Scheduling Stochastically-Executing Soft Real-Time Tasks: 
A Multiprocessor Approach Without Worst-Case Execution Times

Alex F. Mills
Department of Statistics and Operations Research
University of North Carolina

James H. Anderson
Department of Computer Science
University of North Carolina

Abstract
We introduce a scheduling method where stochastically-executing soft real-time tasks are assigned to simple sporadic servers with predetermined execution budgets. We outline a proof that using this method, any task system whose average-case total utilization is less than the number of processors can be scheduled so that tardiness is bounded in the average case. The constraint on average-case utilization is extremely mild compared to constraints on worst-case utilization because in multiprocessor systems, worst-case execution times may be orders of magnitude higher than average-case execution times. Unlike in previous work, we completely separate the deterministic and stochastic analysis, so that if all execution times are known, an upper bound on tardiness can be computed. Furthermore, the derived expected tardiness bound depends only on the mean and variance of execution times. For soft real-time systems where bounded expected tardiness is acceptable, this result eliminates the need for timing analysis to determine worst-case execution times.

1 Introduction
The focus of this paper is soft real-time workloads specified as implicit-deadline sporadic task systems. Previous work on scheduling such workloads has focused almost exclusively on allocating processing capacity to jobs based on deterministic worst-case execution times. This is a particular impediment in the implementation of soft real-time systems, where some tardiness is acceptable and therefore the pessimistic assumption that every job may require its worst-case execution time is unnecessary.

In uniprocessor systems, it is well known that the earliest-deadline-first (EDF) scheduling algorithm can schedule any system whose utilization is at most one, without missing any deadlines; therefore, both hard and soft real-time workloads can be scheduled on a uniprocessor without any loss of utilization. In multiprocessor systems, Leontyev and Anderson showed that a number of global scheduling algorithms ensure bounded tardiness without utilization loss \(^1\); thus, soft real-time workloads for which bounded tardiness is sufficient can be supported by such systems. This result extended an earlier proof by Devi and Anderson that showed the same of the global earliest-deadline-first (GEDF) scheduling algorithm \(^2\). In these results, utilizations are defined by assuming worst-case execution costs.

Previous work by Mills and Anderson questioned whether using a deterministic worst-case execution time to compute utilization always makes sense in the realm of soft real-time systems \(^5\). The actual worst-case execution time of a task may be observed so infrequently that it may not be worth dedicating such a large amount of processing time to that task if bounded tardiness is acceptable. Moreover, timing analysis tools may add additional pessimism to the calculation of worst-case execution times. These problems are exacerbated on a multiprocessor, where the worst-case scenario may be even less likely but even more costly. Mills and Anderson demonstrated that processing capacity could be allocated based on average-case execution times, with the result that the expected (mean) tardiness of a task is bounded when the system is scheduled using GEDF. Unfortunately, the expected tardiness bound established by them still depends on worst-case execution times.

In this paper, we consider the sporadic stochastic task model (sporadic interrelease times and stochastic execution times), as in \(^5\). We show that reliance on worst-case execution times can be avoided by scheduling the sporadic stochastic tasks on a system of simple sporadic servers. Each server is a deterministic soft real-time task, and each server’s utilization is the order of the average-case utilization of its corresponding sporadic stochastic task.

Unlike in the previous work, a job in such a system does not necessarily run even when it has the highest priority. Instead, a job may only run when its task’s server executes. This is a similar idea to that proposed but not formalized by Calandrino et al. \(^1\), where jobs that did not complete by the time their execution budget ran out could “steal” from the budget of the next job. Not only do we formalize this concept, but our approach is also more sophisticated, because the server budgets may be replenished.

\(^1\)The published version of \(^5\) contains a small error in the tardiness-bound derivation; a corrected version can be found at http://cs.unc.edu/~anderson/papers/
ished more often than the jobs are released. Hence, if the job (of the same task) after an over-running job is separated by enough time, it may not be necessary to steal the execution budget of the next job, but rather to simply consume the budget of another instance of the server task in the interim time. In this paper, we derive expected tardiness bounds under this scheduling approach; such bounds were not considered in [1].

