, TPS) + A(\Omega, t_1, t_2, PS) - A(\Omega, t_1, t_2, TPS) \\ &= LAG(\Omega, t_1, TPS) + A(\Omega, t_1, t_2, PS) - m \times (t_2 - t_1)) \\ &\leq LAG(\Omega, t_1, TPS). \end{aligned}$$

□

Example 3. Consider that a task set $\tau = \{\tau_1 = (1, 3), \tau_2 = (2, 6), \tau_3 = (3, 9), \tau_4 = (10, 12)\}$ is scheduled upon a two-processor platform. τ_1, τ_2, τ_3 and τ_4 release the first jobs at time instances 2, 1, 0 and 0 respectively. Assume that tasks are scheduled under FIFO, i.e., the job prioritization function is $\Upsilon_{l,j}(t) = r_{l,j}$. As shown in Fig. 3, under FIFO, $\tau_{1,1}$ misses its deadline at time instance 5 by one time unit because it cannot preempt $\tau_{2,1}$ or $\tau_{4,1}$ under FIFO, which have earlier release times.

Let $\Omega = \{\tau_{1,1}, \tau_{1,2}, \tau_{1,3}, \tau_{2,1}, \tau_{3,1}, \tau_{4,1}, \}$ be the set of jobs with absolute deadlines no greater than 12. The time interval $[3, 6)$ in Fig. 3 is a busy interval for Ω. By Eq. (9), $LAG(\Omega, 6, TPS) = LAG(\Omega, 3, TPS) + A(\Omega, 3, 6, PS) - A(\Omega, 3, 6, TPS)$. The allocation of Ω in the PS schedule during $[3, 6)$ is $A(\Omega, 3, 6, PS) = \frac{3}{3} + \frac{6}{6} + \frac{9}{9} + \frac{30}{12} = 5.5$. The allocation of Ω in TPS throughout [3, 6) is 6. Thus, $LAG(\Omega, 6, TPS) - LAG(\Omega, 3, TPS) = 5.5 - 6 = -0.5$.

Let $\Omega = \{\tau_{1,1}\}$ be the set of jobs with deadlines no greater than 5. Since the jobs $\tau_{2,1}, \tau_{3,1}, \tau_{4,1}$, which have deadlines after time 5, execute within the interval $[0, 5)$ in Fig. 3, $[0, 5)$ is non-busy for Ω in TPS. By Eq. (9), $LAG(\Omega, 5, TPS) = LAG(\Omega, 0, TPS) + A(\Omega, 0, 5, PS) - A(\Omega, 0, 5, TPS)$. The allocation of Ω in the PS schedule throughout the interval $[0, 5)$ is $A(\Omega, 0, 5, PS) = 3 \times \frac{1}{3}$. The allocation of Ω in TPS is $A(\Omega, 0, 5, TPS) = 0$. Thus, $LAG(\Omega, 5, TPS) = 1$. Fig. 3 shows that at time 4, $\tau_{1,1}$ is pending. This job has 1 unit execution cost, which is equal to the amount of pending work given by $LAG(\Omega, 5, TPS)$.

Lemma 1 indicates a very important observation, that is for a specific job set, if its LAG increases throughout a time interval, then this time interval is non-busy. Such non-busy intervals make jobs tardy. Based on this observation, we use lags to upper bound TPS-induced tardiness. In the rest of the paper, τ denotes a concrete task system, which satisfies Eqs. (4) and (5). τ has a PS schedule, with fixed $\{\hat{u}_l\}$ $(1 \leq l \leq n)$. $r_{l,j}$, $e_{l,j}$ and $f_{l,j}$ for all $\tau_{l,j} \in \tau$ are given, because this is a posteriori analysis:

- $r_{l,j}$ (the release time of $\tau_{l,j}$).
- $e_{l,j}$ (the actual execution time of $\tau_{l,j}$).
- $f_{l,j}$ (the completion time of $\tau_{l,j}$ in PS).

Definition 10. *$\hat{f}_{l,j} = \max\{f_{l,j}, r_{l,j+1}\}$ represents the earliest time instant, which is not earlier than $r_{l,j+1}$, by which $\tau_{l,j}$ has completed execution in PS.*

According to the definition of $f_{l,j}$, *job $\tau_{l,j}$'s tardiness* in PS schedule is $\hat{f}_{l,j} - r_{l,j+1}$.

Definition 11. *Let $f^*_{l,j}$ denote $\tau_{l,j}$'s completion time in TPS.* TPS-induced tardiness *equals $\{f^*_{l,j} - \hat{f}_{l,j}, 0\}$, as illustrated earlier in Fig. 2*

Definition 12. *$\hat{e}_{l,j} = \max_{1 \leq j' \leq j}\{e_{l,j'}\}$ denotes the maximum execution time of jobs released by τ_l at or before $r_{l,j}$.*

Definition 13. *Let $\rho = \max_{\tau_i \in \tau}\phi_i + \max_{\tau_l \in \tau}\varphi_l$. According to Definition 2, ρ represents the maximum range of a job's tunable priority window.*

We first assume that the following property for TPS schedule exists. Then we calculate the corresponding x to validate its existence.

(P) The TPS-induced tardiness of every job $\tau_{i,k}$ satisfying $\hat{f}_{i,k} < \hat{f}_{l,j}$ is at most $x + \hat{e}_{i,k}$, where $x \geq \rho$.

Upper bounding the TPS-induced tardiness of $\tau_{l,j}$ equals to determining the smallest $x \geq \rho$ such that property **(P)** holds, which implies a TPS-induced tardiness of at most $x + \hat{e}_{i,j}$ for all jobs of every task $\tau_i \in \tau$ by induction.

A simple case is that $\tau_{l,j}$ completes by $\hat{f}_{l,j}$, so we assume otherwise. $\tau_{l,j}$'s completion time depends on the jobs that can compete with $\tau_{l,j}$ after $\hat{f}_{l,j}$. Similar to the analysis framework used in [9], [21], a minimum value for x can be calculated through the following three steps.

1) Calculate an upper bound on pending workloads of tasks in τ (including $\tau_{l,j}$) that have priorities no lower than $\tau_{l,j}$ after $\hat{f}_{l,j}$. (Section 3.1)
2) Determine the amount of such workloads necessary for $\tau_{l,j}$'s tardiness to exceed $x + \hat{e}_{l,j}$. (Section 3.2)
3) Identify the smallest $x \geq \rho$ such that property **(P)** holds by requiring that the upper bound in Step 1 is no larger than the necessary condition in Step 2. (Section 3.3)

Determine how other jobs delay $\tau_{l,j}$'s execution is the key in reasoning about $\tau_{l,j}$'s tardiness. We classify such higher priority jobs according to the relation between their and $\tau_{l,j}$'s prioritization functions and $\hat{f}$, as following four categories:

- $\hat{f}L = \{\tau_{i,k} :: \exists t : \Upsilon_{i,k}(t) > \Upsilon_{l,j}(t) \wedge (\hat{f}_{i,k} \leq \hat{f}_{l,j})\}$
- $\hat{F}L = \{\tau_{i,k} :: \exists t : \Upsilon_{i,k}(t) > \Upsilon_{l,j}(t) \wedge (\hat{f}_{i,k} > \hat{f}_{l,j})\}$
- $\hat{f}H = \{\tau_{i,k} :: \exists t : \Upsilon_{i,k}(t) \leq \Upsilon_{l,j}(t) \wedge (\hat{f}_{i,k} \leq \hat{f}_{l,j})\}$
- $\hat{F}H = \{\tau_{i,k} :: \exists t : \Upsilon_{i,k}(t) \leq \Upsilon_{l,j}(t) \wedge i \neq l \wedge (\hat{f}_{i,k} > \hat{f}_{l,j})\}$

In this classification, $\hat{f}$ and $\hat{F}$ indicate that the completion times of the corresponding jobs in PS are no later and later than $\hat{f}_{l,j}$, respectively. H represents that the priority of $\tau_{i,k}$ is at least $\tau_{l,j}$'s priority, and L denotes that the priority of $\tau_{i,k}$ is lower than $\tau_{l,j}$'s priority. It is evident that $\tau_{l,j} \in \hat{f}H$.

Let $\Omega = \hat{f}H \cup \hat{f}L$. In PS schedule, the completion times of jobs in Ω are no later than $\hat{f}_{l,j}$. Jobs in Ω are not performed beyond $\hat{f}_{l,j}$ in the PS schedule. Because the jobs in $\hat{f}H \cup \hat{F}H$ have priorities no lower than $\tau_{l,j}$, $\tau_{l,j}$'s execution may be delayed (in the worst case scenario) until the number of ready jobs in $\hat{f}H \cup \hat{F}H$ including $\tau_{l,j}$ is at most m.

3.1 Step1: Determining an Upper Bound on Competing Work

As we discussed earlier, a minimum value for x can be calculated through three steps. In this section, we perform the first step and calculate an upper bound, denoted by $W(\hat{f}H \cup \hat{F}H, \hat{f}_{l,j}, TPS)$, on competing workload for $\tau_{l,j}$.

According to the definitions of task subsets, priorities of jobs in $\hat{f}H \cup \hat{F}H$ are no lower than those of $\tau_{l,j}$. Thus, the competing workloads from $\hat{f}H \cup \hat{F}H$ for $\tau_{l,j}$ beyond $\hat{f}_{l,j}$, $W(\hat{f}H \cup \hat{F}H, \hat{f}_{l,j}, TPS)$, can be upper bounded by the sum of (i) the amount of workloads pending at $\hat{f}_{l,j}$ for jobs in $\hat{f}H$, and (ii) the amount of workloads $W(\hat{F}H, \hat{f}_{l,j}, TPS)$ required by jobs in $\hat{F}H$ which can postpone $\tau_{l,j}$'s execution after $\hat{f}_{l,j}$. For the workloads described in (i), since the completion times of jobs from Ω in PS are no later than $\hat{f}_{l,j}$, they are not performed in the PS schedule after $\hat{f}_{l,j}$. Thus, $LAG(\hat{f}H, \hat{f}_{l,j}, TPS)$, which denotes the workload pending from jobs in $\hat{f}H$, must be positive so that $\tau_{l,j}$ can execute beyond its completion time in PS at $\hat{f}_{l,j}$.

Instead of bounding $LAG(\hat{f}H, \hat{f}_{l,j}, TPS)$, We bound $LAG(\Omega, \hat{f}_{l,j}, TPS)$ because $LAG(\hat{f}H, \hat{f}_{l,j}, TPS) \leq LAG(\Omega, \hat{f}_{l,j}, TPS)$. This inequality holds because $\Omega = \hat{f}H \cup \hat{f}L$, and $LAG(\hat{f}L, \hat{f}_{l,j}, TPS)$ is non-negative since according to the definition of $\hat{f}L$, the jobs in $\hat{f}L$ cannot execute more workload by the time instant $\hat{f}_{l,j}$ in TPS than that they have executed in PS. Thus, we have $W(\hat{f}H \cup \hat{F}H, \hat{f}_{l,j}, TPS) \leq LAG(\Omega, \hat{f}_{l,j}, TPS) + W(\hat{F}H, \hat{f}_{l,j}, TPS)$. Therefore, $W(\hat{f}H \cup \hat{F}H, \hat{f}_{l,j}, TPS)$ can be obtained by determining upper bounds for $LAG(\Omega, \hat{f}_{l,j}, TPS)$ and $W(\hat{F}H, \hat{f}_{l,j}, TPS)$.

Upper bound on $LAG(\Omega, \hat{f}_{l,j}, TPS)$. Becasue we are deriving the upper bound on $LAG(\Omega, \hat{f}_{l,j}, TPS)$, all busy and non-busy intervals discussed herein are w.r.t. Ω and TPS represents any scheduler, which can be realized by TPS, unless stated otherwise.

According to Lemma 1, we know that if there is no non-busy interval for Ω existing in $[0, \hat{f}_{l,j})$, then $LAG(\Omega, \hat{f}_{l,j}, TPS) \leq LAG(\Omega, 0, TPS) = 0$. We thus discuss the more complicated case where some non-busy intervals exist in $[0, \hat{f}_{l,j})$. There are two reasons for an interval to be non-busy for jobs in Ω:

- The number of ready jobs in Ω is less than the number of available processors. In this case, it does not matter whether jobs from $\hat{F}H$ or $\hat{F}L$ are performed in this interval. Such intervals are called non-busy non-occupation.
- Some ready jobs in Ω are not performed within some sub-intervals in $[0, \hat{f}_{l,j})$, since one or more processors are occupied by jobs in $\hat{F}H$, which have higher priorities. Therefore, such intervals are called non-busy occupation.

Definition 14. *$\tau_{c,h}$ is a* carry-in *job from task τ_c if $r_{c,h} \leq \hat{f}_{l,j} < \hat{f}_{c,h}$ holds.*

For each task τ_k, at most one such job could exist. The LAG for Ω can be increased by such carry-in jobs, because they can postpone the execution of jobs in Ω before time instance $\hat{f}_{l,j}$.

Definition 15. *Let τ_H represent a task set. τ_H includes all the tasks having carry-in jobs in $\hat{F}H$.*

Definition 16. *Suppose $\tau_{c,h}$ is a carry-in job. δ_c represents the amount of workload performed by $\tau_{c,h}$ in TPS by the time instant $\hat{f}_{l,j}$.*

In the following analysis, we consider a time instant t_2: before the time instant $\hat{f}_{l,j}$, if some non-busy non-occupation intervals exist and throughout this time interval LAG for Ω increases, then $[t_1, t_2]$ denotes the latest such interval; otherwise, $t_1 = t_2 = 0$. Note that by Lemma 1, for Ω LAG increase only across non-busy intervals.

Proofs for Lemmas 2 and 3 have also been provided previously in [9], [18], [21] for ordinary sporadic task systems with deterministic execution times and periods. Intuitively, when we derive TPS-induced tardiness, $LAG(\Omega, \hat{f}_{l,j}, TPS) + W(\hat{F}H, \hat{f}_{l,j}, TPS)$'s value only depends on allocations to jobs in $\Omega \cup \hat{F}H$ in PS and allocations to the same jobs in TPS, which can compete processing time with $\tau_{l,j}$.

Thus, the derivation of the TPS-induced tardiness is not affected by the task's stochastic properties in the PS schedule. Also, Property (**P**) alone is sufficient to determine the amount of workload in $\Omega \cup \hat{F}H$ which competes processing capacity with $\tau_{l,j}$.

Based on these intuitions, Lemmas 2 and 3 still hold under the stochastic task model.

Lemma 2. $LAG(\Omega, \hat{f}_{l,j}, TPS) \leq LAG(\Omega, t_2, TPS) + \sum_{\tau_c \in \tau_H} \delta_c(1 - \hat{u}_c)$.

Proof. By Eq. (8),

$$LAG(\Omega, \hat{f}_{l,j}, TPS) \leq LAG(\Omega, t_2, TPS) + A(\Omega, t_2, \hat{f}_{l,j}, PS) - A(\Omega, t_2, \hat{f}_{l,j}, TPS). \tag{10}$$

We split $[t_2, \hat{f}_{l,j}]$ into k non-overlaping consecutive intervals $[t_{v_x}, t_{w_y})$, where $1 \leq i \leq k$. We have $t_2 = t_{v_1}$, $t_{w_{(i-1)}} = t_{v_i}$ and $t_{w_k} = \hat{f}_{l,j}$. After splitting, each such interval $[t_{v_i}, t_{w_i})$ belongs to one of the three categories: (i) busy, (ii) non-busy occupation, or (iii) non-busy non-occupation. Suppose $[t_{v_i}, t_{w_i})$ is an occupation interval, if any task $\tau_c \in \tau_H$ is performed at some time instants in the interval, then it must execute continuously across this interval. Note that τ_c executing continuously throughout $[t_2, \hat{f}_{l,j})$ is not necessary. In order to calculate LAG, let $\alpha_c \subseteq \tau_H$ denote a set of tasks that are performed continuously across $[t_{v_i}, t_{w_i})$ for any occupation interval $[t_{v_i}, t_{w_i})$. Across the interval $[t_2, \hat{f}_{l,j})$, for tasks in Ω, the total allocation difference can be denoted as $A(\Omega, t_2, \hat{f}_{l,j}, PS) - A(\Omega, t_2, \hat{f}_{l,j}, TPS) = \sum_{i=1}^{b}(A(\Omega, t_{v_i}, t_{w_i}, PS) - A(\Omega, t_{v_i}, t_{w_i}, TPS))$. Then, the

workload performed in PS minus the workload performed in the *TPS* throughout every such intervals $[t_{v_i}, t_{w_i})$ can be upper bounded. We then upper bound the total allocation difference throughout $[t_2, \hat{f}_{l,j})$ by summing all these bounds together. According to the above discussed three categories, there are three possible cases.

Case 1. $[t_{v_i}, t_{w_i})$ is busy. $A(\Omega, t_{v_i}, t_{w_i}, TPS) = m(t_{w_i} - t_{v_i})$, since all processors are busy and used by jobs in Ω in *TPS*. And $A(\Omega, t_{v_i}, t_{w_i}, PS) \leq U_{sum}(t_{w_i} - t_{v_i})$. Because $U_{sum} \leq m$, then

$$A(\Omega, t_{v_i}, t_{w_i}, PS) - A(\Omega, t_{v_i}, t_{w_i}, TPS) \leq 0. \tag{11}$$

Case 2. $[t_{v_i}, t_{w_i})$ is non-busy non-occupation. According to the definition of LAG, LAG is not increased for tasks in Ω across $[t_{v_i}, t_{w_i})$ by the selection of $[t_1, t_2]$. Therefore, from Eq. (9), we have

$$A(\Omega, t_{v_i}, t_{w_i}, PS) - A(\Omega, t_{v_i}, t_{w_i}, TPS) \leq 0. \tag{12}$$

Case 3. $[t_{v_i}, t_{w_i})$ is non-busy occupation. Let $\sum_{\tau_c \in \alpha_c} \hat{u}_c$ denote the total utilization of tasks $\tau_c \in \alpha_c$. According to the definition of Ω, these tasks' carry-in jobs are not in Ω. Thus, the total allocation to jobs in Ω during $[t_{v_i}, t_{w_i})$ in PS is no more than $A(\Omega, t_{v_i}, t_{w_i}, PS) = (t_{w_i} - t_{v_i})(m - \sum_{\tau_c \in \alpha_c} \hat{u}_c)$. Every processor is busy at each time instant during the interval $[t_{v_i}, t_{w_i})$, since the interval is occupation. Thus, $A(\Omega, t_{v_i}, t_{w_i}, TPS) = (t_{w_i} - t_{v_i})(m - |\alpha_c|)$ holds. Put all pieces together, the allocation difference for jobs in Ω across the interval is

$$\begin{aligned} &A(\Omega, t_{v_i}, t_{w_i}, PS) - A(\Omega, t_{v_i}, t_{w_i}, TPS) \\ &\leq (t_{w_i} - t_{v_i})\left(\left(m - \sum_{\tau_c \in \alpha_c} \hat{u}_c\right) - (m - |\alpha_c|)\right) \\ &= (t_{w_i} - t_{v_i}) \sum_{\tau_c \in \alpha_c} (1 - \hat{u}_c). \end{aligned} \tag{13}$$

To complete the proof, for any busy interval $[t_{v_i}, t_{w_i})$ or non-busy non-occupation interval $[t_{v_i}, t_{w_i})$, define $\alpha_c = null$. The total allocation differences between PS and TPS for all intervals $[t_{v_i}, t_{w_i})$ which are provided by Eqs. (11), (12), and (13), then

$$\begin{aligned} &A(\Omega, t_{v_i}, t_{w_i}, PS) - A(\Omega, t_{v_i}, t_{w_i}, TPS) \\ &\leq \sum_{i=1}^{b} \sum_{\tau_c \in \alpha_c} (t_{w_i} - t_{v_i})(1 - \hat{u}_c)). \end{aligned} \tag{14}$$

For each task $\tau_c \in \tau_H$, the amount of workload executed by τ_c's carry-in job is no more than δ_c during all sub intervals. Thus, $A(\Omega, t_{v_i}, t_{w_i}, PS) - A(\Omega, t_{v_i}, t_{w_i}, TPS) \leq \sum_{\tau_c \in \tau_H} \delta_c(1 - \hat{u}_c)$. Setting it into Eq. (10), we obtained that $LAG(\Omega, \hat{f}_{i,j}, TPS) \leq LAG(\Omega, t_2, TPS) + \sum_{\tau_c \in \tau_H} \delta_c(1 - \hat{u}_c)$. □

Lemma 3. *Let e_p^t denote the maximum job released by τ_p before t, then $lag(\tau_p, t, TPS) \leq x \times \hat{u}_p + e_p^t$ for any task τ_p and $t \in [0, \hat{f}_{l,j}]$.*

Proof. $\hat{f}_{p,j}$ represents the completion time of $\tau_{p,j}$, which is τ_p's earliest pending job pends at time instant t in *TPS*. γ_p denotes the amount of workload $\tau_{p,j}$ has executed by t. First we consider the case where $\hat{f}_{p,j} < t$. By Eq. (7),

$$\begin{aligned} lag(\tau_p, t, TPS) &= \sum_{q \geq j} lag(\tau_{p,q}, t, TPS) \\ &= \sum_{q \geq j} (A(\tau_{p,q}, 0, t, PS) - A(\tau_{p,q}, 0, t, TPS)). \end{aligned} \tag{15}$$

$A(\tau_{p,q}, 0, t, TPS) = A(\tau_{p,q}, r_{p,q}, t, TPS)$, because $\tau_{p,q}$ does not execute if it is not released. Thus,

$$\begin{aligned} &lag(\tau_p, t, TPS) \\ &= \sum_{q > j} (A(\tau_{p,q}, r_{p,q}, t, PS) - A(\tau_{p,q}, r_{p,q}, t, TPS)) \\ &\quad + A(\tau_{p,j}, r_{p,j}, t, PS) - A(\tau_{p,j}, r_{p,j}, t, TPS). \end{aligned} \tag{16}$$

According to PS's definition, $\sum_{q>j} A(\tau_{p,q}, r_{p,q}, t, PS) \leq \hat{u}_p(t - \hat{f}_{p,j})$. It is evident that $A(\tau_{p,j}, r_{p,j}, t, PS) = e_{p,j}$, $A(\tau_{p,j}, r_{p,j}, t, TPS) = \gamma_p$ and $\sum_{q>j} A(\tau_{p,q}, r_{p,q}, t, TPS) = 0$ based on the selection of $\tau_{p,q}$. Putting these results into Eq. (11), then

$$lag(\tau_p, t, TPS) \leq \hat{u}_p(t - \hat{f}_{p,j}) + e_{p,j} - \gamma_p. \tag{17}$$