In [5], jobs of every task could potentially be affected by a single job with a much longer-than-average execution time. In this paper, because the servers have a limited budget in each period, only jobs of the same task can be affected by such overruns. There are potential disadvantages in this case, but the result is that in both the uniprocessor and multiprocessor cases, sporadic stochastic tasks have bounded expected tardiness. Moreover, unlike our previous work, the expected tardiness bounds do not depend on worst-case execution times—in fact, tasks that do not have a worst-case execution time can still be scheduled. The resulting bounds also lead to an interesting decision problem in allocating execution times to the servers.

Goal. The goal of this work is to develop a scheduling policy where sporadic task systems with stochastic execution times can be scheduled such that average-case (expected) tardiness is bounded, but where the bound depends on the execution times only in terms of the server budgets, which in turn can be written in terms of the mean and/or variance of execution times.

In the remainder this WIP paper, we first introduce the system model, and then explain the proposed scheduling policy. Finally, we give a brief outline of the analysis.

2 System Model

We consider a system $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ consisting of $n$ sporadic stochastic tasks. The sporadic stochastic model is as follows.

A sporadic stochastic task $\tau_i$ is specified by its period, $p_i$, and its execution time distribution function $G_i(x)$, which gives the probability that a randomly selected job of $\tau_i$ requires at most $x$ time units to execute. We require this distribution to have finite mean and variance. Any of the standard probability distributions used for modeling, such as uniform, exponential, Weibull, etc., have this property. Moreover, in this paper we explicitly do not require these tasks to have a worst-case execution time (WCET) (that is, we do not require the existence of a finite $x$ for which $G_i(x) = 1$; if such an $x$ exists, we do not have to know it to do the analysis). In fact, nowhere in this paper do we use worst-case execution times. We denote the expected, or mean, execution time of $\tau_i$ as $\bar{x}_i$. We denote the $j$th job of $\tau_i$ by $\tau_{i,j}$. We denote the actual execution time of $\tau_{i,j}$ by the random variable $X_{i,j}$. We denote the release time of $\tau_{i,j}$ by $r_{i,j}$ and the deadline of $\tau_{i,j}$ by $d_{i,j}$. Throughout this paper, all deadlines are assumed to be implicit; that is, $d_{i,j} = r_{i,j} + p_i$.

The notion of a sporadic stochastic task is a generalization of that of a sporadic task that is widely used in the real-time scheduling literature, where every task requires $c_i$ time units to complete. In particular, we can model the deterministic case using our notation by setting

$$G_i(x) = \begin{cases} 1 & \text{if } x \geq c_i, \\ 0 & \text{if } x < c_i. \end{cases}$$

We say that a system of sporadic stochastic tasks is stable if and only if its total expected utilization,

$$\bar{u}_{\text{sum}} = \sum_{\tau_i \in \tau} \bar{x}_i/p_i$$

is strictly less than the number of processors (strict inequality is needed for technical reasons in the proofs). We assume throughout the paper that $\bar{u}$ is stable.

Simple Sporadic Servers. Each task $\tau_i \in \tau$ will run on a unique simple sporadic server $T_i$, which will carefully control the amount of time that $\tau_i$ is allowed to execute during each of its periods. This will therefore ensure that worst-case execution times do not appear in the tardiness bound.

Let $T = \{T_1, T_2, \ldots, T_n\}$ be a system of simple sporadic servers. Each simple sporadic server $T_i$ has period $p_i$ and execution budget $e_i$. While $p_i$ is the same as the period of $\tau_i$, $e_i$ is a deterministic execution budget: it is not related to $G_i(x)$, the execution time distribution of $\tau_i$, except that we require $e_i > \bar{x}_i$ (strict inequality is explicitly required in one of the results we use later in the paper).

The budget of $T_i$ is replenished whenever $T_i$ is eligible and backlogged. $T_i$ is eligible if and only if it has never had its budget replenished or at least $p_i$ time units have elapsed since its last replenishment. $T_i$ is backlogged if and only if there is pending work of $\tau_i$. We call $T_{i,j}$ the $j$th instance of $T_i$. The replenishment time of $T_{i,j}$ is denoted by $R_{i,j}$. The deadline of $T_{i,j}$, denoted $D_{i,j}$, is the earliest time at which $T_{i,j+1}$ could be replenished, which is $D_{i,j} = R_{i,j} + p_i$ (i.e., server deadlines are implicit).