According to Property (**P**), $\tau_{p,j}$'s tardiness is no greater than $x + \hat{e}_{p,j}$, thus $t + e_{p,j} - \gamma_p \leq \hat{f}_{p,j} + x + \hat{e}_{p,j}$. Therefore, $t - \hat{f}_{p,j} \leq x + \hat{e}_{p,j} + \gamma_p - e_{p,j}$. From Eq. (17), then

$$\begin{aligned} lag(\tau_p, t, TPS) &\leq \hat{u}_p(t - \hat{f}_{p,j}) + e_{p,j} - \gamma_p \\ &= \hat{u}_p(x + \hat{e}_{p,j} + \gamma_p - e_{p,j}) + e_{p,j} - \gamma_p \\ &\leq x \times \hat{u}_p + \hat{e}_{p,j} \leq x \times \hat{u}_p + e_p^t. \end{aligned}$$

Then, we consider the case $\hat{f}_{p,j} \geq t$. Based on Eq. (7) and the definition of $\tau_{p,j}$,

$$\begin{aligned} &lag(\tau_p, t, TPS) = \sum_{q \leq j} lag(\tau_{p,q}, t, TPS) \\ &= \sum_{q \leq j} (A(\tau_{p,q}, 0, t, PS) - A(\tau_{p,q}, 0, t, TPS)) \\ &= \sum_{q < j} (A(\tau_{p,q}, r_{p,q}, t, PS) - A(\tau_{p,q}, r_{p,q}, t, TPS)) \\ &\quad + A(\tau_{p,j}, r_{p,j}, t, PS) - A(\tau_{p,j}, r_{p,j}, t, TPS). \end{aligned} \tag{18}$$

According to PS's definition and $\hat{f}_{p,j} \geq t$, $\sum_{h<j} A(\tau_{p,h}, r_{p,h}, t, PS) \leq \sum_{h<j} e_{p,h}$; because in the schedule *TPS*, $\tau_{p,j}$ is the earliest pending job of τ_p at time t, $\sum_{h<j} A(\tau_{p,h}, r_{p,h}, t, TPS) = \sum_{h<j} e_{p,h}$. Also, $A(\tau_{p,j}, r_{p,j}, t, PS) \leq e_{p,j}$ and $A(\tau_{p,j}, r_{p,j}, t, TPS) = \gamma_p \geq 0$, and setting these values into Eq. (18)

$$\begin{aligned} &lag(\tau_p, t, TPS) \\ &\leq \left(\sum_{q<j} e_{p,q} - \sum_{q<j} e_{p,q}\right) + e_{p,j} - \gamma_p \\ &\leq e_{p,j} \leq e_p^t. \end{aligned} \tag{19}$$

Therefore, $lag(\tau_p, t, S) \leq x \times \hat{u}_p + e_p^t$ holds for any task τ_p and $t \in [0, \hat{f}_{l,j}]$. Thus, the lemma is proved. □

Before we prove the upper bound on LAG of jobs in Ω at time t_2, we introduce two definitions first. Let $k' =$

$\max\{k : \tau_{i,k} \in \Omega, r_{i,k} \le \hat{f}_{l,j}\}$. Define

$$U_L = \sum_{i=1}^{m-1} \hat{u}^i, \tag{20}$$

where $\hat{u}^i$ is the ith largest value of $\{\hat{u}_i\}$ and define

$$\eta_{l,j} = \sum_{i=1}^{m-1} \hat{e}^i, \tag{21}$$

where $\hat{e}^i$ is the ith largest value of $\{\hat{e}_{i,k'}\}$.

Lemma 4. $LAG(\Omega, t_2, S) \le x \times U_L + \eta_{l,j}$.

Proof. Based on the definition of LAG, we sum individual task lags at t_2 to upper bound $LAG(\Omega, t_2, TPS)$. The lemma holds trivially if $t_2 = 0$ and $LAG(\Omega, t_2, TPS) = 0$. We discuss the case where $t_2 > 0$. Since t_2 is non-busy non-occupation, $|\chi| \le (m-1)$. If no jobs from a task are pending at t_2, then $lag(\tau_i, t_2, S) = 0$. Thus, according to Eq. (7) and Lemma 3, the following equation holds:

$$\begin{aligned} &LAG(\Omega, t_2, S) \\ &= \sum_{\tau_i : \tau_{i,j} \in \Omega} lag(\tau_i, t_2, TPS) \le \sum_{\tau_i \in \chi} lag(\tau_i, t_2, TPS) \\ &\le \sum_{\tau_i \in \chi} \hat{u}_i \times x + e_i^{t_2} \le \eta_{l,j} + x \times U_L. \end{aligned} \tag{22}$$

□

Thus, $LAG(\Omega, \hat{f}_{i,j}, S)$ is upper bounded by $x \times U_L + \eta_{l,j} + \sum_{\tau_c \in \tau_H} \delta_c(1 - \hat{u}_c)$ according to Lemmas 2 and 4.

Upper Bound on $W(\hat{F}H, \hat{f}_{i,j}, TPS)$. To calculate a bound on $W(\hat{F}H, \hat{f}_{i,j}, TPS)$, which denotes the total workload of jobs that can delay $\tau_{l,j}$ after time instant $\hat{f}_{l,j}$, first we consider such a job with the latest possible release time. If a job's release time is far behind $\hat{f}_{i,j}$, it may not be able to compete with $\tau_{l,j}$ since its priority cannot be tuned higher than $\tau_{l,j}$'s priority because the range of the priority window is limited.

Lemma 5. *If* $\tau_{i,k} \in \hat{F}H \cup \hat{f}H$, *then* $r_{i,k} \le \hat{f}_{l,j} + \varphi_l + \phi_i$.

Proof. Given that $r_{l,j} - \phi_l \le \Upsilon_{l,j}(t) \le r_{l,j+1} + \varphi_l$ holds for any job $\tau_{l,j}$ and $r_{l,j+1} \le \hat{f}_{l,j}$ (by Definition 11), if $\tau_{i,k} \in \hat{F}H \cup \hat{f}H$, then at time t, $r_{i,k} - \phi_i \le \Upsilon_{i,k}(t) \le \Upsilon_{l,j}(t) \le \hat{f}_{l,j} + \varphi_l$ holds. This implies $r_{i,k} \le \hat{f}_{l,j} + \varphi_l + \phi_i$. □

Corollary 1. *All jobs in* $\hat{F}H \cup \hat{f}H$ *are released no later than* $\hat{f}_{l,j} + \rho$ *according to the definitions of those job sets, where* $\rho = \max_{\tau_h \in \tau}(\phi_h) + \max_{\tau_h \in \tau}(\varphi_h)$.

Lemma 6. *Let* $\tilde{p}_i$ *denote the minimum release time interval and* $\tilde{e}_i$ *denote the maximum job execution time of* τ_i *before* $\hat{f}_{l,j} + \varphi_l + \phi_i$. $W(\hat{F}H, \hat{f}_{l,j}, TPS) \le \sum_{\tau_i \in \tau_H}(e_{i,k} - \delta_i) + \sum_{\tau_i \in \tau \setminus \tau_l}(\lceil \frac{\varphi_l + \phi_i}{\tilde{p}_i} \rceil)\tilde{e}_i$ *holds.*

Proof. $\hat{F}H$ includes two types of jobs: a carry-in job or a job, the release time of which is later than $\hat{f}_{l,j}$. Suppose $\tau_{i,k}$ in $\hat{F}H$ belongs to the latter case. According to Lemma 5, the release time of $\tau_{i,k}$ is in the interval $(\hat{f}_{l,j}, \hat{f}_{l,j} + \varphi_l + \phi_i]$. Therefore, every task τ_i possibly has a carry-in job in $\hat{F}H$ and at most $\lceil \frac{\varphi_l + \phi_i}{\tilde{p}_i} \rceil$ jobs in $\hat{F}H$ with release times later than $\hat{f}_{l,j}$. If $\tau_{i,k}$ in $\hat{F}H$ belongs to the first case, τ_i has a carry-in job. According to the definition of τ_H, τ_i is in τ_H, and the workload generated by $\tau_{i,k}$ after $\hat{f}_{l,j}$ is no greater than $e_{i,k} - \delta_i$. It is evident that the workload produced by any job of τ_i in $\hat{F}H$ with release time later than $\hat{f}_{l,j}$ is no greater than $\tilde{e}_i$. From these facts, the lemma follows. □

Upper Bound on $W(\hat{f}H \cup \hat{F}H, \hat{f}_{l,j}, TPS)$. Since $W(\hat{f}H \cup \hat{F}H, \hat{f}_{l,j}, TPS) \le LAG(\Omega, \hat{f}_{l,j}, TPS) + W(\hat{f}H, \hat{f}_{l,j}, TPS)$, by Lemmas 2, 4, and 6, we have

$$\begin{aligned} &W(\hat{f}H \cup \hat{F}H, \hat{f}_{l,j}, TPS) \\ &\le x \times U_L + \eta_{l,j} + \sum_{\tau_c \in \tau_H}(\delta_c(1 - \hat{u}_c) + (e_{c,k} - \delta_{c,k})) \\ &\quad + \sum_{\tau_i \in \tau \setminus \tau_l}\Big(\Big\lceil \frac{\varphi_l + \phi_i}{\tilde{p}_i} \Big\rceil\Big)\tilde{e}_i \\ &\le x \times U_L + \eta_{l,j} + \sum_{\tau_i \in \tau \setminus \tau_l}\Big(\Big\lceil \frac{\varphi_l + \phi_i}{\tilde{p}_i} \Big\rceil + 1\Big)\tilde{e}_i. \end{aligned} \tag{23}$$

3.2 Step 2: Determining Necessary Condition for Tardiness to Exceed $x + \hat{e}_{l,j}$

In this section, a lower bound on the amount of workload with priorities higher than $\tau_{l,j}$ is derived, which is necessary for $\tau_{l,j}$ to complete execution more than $x + \hat{e}_{l,j}$ time units after its deadline.

Lemma 7. *If* $\tau_{l,j}$*'s tardiness exceeds* $x + \hat{e}_{l,j}$, *where* $x \ge \rho$ *(*ρ *is defined in Definition 13), then* $W(\hat{f}H \cup \hat{F}H, \hat{f}_{l,j}, TPS) > \rho + m \times (x - \rho) + \hat{e}_{l,j}$.

Proof. This lemma is proved by contraposition. That is, we prove that $W(\hat{f}H \cup \hat{F}H, \hat{f}_{l,j}, S) \le \rho + m \times (x - \rho) + \hat{e}_{l,j}$ and show that $\tau_{l,j}$'s tardiness is no larger than $x + \hat{e}_{l,j}$. Since jobs in $\hat{f}L \cup \hat{F}L$ cannot preempt $\tau_{l,j}$ and thus they can not postpone the completion time of $\tau_{l,j}$, they are ignored for this proof. By Corollary 1, the release times of jobs in $\hat{f}H \cup \hat{F}H$ are no later than $\hat{f}_{l,j} + \rho$. Therefore, after $\hat{f}_{l,j} + \rho$, the number of tasks, which have pending jobs in $\hat{f}H \cup \hat{F}H$, decreases.

Consider the time instant $b_{l,j} = \max\{f^*_{l,j-1}, v_{l,j}\}$, where $v_{l,j} = \min\{t \ge \hat{f}_{l,j} : [t, \infty)$ is a non-busy interval$\}$.

Since $b_{l,j} \ge v_{l,j} \ge \hat{f}_{l,j} \ge r_{l,j+1} \ge r_{l,j}$, $\tau_{l,j}$ must have started executing in TPS at $b_{l,j}$. τ_{l,j_1} has completed (since $b_{l,j} \ge f^*_{l,j-1}$), and at least one processor is idle after $b_{l,j}$. It is evident that $\tau_{l,j}$ will complete execution by $b_{l,j} + e_{l,j}$, which means that

$$\begin{aligned} f^*_{l,j} &\le \max\{f^*_{l,j-1} + e_{l,j}, v_{i,j} + e_{l,j}\} \\ &\le \max\{\hat{f}_{l,j-1} + x + \hat{e}_{l,j-1} + e_{l,j}, v_{l,j} + e_{l,j}\}. \end{aligned} \tag{24}$$

There exist two possible cases depending on the relationship between $\hat{f}_{l,j-1} + x + \hat{e}_{l,j-1} + e_{l,j}$ and $v_{l,j} + e_{l,j}$.

Case 1. $\hat{f}_{l,j-1} + x + \hat{e}_{l,j-1} + e_{l,j} \ge v_{l,j} + e_{l,j}$. According to the definition of PS, τ_l's jobs are performed sequentially in PS schedule at a constant rate of $\hat{u}_l$, and do not start executing until their previous jobs complete execution, thus,

$$\begin{aligned} f_{l,j} &\ge f_{l,j-1} + e_{i,j}/\hat{u}_i \\ &\ge f_{l,j-1} + e_{l,j}. \end{aligned} \tag{25}$$

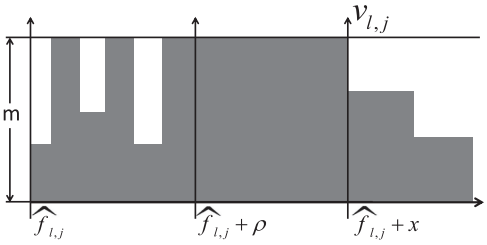


Fig. 4. The structure of workload on processers after $\hat{f}_{l,j}$. When $v_{i,j} > \hat{f}_{l,j}+x$, the earliest non-busy time instant after $\hat{f}_{l,j}+\rho$ is $v_{l,j}$.

Also, jobs cannot execute in PS before they are released, thus,

$$\begin{aligned} f_{i,j} &\geq r_{i,j} + e_{i,j}/\hat{u}_l, \\ &\geq r_{l,j} + e_{l,j}. \end{aligned} \tag{26}$$

Combine Eq. (25) with Eq. (26), we get $f_{l,j} \geq \max\{f_{l,j-1}, r_{l,j}\} + e_{l,j} = \hat{f}_{l,j-1} + e_{l,j}$. Thus, Eq. (24) becomes

$$\begin{aligned} f^*_{l,j} &\leq \max\left\{\hat{f}_{l,j-1} + x + \hat{e}_{l,j-1} + e_{l,j}, v_{l,j} + e_{l,j}\right\} \\ &= \hat{f}_{l,j-1} + x + \hat{e}_{l,j-1} + e_{l,j} \\ &\leq f_{l,j} + x + \hat{e}_{l,j-1} \\ &\leq \hat{f}_{l,j} + x + \hat{e}_{l,j}, \end{aligned} \tag{27}$$

which indicates that $\tau_{l,j}$'s tardiness is not greater than $x + \hat{e}_{l,j}$.

Case 2. $\hat{f}_{l,j-1} + x + \hat{e}_{l,j-1} + e_{l,j} < v_{l,j} + e_{l,j}$. Since $\hat{e}_{l,j-1} \leq \hat{e}_{l,j}$ and $e_{l,j} \leq \hat{e}_{l,j}$ (by Definition 12) , if $v_{l,j} \leq \hat{f}_{l,j} + x$, then according to $v_{l,j}$'s definition, $\tau_{l,j}$'s tardiness is no greater than $x + \hat{e}_{l,j}$. Thus, we consider the other case, i.e., $v_{i,j} > \hat{f}_{l,j} + x$, where $[\hat{f}_{l,j} + \rho, \hat{f}_{l,j} + x]$ is a busy interval. The reason is that according to Corollary 1, the release times of all jobs in $\hat{f}H \cup \hat{F}H$ are no later than $\hat{f}_{l,j} + \rho$. Therefor, after $\hat{f}_{l,j} + \rho$, the number of tasks, which have pending jobs in $\hat{f}H \cup \hat{F}H$, decreases. When $v_{i,j} > \hat{f}_{l,j} + x$, the earliest non-busy time instant after $\hat{f}_{l,j} + \rho$ is $v_{l,j}$, which is illustrated in Fig. 4. No job will be assigned to a processor, which becomes available at any non-busy time instant $t \geq \hat{f}_{l,j+\rho}$, because no job is released after $\hat{f}_{l,j} + \rho$. Therefore, the time interval after time instant t is non-busy. According to the selection of $v_{l,j}$, $v_{l,j}$ equals to t. The amount of workload in TPS during $[\hat{f}_{l,j} + \rho, \hat{f}_{l,j} + x]$ is no smaller than $(x - \rho) \times m$. Because $W(\hat{f}H \cup \hat{F}H, \hat{f}_{l,j}, S) \leq \rho + m \times (x - \rho) + \hat{e}_{l,j}$, the amount of workload in TPS during $[\hat{f}_{l,j}, \hat{f}_{l,j} + \rho]$ and $[\hat{f}_{l,j} + x, \hat{f}_{l,j} + x + \hat{e}_{l,j}]$ is no greater than $\rho + \hat{e}_{l,j}$. $\tau_{l,j}$ will hence complete by $\hat{f}_{l,j} + x + \hat{e}_{l,j}$ since $x \geq \rho$, even if all the workloads are performed sequentially. Thus, $f^*_{l,j} \leq \hat{f}_{l,j} + x + \hat{e}_{l,j}$ and the contraposition holds. □

3.3 Step 3: Deriving TPS-Induced Tardiness Bound

According to Lemma 7, requiring Eq. (23), which is an upper bound on $W(\hat{f}H \cup \hat{F}H, \hat{f}_{l,j}, TPS)$, to be no greater than $\rho + m \times (x - \rho) + \hat{e}_{l,j}$ will enforce that $\tau_{l,j}$'s TPS-induced tardiness is no greater than $x + \hat{e}_{l,j}$. The following inequality holds.

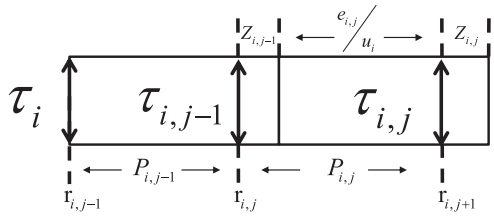


Fig. 5. Recursion for tardiness in PS. $Z_{i,j}$ denotes the tardiness of $\tau_{i,j}$ in PS and $Z_{i,j}$ can be denoted by $Z_{i,j-1}$ as a recursion: $Z_{i,j} = Z_{i,j-1} + e_{i,j}/\hat{u}_i - p_{i,j}$.

$$\begin{aligned} x \times U_L + \eta_{l,j} + \sum_{\tau_i \in \tau \setminus \tau_l} \left(\left\lceil \frac{\varphi_l + \phi_i}{\tilde{p}_i} \right\rceil + 1 \right) \tilde{e}_i \\ \leq \rho + m \times (x - \rho) + \hat{e}_{l,j}. \end{aligned} \tag{28}$$

Thus,

$$x \geq \frac{\eta_{l,j} + R(l)}{m - U_L}, \tag{29}$$

where

$$R(l) = (m-1)\rho - \hat{e}_{l,j} + \sum_{\tau_i \in \tau \setminus \tau_l} \left(\left\lceil \frac{\varphi_l + \phi_i}{\tilde{p}_i} \right\rceil + 1 \right) \tilde{e}_i. \tag{30}$$

If x equals to the right-hand side of Eq. (29), then the TPS-induced tardiness of $\tau_{l,j}$ will not be greater than $x + \hat{e}_{l,j}$.

Theorem 1. *The TPS-induced tardiness for a task $\tau_{l,j}$ in TPS schedule is at most $x + \hat{e}_{l,j}$, where x is obtained above.*

4 Bounding Stochastic-Induced Tardiness

We consider the stochastic-induced tardiness as a stochastic variable and derive a bound on its expected value in this section. Specially, under the PS schedule, we demonstrate that any job's expected tardiness can be upper bounded by a constant value that only depends on tasks' period distribution, execution time distribution, and the values of $\hat{u}_l$. Our analysis is motivated by the stochastic analysis framework first presented by Mills and Anderson [27]. We leverage and extend this framework by considering an additional stochastic task property, i.e., the stochastic task period.

4.1 The Stochastic Model

As we discussed in Section 2.2, PS can be viewed as a system consisting of n processors, and the processing capacity of the ith processor is a fraction $\hat{u}_i$ of one processor in a real system. The ith processor is dedicated to execute τ_i. Thus, jobs from different tasks do not have interference with each other in PS and we can consider each task independently.

Consider task τ_i for example. Jobs of τ_i are executed sequentially. Let $Z_{i,j}$ denote the tardiness of $\tau_{i,j}$ in PS. Thus, $Z_{i,j} = \max\{0, \hat{f}_{i,j} - r_{i,j+1}\}$ and $Z_{i,j}$ is the *stochastic-induced tardiness* for $\tau_{i,j}$. $Z_{i,j}$ can be denoted by $Z_{i,j-1}$ as a recursion, i.e. $\tau_{i,j}$'s tardiness can be calculated using three values: the tardiness of $\tau_{i,j-1}$ plus the execution time of $\tau_{i,j}$ in PS schedule (i.e. $e_{i,j}/\hat{u}_i$), and minus its period (i.e. $p_{i,j}$)

$$Z_{i,j} = \max\{0, Z_{i,j-1} + e_{i,j}/\hat{u}_i - p_{i,j}\}. \tag{31}$$

This recursion is visualized in Fig. 5, $Z_{i,j} = Z_{i,j-1} + e_{i,j}/\hat{u}_i - p_{i,j}$ and $Z_{i,0} = 0$.

Similar recursive equations in queuing theory are used to represent the customers' waiting time in a queue under the single server G/G/1 queuing model. The queueing model and Eq. (31), were first studied in 1952 by Lindley [20].