The budget of $T_i$ is consumed whenever it has the highest priority according to the scheduling algorithm in use. For example, if the scheduling algorithm is GEDF, then the budget of $T_i$ is consumed whenever it has one of the $m$ earliest deadlines.

It should be clear that the replenishments of $T_i$ do not necessarily correspond to the releases of $\tau_i$. In particular, there are at least as many replenishments of $T_i$ as there are releases of $\tau_i$. From existing results, if we treat $R_{i,j}$ as a release time and $D_{i,j}$ as a deadline, then $T = \{T_1, T_2, \ldots, T_n\}$ is schedulable as a soft real-time
can verify that $T$ is schedulable by EDF on a uniprocessor because $3/5 + 1/3 \leq 1$.

We will use the example given in Fig. 1 to illustrate some important properties of the considered scheduling approach.

**Initial Replenishment.** Both servers are replenished at time 0 because they are both eligible (having never been replenished before) and backlogged (because both $\tau_1$ and $\tau_2$ release jobs at that time). The eligibility times are set to 5 (for $T_1$) and 3 (for $T_2$).

**Consumption Rule.** The budgets are consumed according to how the server instances of $T$ are scheduled. In the example, $T$ is scheduled using EDF on a uniprocessor. For example, $T_2$ begins consuming its budget first at time 0 because $T_{2,1}$ has a higher priority (earlier deadline) than $T_{1,1}$.

**Idleness.** $T$ is scheduled without regard for $\tau$, which may cause idleness. Although $\tau_{2,1}$ finishes executing at 0.8, $T_2$ continues its consumption, even though this means that the processor is idle. In reality, it would be possible to remove this idleness and improve performance; however, to ease the analysis we will assume that $T$ is scheduled as if the scheduler knows only the release times of $\tau$.

**Suspension of Jobs.** At time 4, the budget of $T_{1,1}$ has been consumed, so $\tau_{1,1}$ suspends its execution. At time 5, $T_1$ is eligible, so its budget is replenished, even though $\tau_{1,2}$ has not yet been released. $\tau_{1,1}$ resumes executing and continues executing until time 7, when it completes.

**Replenishment Rule.** The budget of a server is replenished only when it is eligible and backlogged. Therefore, a replenishment will not necessarily occur at the next server deadline. For example, $T_1$ becomes eligible for replenishment at time 10 but is not replenished until time 11.3 because no job of $\tau_1$ can execute until that time.

### 2.2 Remaining Work Process

We define the remaining work process $W_{i,k}$ to be the amount of work of $\tau_i$ due to jobs with deadlines at or prior to $D_{i,k}$ that has not completed by the time $T_{i,k}$ has completed. In a deterministic system, each job of $\tau_i$ would require only the budget of a single job of the server $T_i$; in the stochastic system, some jobs will run longer and hence some work will be left over to run after $T_i$ has been replenished. It is this amount of work that we keep track of in the remaining work process.

For example, in Fig. 1, $W_{1,1} = 1$ because there is one unit of work of $\tau_1$, specifically the unfinished part of $\tau_{1,1}$, which has a deadline at or prior to time 5 but which does not complete by the time $T_{1,1}$ completes (which is time 4). $W_{1,2} = 0$ because all the work of $\tau_1$ with a deadline at or prior to time 10, namely $\tau_{1,1}$, completes by the time $T_{1,2}$ completes at time 9.
3 Outline of Analysis

Proposition 1. Any job of $\tau_i$ with deadline at most $D_{i,k}$ completes no later than the actual time at which the server instance $T_{i,k} + [W_{i,k}/e_i] + 1$ completes.

Proof. Take an arbitrary job $\tau_{i,j}$ where $d_{i,j} \leq D_{i,k}$. Either $\tau_{i,j}$ completes by the time $T_{i,k}$ completes, or it does not. If it does, the result is immediate, because $T_{i,k}$ completes before any later job of $T_{i}$ (in particular, $T_{i,k} + [W_{i,k}/e_i]$).