Theorem 2 ([20]). *The stochastic process $\{Z_{i,j}, j \geq 1\}$ has a limit probability distribution function $\pi_i(\cdot)$ as j increases if and only if $E(S_{i,j} - I_{i,j}) < 0$.*

According to the above Theorem, $Z_{i,j}$ has a limit probability distribution function $\pi_i(\cdot)$ as j increases if and only if

$$E(S_{i,j} - I_{i,j}) < 0, \tag{32}$$

where $E(S_{i,j} - I_{i,j})$ represents the expectation of $(S_{i,j} - I_{i,j})$, $S_{i,j}$ denotes the service time provided by the server and $I_{i,j}$ is the inter arrival time between consecutive customers. We set $S_{i,j} = e_{i,j}/\hat{u}_i$ and $I_{i,j} = r_{i,j+1} - r_{i,j}$, therefore,

$$E(S_{i,j} - I_{i,j}) = E(e_{i,j}/\hat{u}_i - p_{i,j}) = \bar{e}_i/\hat{u}_i - \bar{p}_i < 0, \tag{33}$$

where $\bar{e}_i/\hat{u}_i - \bar{p}_i$ is negative by Eq. (5). Thus, Z_i has a probability distribution function $\pi_i(\cdot)$ as j increases. According to [15] (p. 474), the expectation of Z_i can be upper bounded by Eq. (34)

$$E(Z_i) \leq \frac{\sigma'^2_i + \sigma^2_i}{2(\bar{p}_i - \bar{e}_i/\hat{u}_i)}, \tag{34}$$

where σ'^2_i and σ^2_i are the variance of the period and execution time distributions of τ_i, respectively, as defined earlier in Assumptions 1 and 2.

Lemma 2 implies that as j becomes large, $Z_{i,j}$ gets close to a limit probability distribution function $\pi_i(\cdot)$, and $Z_{i,j}$ has a finite mean by Eq. (34). Based on these results, we bound all jobs's tardiness next.

4.2 Tardiness Bound for All Jobs

Based on the expected tardiness bound for $\tau_{i,j}$ with large j, in this section, we bound the expected tardiness for all j by constructing a new execution schedule $\tilde{PS}$ for τ_i. Intuitively, if each job $\tau_{i,j}$'s tardiness in $\tilde{PS}$ schedule is no less than that of $\tau_{i,j}$ in PS, then τ_i's tardiness bound under $\tilde{PS}$ can also bound its tardiness under PS.

In order to clearly illustrate the process of constructing the $\tilde{PS}$ schedule, we shall first describe a stochastic property of $\tau_{i,j}$'s tardiness in the PS. It is evident that $Z_{i,j}$ is a Markov process. This is because $Z_{i,j}$'s future value only depends on its present state. For example, by examining Eq. (31), it can be observed that $Z_{i,j+1}$ depends only on $Z_{i,j}$, $e_{i,j}$ and $p_{i,j}$. According to Lemma 2, $Z_{i,j}$ has a limit probability distribution function $\pi_i(\cdot)$ as j increases. Therefore, if we randomly draw $Z_{i,0}$ from the distribution $\pi_i(x)$ instead of 0, then $Z_{i,j}$ has the probability distribution function $\pi_i(\cdot)$ for all j. In other words, the expected tardiness bound in PS would apply for all j. Based on this observation, we construct $\tilde{PS}$ schedule for τ_i. Let $\tilde{f}_{i,j}$ represent $\tau_{i,j}$'s completion time in $\tilde{PS}$.

Constructing $\tilde{PS}$. We now construct the $\tilde{PS}$ schedule by modifying PS schedule. Note that only one modification needs to be made on PS schedule. That is, for each task τ_i, we randomly draw a value $Z_{i,0}$ from the distribution $\pi_i(\cdot)$ instead of 0 in Eq. (31), which is shown in Fig. 6. Note that the release time and execution time of each job $\tau_{i,j}$ are not

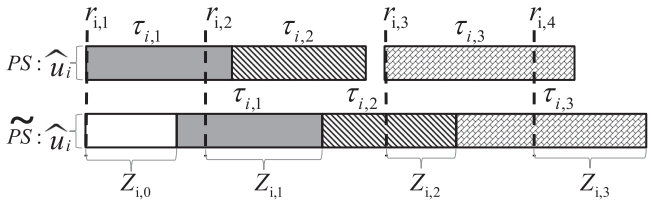


Fig. 6. Recursion for tardiness in $\tilde{PS}$.

altered in both PS and $\tilde{PS}$ schedules. $\tau_{i,1}$ starts executing at $Z_{i,0}$ time units after it is released in $\tilde{PS}$.

Based on the construction of $\tilde{PS}$, $\tau_{i,1}$ starts executing in $\tilde{PS}$ no earlier than the time it starts executing in PS, because $\tau_{i,1}$'s execution is postponed by $Z_{i,0} \geq 0$ in $\tilde{PS}$. Thus, the first job of each task completes in $\tilde{PS}$ no earlier than the time instant it completes execution in PS. Moreover, if $\tilde{f}_{i,j-1} \geq f_{i,j-1}$, then $\tilde{f}_{i,j} \geq f_{i,j}$. This is because in $\tilde{PS}$ schedule $\tau_{i,j}$ starts executing at $r_{i,j} + z_{i,0}$ when $j = 1$, and at $\max\{r_{i,j}, \tilde{f}_{i,j-1}\}$, when $j > 1$. By this induction argument, the completion time of any job in $\tilde{PS}$ is no earlier than the time it completes in PS. Due to these reasons, for all $\tau_{i,j} \in \tau_i$, $\tilde{f}_{i,j} \geq f_{i,j}$ holds. Hence, we have

$$Z^{PS}_{i,j} \leq Z^{\tilde{PS}}_{i,j}. \tag{35}$$

$Z^{\tilde{PS}}_{i,j}$ denotes $\tau_{i,j}$'s tardiness in $\tilde{PS}$ schedule and $Z^{PS}_{i,j}$ denotes $\tau_{i,j}$'s tardiness in PS schedule. Since $Z^{\tilde{PS}}_{i,j}$ has distribution $\pi_i(\cdot)$ for all j, it has $E(Z^{\tilde{PS}}_{i,j}) = E(Z_i) \leq \frac{\sigma'^2_i + \sigma^2_i}{2(\bar{p}_i - \bar{e}_i/\hat{u}_i)}$ according to Eq. (34). Thus, $\forall j$, by Eq. (35),

$$E\left(Z^{PS}_{i,j}\right) \leq \frac{\sigma'^2_i + \sigma^2_i}{2(\bar{p}_i - \bar{e}_i/\hat{u}_i)}. \tag{36}$$

Note that any job's expected tardiness in PS schedule is bounded by Eq. (36). As we discussed earlier, different choices of $\hat{u}_l$ may lead to different derived tardiness bounds. We thus use the following Theorem to optimize $\hat{u}_i$. To simplify the notation used in the following theorem, let $a_i = \sigma'^2_i + \sigma^2_i$. Note that σ_i and σ'_i are given constant values.

Theorem 3. *A constant χ exists such that for all (i,j) where $\tau_{i,j}$ is a job of τ_i,*

$$\frac{E\left(Z^{PS}_{i,j}\right)}{\hat{u}_i} \leq \chi. \tag{37}$$

Proof. To prove this result, we need to find values for $\{\hat{u}_i\}$ and χ. Specially, we need $\chi \geq E(Z^{PS}_{i,j})\frac{1}{\hat{u}_i}, \forall l$. By (36), we have

$$\chi \geq \max_i \left\{ \frac{a_i}{2(\bar{p}_i \hat{u}_i - \bar{e}_i)} \right\}. \tag{38}$$

In (38), $\hat{u}_i$ is a variable and it can be any value that satisfies (4) and (5). Thus, the values for $\{\hat{u}_i\}$ and χ should be identified simultaneously. To achieve this goal, we characterize these requirements using a linear programming: let $\zeta = \chi^{-1}$ and $\hat{u}_i, l = 1, 2, 3, \ldots$ be decision variables,

$$\zeta = \chi^{-1} \leq \min_i \{ \frac{2(\bar{p}_i \hat{u}_i - \bar{e}_i)}{a_i} \}. \tag{39}$$

Solving the following linear programming will yield a valid group of $\hat{u}_i$ and an upper bound on the expected tardiness in PS schedule.

$$\max \zeta \tag{40}$$

$$s.t.\ \ \bar{p}_i\hat{u}_i - \frac{a_i}{2}\zeta \geq \bar{e}_i, \forall i \tag{41}$$

$$\sum_{i=1}^{n} \hat{u}_i \leq m \tag{42}$$

$$\bar{u}_i \leq \hat{u}_i \leq 1, \forall i. \tag{43}$$

In the above linear programming, constraint (41) enforces (39) and constraint (42) enforces (4). We use constraint set (43) to enforce (5). Note that in (43), $\hat{u}_i$ may equal $\bar{u}_i$. However, it is evident that $\hat{u}_i = \bar{u}_i$ can only occur if $\sigma_i'^2 + \sigma_i^2 = 0$ holds by satisfying the above constraints. Given that ζ is maximized, $\sigma_i'^2 + \sigma_i^2 = 0$ becomes the deterministic case, where τ_i's tardiness in PS is 0.

We denote $\hat{u}_i^*$ and ζ^* as the solution to the above linear programming, and $\chi = \frac{1}{\zeta^*}$. Then the theorem is proved. □

5 Combine TPS-Induced and Stochastic-Induced Tardiness

We now combine the TPS-induce tardiness (Section 3) and the stochastic-induced tardiness (Section 4) to show that tardiness in TPS is bounded.

We denote these random variables by capital letters: let $\hat{X}_{l,j} = \hat{e}_{l,j}$, i.e., $\hat{X}_{l,j} = \max_{j'<j} e_{l,j'}$; let $\tilde{X}_l = \tilde{e}_l$; let $\tilde{Y}_l$ denote the stochastic equivalent of $\tilde{p}_l$. The definitions of $\tilde{e}_l$ and $\tilde{p}_l$ are given in Lemma 6. Let $E_{l,j}$ denote the stochastic equivalent of $\eta_{l,j}$. $\eta_{l,j}$ is defined in Eq. (21).

Theorem 4. *τ is a stochastic task system, then the expected tardiness of each job $\tau_{l,j}$ in TPS is upper bounded by*

$$\beta_{l,j} \geq \chi\hat{u}_l + \left(1 - \frac{1}{m - U_L}\right) \times E(\hat{X}_{l,j}) + \frac{(m-1)\rho}{m - U_L} + \frac{E(E_{l,j}) + \sum_{\tau_i \in \tau \setminus \tau_l} \left(E\left(\left\lceil \frac{\varphi_l + \phi_i}{\tilde{Y}_{i,k}} \right\rceil\right) + 1\right) E(\tilde{X}_{i,k})}{m - U_L}, \tag{44}$$

where $\chi = \frac{1}{\zeta^*}$ and ζ^* is the solution to the linear programming (Eqs. (40), (41), (42), and (43)), provided $\bar{U} < m$ holds.

Proof. Recall that $f_{l,j}^*$ denotes the completion time of $\tau_{l,j}$ in TPS. Then, based on Property (P) and Eq. (29),

$$\begin{aligned} f_{l,j}^* &= \hat{f}_{l,j} + x + \hat{e}_{l,j} \\ &= \hat{f}_{l,j} + \hat{e}_{l,j} \\ &\quad + \frac{\eta_{l,j} + (m-1)\rho - \hat{e}_{l,j} + \sum_{\tau_i \in \tau \setminus \tau_l} \left(\left\lceil \frac{\varphi_l + \phi_i}{\tilde{p}_i} \right\rceil + 1\right) \tilde{e}_i}{m - U_L}. \end{aligned} \tag{45}$$

By Eq. (45) and $\hat{f}_{l,j} \geq r_{l,j+1}$ by Definition 10, the tardiness of $\tau_{l,j}$ in TPS, denoted as $Z_{l,j}^{TPS}$, is

$$\begin{aligned} Z_{l,j}^{TPS} &= \max\{f_{l,j}^* - r_{i,j+1}, 0\} \\ &\leq \hat{f}_{l,j} + \hat{e}_{l,j} - r_{i,j+1} \\ &\quad + \frac{\eta_{l,j} + (m-1)\rho - \hat{e}_{l,j} + \sum_{\tau_i \in \tau \setminus \tau_l} \left(\left\lceil \frac{\varphi_l + \phi_i}{\tilde{p}_i} \right\rceil + 1\right) \tilde{e}_i}{m - U_L}. \end{aligned} \tag{46}$$

The quantity $\hat{f}_{l,j} - r_{l,j+1}$, which represents the time from job $\tau_{l,j+1}$'s release time until it is completed in PS, is not given. However, we know that Eq. (46) holds regardless of the value of $\hat{f}_{l,j} - r_{l,j+1}$ is. Thus, $\hat{f}_{l,j} - r_{l,j+1}$ is just the random variable $T_{l,j}^{TPS}$. By taking the expectation of both sides of Eq. (46) and $Z_{l,j}^{PS} = \hat{f}_{l,j} - r_{i,j+1}$, we obtain

$$\begin{aligned} E\left(Z_{l,j}^{TPS}\right) &\leq E\left(Z_{l,j}^{PS}\right) + E(\hat{X}_{l,j}) + \frac{(m-1)\rho}{m - U_L} \\ &\quad + \frac{E(\eta_{l,j}) - E(\hat{X}_{l,j}) + \sum_{\tau_i \in \tau \setminus \tau_l} \left(E\left(\left\lceil \frac{\varphi_l + \phi_i}{\tilde{Y}_{i,k}} \right\rceil\right) + 1\right) E(\tilde{X}_{i,k})}{m - U_L}. \end{aligned} \tag{47}$$

By Theorem 3, $E(Z_{l,j}^{PS}) \leq \chi\hat{u}_{l,j}$. Substituting it into Eq. (47) makes Eq. (49) hold. □

If every task τ_l has a maximum execution time e_l and minimum release time interval p_l, then $E(\hat{X}_{l,j}) \leq e_l$, $E(\tilde{X}_{l,j}) \leq e_l$ and $E(I_{l,j}) \geq p_l$ for all $\tau_{l,j} \in \tau$ (recall that $I_{l,j}$ is defined below Eq. (32)), and

$$E(E_{l,j}) \leq \sum_{\tau_k \in \varepsilon'_{max}} e_i, \tag{48}$$

for each $\tau_{l,j}$, where ε'_{max} denotes a task set consisting of m-1 tasks having largest e_l. Then, $\beta_{l,j}$ (derived in Theorem 4) is bounded by Eq. (49), which is a constant value, for all (l,j). The following corollary immediately follows.

Corollary 2. *If worst-case execution times $\{e_l, \tau_l \in \tau\}$ and worst-case release time interval $\{p_l, \tau_l \in \tau\}$ exist, then for all (l,j), $\beta_{l,j}$ is upper-bounded by a constant.*

6 Video Decoding Case Study

In this section, we illustrate the application of our theoretical results using a real-world case study conducted involving video decoding. Several practical reasons motivate us to choose video decoding as a case study. First, video processing is now pervasively used in many cyber-physical embedded systems such as autonomous driving. For example, Volvo used the latest NVIDIA DRIVE PX2 GPU computing engine to power a fleet of 100 Volvo XC90 SUVs starting to hit the road in 2017 [2]. In autonomous driving systems, multiple real-time video streams may compete for the limited computing resources for data processing (e.g., object recognition). Thus, designing a capable real-time scheduler to properly schedule the video steams among processors is critical to guarantee the efficiency of the entire embedded system. Second, video decoding times vary from frame to frame with MPEG, and the frames per second (FPS) of video streams fluctuates over time. An interesting observation from our experiments is that decoding times are higher for scenes with background changes. Therefore, the MPEG video decoding tasks have stochastic execution times and stochastic periods, which are more suitable to be analyzed using our proposed method, rather than the traditional worst-case analysis.

Experimental Setup. Our scheduler is implemented as a plugin of LITMUS^{RT} [1] on a GPU server with a 12-core Intel Xeon CPU and a NVIDIA "Pascal" GTX 1080 TI GPU, which has 16 SMs. In our case study, we target at a multi-tasking environment where a system of 12 tasks is processed in our experiments, where each task decodes a separate movie trailer and each job of each task decodes one frame of video. The GPU kernel code is only written for handling decoding tasks. For each kernel, a configuration is created which indicates

TABLE 2
Computed Information for 12 MPEG Decoding Tasks (All Times are in MS)

Task	$\overline{e_l}$	σ_l^2	$\overline{p_l}$	$\sigma_l'^2$	WCET	$\min\{p_l\}$
τ_1	7.23	52.20	43.43	0.57	45.35	43.02
τ_2	7.19	51.29	43.67	0.11	35.72	42.97
τ_3	6.87	44.66	43.40	0.09	27.85	42.99
τ_4	6.79	60.17	43.48	0.10	51.26	43.11
τ_5	6.83	44.85	43.53	0.11	66.48	42.96
τ_6	6.98	46.31	43.55	0.15	48.46	43.05
τ_7	7.51	58.72	43.48	0.11	30.63	42.89
τ_8	6.84	62.90	43.37	0.10	55.46	42.94
τ_9	6.75	42.67	43.36	0.09	31.09	42.88
τ_{10}	6.79	43.77	43.64	0.11	44.71	43.01
τ_{11}	7.42	55.94	43.35	0.07	46.36	43.05
τ_{12}	7.27	53.24	43.38	0.08	38.30	43.08

TABLE 3
τ_l's Actual Average Tardiness and Its Expected Tardiness Bound, When $\hat{u}_l = \frac{\overline{e_l}}{\overline{p_l}}$

Task	$\hat{u}_l$	actual tardiness	expected tardiness bound
τ_1	0.166	202.34	483.55
τ_2	0.165	207.14	490.17
τ_3	0.158	60.12	477.55
τ_4	0.156	185.51	486.12
τ_5	0.157	94.32	489.45
τ_6	0.160	170.66	510.02
τ_7	0.173	152.81	496.36
τ_8	0.158	265.91	488.23
τ_9	0.156	109.14	485.62
τ_{10}	0.156	201.74	464.72
τ_{11}	0.171	151.66	499.25
τ_{12}	0.168	253.80	504.55

that 4 SMs are grouped together to decode one frame (one job) at a time. This configuration also specifies that at most 4 tasks can execute simultaneously. To enable preemptive scheduling on GPU, we apply our previously developed GPGPU runtime module GPES [38], which indirectly enables limited preemptive GPGPU execution through breaking both computation and data into fine-grained sub-chunks. To obtain per-job execution times and periods, tasks are run in isolation and all videos are preloaded into memory to avoid page faults. The calculated information for each trailer is given in Table 2. The time unit in the tables is Millisecond (ms).

Scheduling Algorithm. The response time of each task also depends on the scheduling algorithm used to schedule the task system. In our case study, prioritization function $\Upsilon_{l,j}(t)$ is used to simulate Global-FIFO: when jobs are released, they are prioritized by their release times. In other words, according to the definition of the prioritization function, $\Upsilon_{l,j}(t) = r_{l,j}$, $\phi_l = 0$ and $\varphi_l = 0$. Based on Theorem 4, the expected tardiness of each job $\tau_{l,j}$ in FIFO is upper bounded by

$$\beta_{l,j} \geq \chi \hat{u}_l + \left(1 - \frac{1}{m - U_L}\right) \times E(\hat{X}_{l,j}) + \frac{(m-1)\rho}{m - U_L} + \frac{E(E_{l,j}) + \sum_{\tau_i \in \tau \setminus \tau_l} E(\tilde{X}_{i,k})}{m - U_L}, \tag{49}$$

where $\chi = \frac{1}{\zeta^*}$ and ζ^* is the solution to the linear programming (Eqs. (40), (41), (42), and (43)), provided $\bar{U} < m$ holds. For simplicity, we consider the usage of FIFO herein and the above expected tardiness bound, even though global algorithms, which also can be simulated by the prioritization function, with better tardiness bounds exist, such as Global-EDF [9]. These algorithms with better bounds are more complicated to explain, and Global-FIFO is sufficient to illustrate our analysis.

$\hat{u}_l$ Selection. We now examine expected response time bounds for the video decoding tasks when the 12 trailers are scheduled together on the GPU server. In our analysis, PS is considered to be an ideal schedule where for each $\tau_l \in \tau$, at every time instant that τ_l has pending jobs, a fraction $\hat{u}_l$ of the processing capacity of one processor is allocated to τ_l. As we discussed in Section 2.2, different choices of $\hat{u}_l$ will lead to different derived tardiness bounds. Traditionally, PS is also used to analyze the schedulability of tasks under the worst-case scenario [10], [12] and $\hat{u}_l$ is defined to be $\frac{e_l}{p_l}$ for ordinary sporadically task τ_l, where e_l is the worst-case execution time and p_l is the minimal period of τ_l. In the stochastic case, this definition is not applicable. Intuitively, we can define $\hat{u}_l$ as $\frac{\overline{e_l}}{\overline{p_l}}$, where $\overline{e_l}$ is the average execution time and $\overline{p_l}$ is the average period of τ_l. Based on this definition, we can calculate the expected response time bound for τ_l using Eq. (49) and the corresponding results are given in Table 3. In Section 4.2, we optimize $\hat{u}_l$ using a linear programming (Eqs. (40), (41), (42), and (43)) to further reduce the expected response time bound for τ_l and the experimental results are given in Table 4.

Results. The results are collected and shown in Tables 2, 3, and 4. For Table 2, the "$\overline{e_l}$" column denotes the average execution time of τ_l, the "σ_l^2" column denotes the variance of τ_l's execution time; the "$\overline{p_l}$" column denotes the average period of τ_l; the "$\sigma_l'^2$" column denotes the variance of τ_l's period; the "WECT" column denotes the worst-case execution time of τ_l; the $\min\{p_l\}$ column denotes the minimum period of τ_l. For Tables 3 and 4, the "$\hat{u}_l$" column denotes the selected $\hat{u}_l$ for τ_l, the "actual tardiness" column denotes the actual average decoding tardiness of τ_l measured in our experiments; the "expected tardiness bound" column denotes the expected tardiness bound of τ_l calculated using Eq. (49) with the corresponding $\hat{u}_l$.