Otherwise, there is some work remaining on $\tau_{i,j}$ when $T_{i,k}$ completes. The amount of such work is no more than $W_{i,k}$, because $W_{i,k}$ by definition includes all work due to jobs of $\tau_i$ with deadlines at most $D_{i,k}$ that did not complete by the time $T_{i,k}$ completed.

Because all work of $\tau_i$ is completed sequentially, we can guarantee that the remaining work of $\tau_{i,j}$ is completed once $T_i$ has executed for at least $W_{i,k}$ time units following the completion of $T_{i,k}$. Because each instance of $T_i$ executes for $e_i$ time units when there is pending work of $\tau_i$, this means that the remaining work of $\tau_{i,j}$ will complete no later than the time when $T_{i,k} + [W_{i,k}/e_i]$ completes.

For example, in Figure 1, $\tau_{1,1}$ has a deadline no later than $D_{1,1} = 5$, and $[W_{1,1}/e_1] = 1$. By Lemma 1, $\tau_{1,1}$ should complete no later than the time that $T_{1,2}$ completes, namely time 7. Indeed, we see that this is the case, as $\tau_{1,1}$ completes at time 6.

Proposition 1 forms the basis for the following analysis, because it relates the completion time of a sporadic stochastic job with the completion time of a particular server instance. The next proposition describes those completion times on a uniprocessor.

Proposition 2. When $T$ is scheduled on a uniprocessor using EDF, the completion time of $T_{i,[k+W_{i,k}/e_i]}$ is no later than $D_{i,k} + [W_{i,k}/e_i]p_i$.

Proof. In the case where $W_{i,k} = 0$, the lemma simply states that $T_{i,k}$ completes by $D_{i,k}$, which follows immediately from the optimality of EDF on a uniprocessor.

In the case where $W_{i,k} > 0$, we have $R_{i,k+1} = p_i + R_{i,k}$ because $T_i$ is eligible and backlogged, and replenishments of $T_i$ continue to occur every $p_i$ time units until all remaining work of $\tau_i$, including the work in $W_{i,k}$, is completed. Therefore, $R_{i,[k+W_{i,k}/e_i]}$ occurs $[W_{i,k}/e_i]p_i$ time units after $R_{i,k}$ does; i.e., at time $D_{i,k} + ([W_{i,k}/e_i] - 1)p_i$.

Because $T$ is scheduled using EDF, $T_{i,[k+W_{i,k}/e_i]}$ completes no later than its deadline, which is $p_i$ time units later than its replenishment time, i.e., at time $D_{i,k} + [W_{i,k}/e_i]p_i$.

A similar proposition can be proved for multiprocessor scheduling, where $T$ is scheduled using a global algorithm with bounded tardiness. A number of scheduling algorithms, such as global earliest-deadline-first (GEDF), can schedule $T$ on a multiprocessor with bounded tardiness. In particular, any algorithm with window-constrained priorities has this property.

The results proved so far lead almost directly a bound on the completion time of an arbitrary server instance $T_{i,j}$ in terms of the remaining work process associated with a corresponding server instance. In the following paragraphs, we outline the remaining analysis.

Analysis of Remaining Work Process. In order to derive a tardiness bound for an arbitrary job $\tau_{i,j}$, we need to examine the remaining work process. We can write a recursion for the remaining work process involving the actual execution times of all the jobs in $\tau_i$ and the execution budget $e_i$. This recursion implies that if all the execution times are known, an exact upper bound on tardiness can be calculated.

Because actual execution times are not actually known in advance, we apply known results from stochastic processes to examine the behavior of the remaining work process. For example, we write an expression for the expected (average) remaining work. Such an expression leads to the derivation of an average-case tardiness bound for the sporadic stochastic tasks.

Choice of Execution Budgets. We stated the problem as if the execution budgets of the servers were predetermined. In fact, the execution budget for $T_i$ may be chosen in any way such that $\bar{e}_i < e_i \leq p_i$. The choice of execution budgets affects the tardiness bound, and hence it should be treated as a design decision. It is possible to ensure that execution budgets are chosen in such a way that they depend only on mean and variance of the execution time distributions, so that worst-case execution times do not appear in the final tardiness bounds.

References