Comparison with a Worst-Case Analysis. As shown in Table 2, under a worst-case scenario, some decoding tasks are not schedulable, since their WCETs are greater than their

TABLE 4
τ_l's Actual Average Tardiness and Its Expected Tardiness Bound, When $\hat{u}_l$ is Calculated by the LP

Task	$\hat{u}_l$	actual tardiness	expected tardiness bound
τ_1	0.237	202.34	365.12
τ_2	0.300	207.14	376.75
τ_3	0.248	60.12	299.23
τ_4	0.365	185.51	388.85
τ_5	0.339	94.32	295.36
τ_6	0.272	170.66	374.23
τ_7	0.384	152.81	371.25
τ_8	0.366	265.91	375.36
τ_9	0.372	109.14	394.12
τ_{10}	0.367	201.74	401.23
τ_{11}	0.373	151.66	358.35
τ_{12}	0.377	253.80	384.23

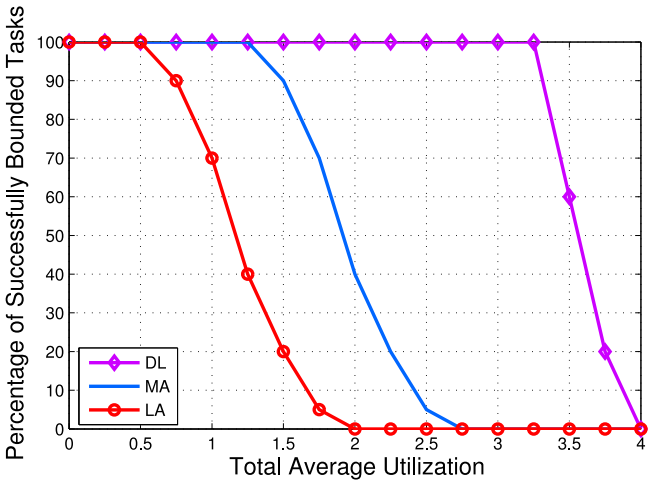


Fig. 7. m = 4, Schedulability.

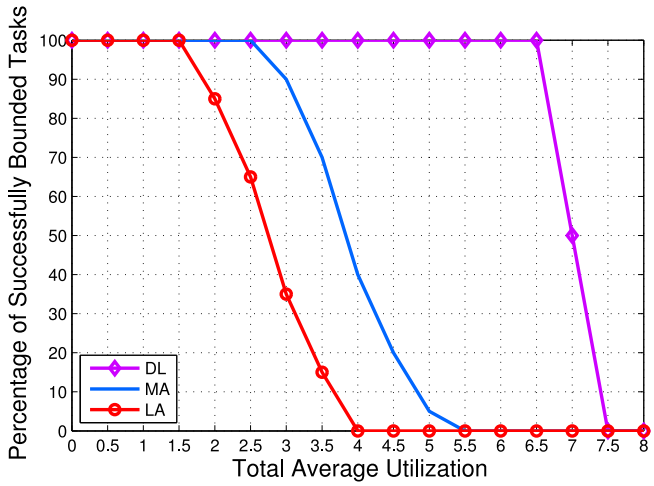


Fig. 8. m = 8, Schedulability.

periods. For example, the WCET of τ_4 is 51.26 ms, which is larger than 43.11 ms, which is the worst-case period of τ_4. In contrast, as shown in Tables 3 and 4, by selecting $\hat{u}_l$ for each task τ_l, all tasks are schedulable and the expected tardiness is bounded. For example, in Tables 3 and 4, the expected tardiness bounds of τ_4 is 486.12 ms and 388.85 ms, respectively.

Tardiness Bounds with Different $\hat{u}_l$. As seen in Tables 3 and 4, we compute the expected tardiness bound for each task using Eq. (49) with different $\hat{u}_l$. As shown in Table 3, when $\hat{u}_l = \frac{\overline{e_l}}{\overline{p_l}}$, all tasks have bounded expected tardiness, which is larger than the actual tardiness obtained from our experiments. The reason is our analysis techniques admit additional pessimism when we upper bound tardiness for each task. Similarly, as seen in Table 4, when $\hat{u}_l$ is calculated by the linear programming (Eqs. (40), (41), (42), and (43)), all tasks are still schedulable with bounded tardiness. An interesting observation from Tables 3 and 4 is that the derived tardiness bound for each task τ_l in Table 4 is smaller than that in Table 3. In other words, the derived tardiness bound is optimized when $\hat{u}_l$ is calculated by the linear programming. For example, the actual average tardiness of τ_7 is 152.81 ms while the derived expected tardiness of τ_7 is 496.36 (371.25) ms in Table 3 (Table 4) when $\hat{u}_l$ is 0.173 (0.384). The reason is that each $\hat{u}_l$ calculated by the linear programming is greater than $\frac{\overline{e_l}}{\overline{p_l}}$ and according to Eq. (49), where $\chi = \frac{1}{\zeta^*}$ and ζ^* is the solution to the linear programming, when $\hat{u}_l$ is greater, τ_l's tardiness bound is smaller.

7 Experiments

In this section, we randomly generate task sets and conduct extensive experiments to evaluate the practicality of our proposed results given in Theorem 4. Our objective is to validate whether the obtained test's utilization cap is reasonable or not, and how practical the derived schedulability is. Our method is also compared with prior methods[18], [27]. In the experimental results, our proposed stochastic schedulability test is denoted as "DL", the test presented in [18] is denoted as "LA", and the test introduced in [27] is denoted as "MA".

Experiment Setup. The UUnifast-Discard method proposed by Emberson et al. [36] was applied to generate a set of utilization values with the given utilization and we adopted a multicore platform in our local cluster to perform our simulations, which has eight 64-bit Intel processors running at 2.0 GHz with 16 GB DDR3 RAM. Tasks' average inter arrival times were randomly selected over $[50\ \text{ms}, 200\ \text{ms}]$ under uniform distribution and the variances were uniformly selected over $[0, 10000\ \text{ms}^2]$. The average utilizations of Tasks were randomly selected over $[0.005, 0.8]$ under uniform distribution. Average execution time for each task was derived by multiplying its average period and average utilization. The variances of tasks' average execution time were uniformly distributed over $[0, 6400\ \text{ms}^2]$. Note that the tasks' execution times and periods were distributed using two independent normal distributions and the means for the two distributions are identical to the calculated average release time interval and the average execution costs. For each category of tasks' average utilization, period, and U_{sum}, 10,000 task sets were randomly produced for multiprocessor platforms consisting of 4 and 8 processors. Every set of Tasks was produced by randomly generating tasks and stopped until the total utilization of all generated tasks was larger than the given utilization cap, and then remove the last task from the task set then the total utilization was equaled to the utilization cap. For every task set, SRT schedulability was tested by "DL," "LA," and "MA". Specially, for DL, we use Theorem 4 to calculate the expected tardiness of each job in TPS, i.e., if all tasks in a task set have finite expected tardiness, this task set is schedulable; otherwise, it is unschedulable.

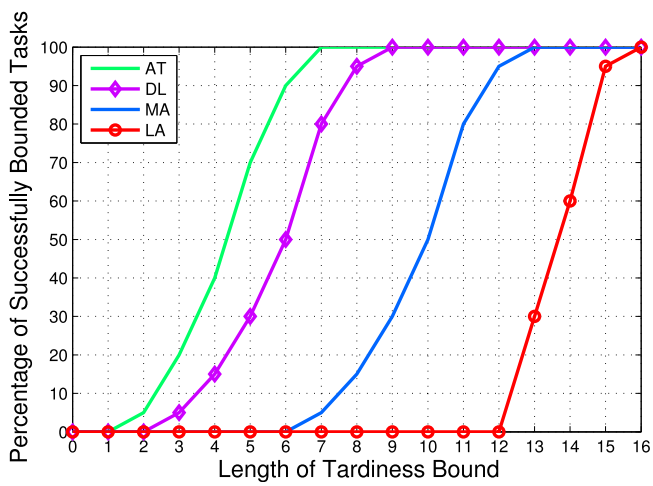


Fig. 9. m = 4, Magnitude of tardiness bound.

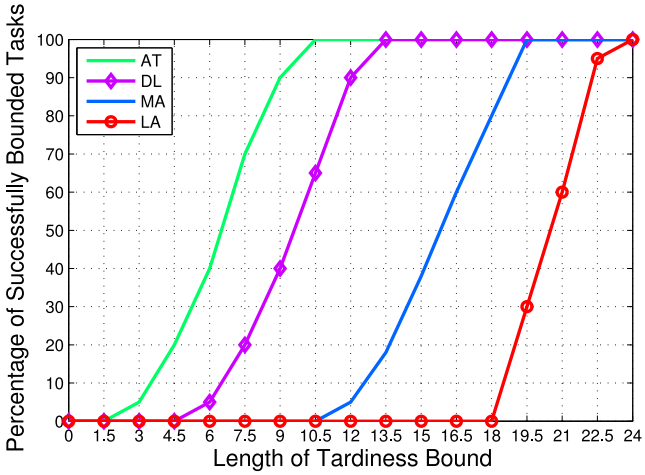


Fig. 10. m = 8, Magnitude of tardiness bound.

Experimental Results. The experimental results are presented in Figs. 7 and 8. X-axis represents the total utilization of each task set and y-axis represents percentage of produced task sets which passed the SRT schedulability test. In Fig. 7, m = 4. In Fig. 8, m = 8. As seen, in experimental results, our proposed test significantly outperform LA and MA by a notable gap. For instance, as shown in Fig. 7, when m = 4, DL can achieve 100 percent schedulability when U_{sum} equals 3.25 while LA and MA cannot achieve 100 percent schedulability when U_{sum} is merely larger than 0.5 and 1.25, respectively. In general, our DL achieves an around 500 percent improvement over LA and an around 200 percent improvement over MA in terms total average utilization.

In Figs. 9 and 10, we compared the magnitude of the tardiness bounds derived by three methods, and the actual observed tardiness at runtime. To ensure all three methods can successfully derive tardiness bounds, the total average utilization is fixed to 0.5. The x-axis denotes the length of tardiness bounds and the y-axis denotes the cumulative percentage of tasks whose tardiness is bounded. In Fig. 9, m = 4. In Fig. 10, m = 8. In all tested scenarios, DL derives a closer tardiness bound to the actual observed tardiness (denoted as AT in figures). In general, DL achieves an around 90 percent improvement over LA and an around 50 percent improvement over MA on average in terms of the magnitude of the derived tardiness bound.

8 Conclusion

We derive a a general soft real-time multiprocessor schedulability analysis framework for practical sporadic task systems specified by stochastic period and execution demand, following probability distributions. We show that this analysis framework can be generally applied to global tunable priority-based schedulers, which allow any job's priority to be changed dynamically at runtime within a priority window of constant length. Experiments demonstrate that our analysis is able to achieve over 500 (200) and 90 percent (50 percent) improvements over the prior deterministic tardiness bound analysis [18] (tardiness bound analysis assuming stochastic execution times but deterministic periods [27]), in terms of schedulability and magnitude of the derived tardiness bound, respectively. A major practical implication is that the task parameters (mean and variance of both execution times and job inter-arrival times) needed to schedule SRT task systems and derive a tardiness bound can be easily estimated from observational data. Many SRT applications [13], [17], [31], [35] in practice where bounded tardiness is acceptable can benefit from our analysis in terms of both schedulability and magnitude of the derived tardiness bound.

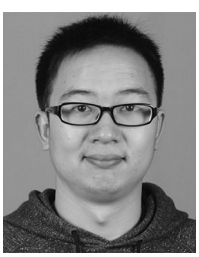
Acknowledgments

This work was supported in part by National Key R&D Program of China (Grant No. 2017YFC0804002), Shenzhen Peacock Plan (Grant No. KQTD2016112514355531), the Science and Technology Innovation Committee Foundation of Shenzhen (Grant No. ZDSYS201703031748284 and No. JCYJ20170817110848086) and the Program for University Key Laboratory of Guangdong Province (Grant No. 2017KSYS008). This work also supported in part by US NSF under Grants CNS 1527727 and CNS CAREER 1750263. Zheng Dong and Yuqun Zhang work was partially accomplished during the visit to Southern University of Science and Technology. This work was partially accomplished during Z. Dong's visit to Southern University of Science and Technology.

References

[1] LITMUS-RT, (2017). [Online]. Available: https://www.litmus-rt.org/
[2] NVIDIA accelerates race to autonomous driving at CES, (2016). [Online]. Available: https://blogs.nvidia.com/blog/2016/01/04/drive-px-ces-recap/
[3] K. J. Åström, *Introduction to Stochastic Control Theory*. North Chelmsford, MA, USA: Courier Corporation, 2012.
[4] K. Balakrishnan, *Exponential Distribution: Theory, Methods and Applications*. Boca Raton, FL, USA: CRC Press, 1996.
[5] R. F. Bass, *Stochastic Processes*, vol. 33. Cambridge, U.K.: Cambridge Univ. Press, 2011.
[6] A. Boukerche, R. W. N. Pazzi, and R. B. Araujo, "HPEQ a hierarchical periodic, event-driven and query-based wireless sensor network protocol," in *Proc. 30th Anniversary IEEE Conf. Local Comput. Netw.*, 2005, pp. 560–567.
[7] J. Chen, S. Paris, and F. Durand, "Real-time edge-aware image processing with the bilateral grid," *ACM Trans. Graph.*, vol. 26, no. 3, 2007, Art. no. 103.
[8] L. K. Church and R. Uzsoy, "Analysis of periodic and event-driven rescheduling policies in dynamic shops," *Int. J. Comput. Integr. Manufacturing*, vol. 5, no. 3, pp. 153–163, 1992.
[9] U. C. Devi, "Soft real-time scheduling on multiprocessors," PhD thesis, Department of Computer Science, Univ. North Carolina at Chapel Hill, Chapel Hill, NC, 2006.
[10] U. C. Devi and J. H. Anderson, "Tardiness bounds under global EDF scheduling on a multiprocessor," *Real-Time Syst.*, vol. 38, no. 2, pp. 133–189, 2008.
[11] Z. Dong, Y. Gu, J. Chen, S. Tang, T. He, and C. Liu, "Enabling predictable wireless data collection in severe energy harvesting environments," in *Proc. IEEE Real-Time Syst. Symp.*, 2016, pp. 157–166.
[12] Z. Dong and C. Liu, "Closing the loop for the selective conversion approach: A utilization-based test for hard real-time suspending task systems," in *Proc. IEEE Real-Time Syst. Symp.*, 2016, pp. 339–350.
[13] M. A. El-Gendy, A. Bose, and K. G. Shin, "Evolution of the internet QoS and support for soft real-time applications," *Proc. IEEE*, vol. 91, no. 7, pp. 1086–1104, Jul. 2003.
[14] C. Gui and P. Mohapatra, "Power conservation and quality of surveillance in target tracking sensor networks," in *Proc. 10th Annu. Int. Conf. Mobile Comput. Netw.*, 2004, pp. 129–143.
[15] D. P. Heyman and M. J. Sobel, *Stochastic Models in Operations Research: Stochastic Optimization*, vol. 2. North Chelmsford, MA, USA: Courier Corporation, 1982.
[16] S. Kumar, T. H. Lai, and A. Arora, "Barrier coverage with wireless sensors," in *Proc. 11th Annu. Int. Conf. Mobile Comput. Netw.*, 2005, pp. 284–298.

[17] M. Lee, A. S. Krishnakumar, P. Krishnan, N. Singh, and S. Yajnik, "Supporting soft real-time tasks in the xen hypervisor," *ACM SIGPLAN Notices*, vol. 45, pp. 97–108, 2010.
[18] H. Leontyev and J. H. Anderson, "Generalized tardiness bounds for global multiprocessor scheduling," in *Proc. 28th IEEE Int. Real-Time Syst. Symp.*, 2007, pp. 413–422.
[19] J. Levinson and S. Thrun, "Automatic online calibration of cameras and lasers," in *Proc. Robot.: Sci. Syst.*, 2013, pp. 29–36.
[20] D. V. Lindley, "The theory of queues with a single server," in *Math. Proc. Cambridge Philosoph. Soc.*, vol. 48, pp. 277–289, 1952.
[21] C. Liu and J. H. Anderson, "An O (m) analysis technique for supporting real-time self-suspending task systems," in *Proc. IEEE 33rd Real-Time Syst. Symp.*, 2012, pp. 373–382.
[22] C. Liu and J. H. Anderson, "Task scheduling with self-suspensions in soft real-time multiprocessor systems," in *Proc. 30th IEEE Real-Time Syst. Symp.*, 2009, pp. 425–436.
[23] C. Liu and J.-J. Chen, "Bursty-interference analysis techniques for analyzing complex real-time task models," in *Proc. IEEE Real-Time Syst. Symp.*, 2014, pp. 173–183.
[24] J. W. S. W. Liu, *Real-Time Systems*, 1st ed. Upper Saddle River, NJ, USA Prentice Hall PTR, 2000.
[25] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proc. 7th IEEE Int. Conf. Comput. Vis.*, 1999, pp. 1150–1157.
[26] M. Mattavelli and S. Brunetton, "Implementing real-time video decoding on multimedia processors by complexity prediction techniques," *IEEE Trans. Consum. Electron.*, vol. 44, no. 3, pp. 760–767, Aug. 1998.
[27] A. F. Mills and J. H. Anderson, "A stochastic framework for multiprocessor soft real-time scheduling," in *Proc. 16th IEEE Real-Time Embedded Technol. Appl. Symp.*, 2010, pp. 311–320.
[28] E. Osuna, R. Freund, and F. Girosit, "Training support vector machines: An application to face detection," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 1997, pp. 130–136.
[29] H. Rinne, *The Weibull Distribution: A Handbook*. Boca Raton, FL, USA: CRC Press, 2008.
[30] H. A. Rowley, S. Baluja, and T. Kanade, "Neural network-based face detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, no. 1, pp. 23–38, Jan. 1998.
[31] A. Srinivasan and J. H. Anderson, "Efficient scheduling of soft real-time applications on multiprocessors," *J. Embedded Comput.*, vol. 1, no. 2, pp. 285–302, 2005.
[32] A. Teichman and S. Thrun, "Group induction," in *Proc. Int. Conf. Intell. Robots Syst.*, 2013, pp. 2757–2763.
[33] P. Viola and M. J. Jones, "Robust real-time face detection," *Int. J. Comput. Vis.*, vol. 57, no. 2, pp. 137–154, 2004.
[34] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, et al., "The worst-case execution-time problem–overview of methods and survey of tools," *ACM Trans. Embedded Comput. Syst.*, vol. 7, no. 3, 2008, Art. no. 36.
[35] W. Yuan and K. Nahrstedt, "Energy-efficient soft real-time CPU scheduling for mobile multimedia systems," *ACM SIGOPS Operating Syst. Rev.*, vol. 37, pp. 149–163, 2003.
[36] P. Emberson, R. Stafford, and R. I. Davis, "Techniques for the synthesis of multiprocessor tasksets," in *Proc. 1st Int. Workshop Anal. Tools Methodologies Embedded Real-time Syst.*, 2010, pp. 6–11.
[37] P. Zhang and D. Ganesan, "Enabling bit-by-bit backscatter communication in severe energy harvesting environments," in *Proc. 11th USENIX Conf. Netw. Syst. Des. Implementation*, 2014, pp. 345–357.
[38] H. Zhou, G. Tong, and C. Liu, "GPES: A preemptive execution system for GPGPU computing," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp.*, 2015, pp. 87–97.



Zheng Dong received the BS degree in computer science from Wuhan University, Wuhan, China, in 2007, and the MS degree in software engineering from the University of Science and Technology of China, Hefei, China, in 2011. He is currently working toward the PhD degree in the Department of Computer Science, University of Texas at Dallas. His research interests include real-time and cyber-physical systems, network coding and wireless sensor network. He received the Outstanding Paper Award at the 38th IEEE RTSS and the best paper nomination at the 23rd IEEE RTCSA.

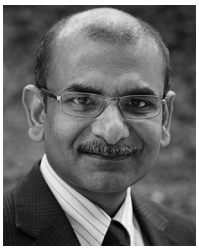
Cong Liu received the PhD degree in computer science from the University of North Carolina at Chapel Hill, in Jul. 2013. He is an assistant professor with the Department of Computer Science, University of Texas at Dallas. His research interests include real-time systems and GPGPU. He has published more than 30 papers in premier conferences and journals. He received the Best Paper Award at the 30th IEEE RTSS, the 17th RTCSA and the 24th RTAS, and the outstanding Paper Award at the 38th IEEE RTSS. He is the recipient of NSF CAREER award and he is a member of the IEEE.

Soroush Bateni is currently working toward the PhD degree in the Department of Computer Science, University of Texas at Dallas. He has been a researcher in the Real-Time Systems lab since 2017. His main research interests are GPU-accelerated embedded cyber-physical systems, including autonomous vehicles, and real-time GPU-accelerated designs.

Zelun Kong received the bachelor's degree in computer science from Wuhan University, in June 2016. He is working toward the PhD degree in the Department of Computer Science, the University of Texas at Dallas. His research interests include artificial intelligence and real-time system.

Liang He is an assistant professor with the University of Colorado Denver. He worked as a research fellow at the University of Michigan, Ann Arbor, during 2015-2017, as a research scientist with the Singapore University of Technology & Design during 2012-2014, and as a research assistant with the University of Victoria during 2009-2011. His research interests include CPSes, cognitive battery management, and networking. He has been a recipient of the best paper/poster awards of MobiSys17, QShine14, WCSP11, and GLOBECOM11, and a best paper candidate of GLOBECOM14. He is a senior member of the IEEE.

Lingming Zhang received the BS and MS degrees in computer science from Nanjing University, in 2007 and Peking University, in 2010, respectively, and the PhD degree from the Department of Electrical and Computer Engineering, University of Texas at Austin, in May 2014. He is an assistant professor with the Computer Science Department, University of Texas at Dallas. His research interests lie broadly in software engineering and programming languages, including automated software analysis, testing, debugging, and verification, as well as software evolution and mobile computing. He has authored more than 40 papers in premier software engineering or programming language conferences and transactions. He has also served on the program/organization committee or artifact evaluation committee for various international conferences (including ICSE, ISSTA, FSE, ASE, ICST, ICSM, and OOPSLA). He has won the Google Faculty Research Award, his research is also being supported by NSF, Huawei, NVIDIA, and Samsung.



Ravi Prakash received the BTech degree in computer science and engineering from the Indian Institute of Technology, in Delhi, and the MS and PhD degree from The Ohio State. He is a professor of computer science with the University of Texas at Dallas. His areas of teaching and research include wireless and sensor networking, mobile computing, multimedia streaming and distributed computing. He is the recipient of multiple research grants from federal agencies and industry including the NSF CAREER award and the NSF Major Research Infrastructure award. He has served the technical and university community in his roles as an editor of the *IEEE Transactions on Mobile Computing*, as the president of the IEEE Dallas section, technical program committee chair of various of various conferences like SRDS, MASS, MSWiM, and is currently the Speaker of the UT Dallas Faculty Senate.

Yuqun Zhang received the BS degree in communications engineering from Tianjin University, Tianjin, China, in 2008, and MS degree in electrical and computer engineering from the University of Rochester from Rochester, NY, in 2010, and the PhD degree in electrical and computer engineering from the University of Texas at Austin, Austin, TX, in 2016. He is currently an assistant professor with the Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen, Guangdong, China. His research interests include software testing and analysis, software engineering for artificial intelligence, and software services computing.