A Multiprocessor Server-Based Scheduler for Soft Real-Time Tasks with Stochastic Execution Demand[*]

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 utilize a multiprocessor server-based approach to schedule a general class of soft real-time systems with stochastic execution demands, when bounded average-case tardiness is sufficient for schedulability. A key feature of the task model considered here is that the stochastic execution-time demands can have arbitrary amounts of dependence within pre-specified time intervals of bounded length. This is an important practical step forward from requiring complete independence of execution times between successive jobs of the same task. Our main result does not require the scheduler to know the execution time of each job in advance, and requires only average-case utilization to be bounded by the number of processors. This constraint is 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.

1 Introduction

Previous work on scheduling soft real-time 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 deadline tardiness is acceptable and therefore the pessimistic assumption that every job may require its worst-case execution time is unnecessary.

In multiprocessor systems, Leontyev and Anderson showed that a number of global scheduling algorithms ensure bounded tardiness without utilization loss [8]; thus, soft real-time workloads for which bounded tardiness is sufficient can be supported on 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 [4]. In these results, utilizations are defined by assuming worst-case execution costs.

In previous work, we questioned whether using a deterministic worst-case execution time to compute utilization always makes sense in the realm of soft real-time systems [13]. 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. In [13], we showed 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 we derived still depends on worst-case execution times, and the task model also requires job execution times to be completely independent of one another.

In this paper, we define a task model for soft real time systems with stochastic execution demands (SRT-SED). SRT-SED generalizes the model used in [13] and also other types of task models, including the sporadic task model and a stochastic or deterministic multiframe task model. We analyze SRT-SED task systems when scheduled on a system of simple sporadic servers. Each server is a deterministically-specified sporadic soft real-time task. By applying existing analysis of soft real-time tasks that comprise the server system, we prove a tardiness bound that has two special cases: a deterministic, worst-case tardiness bound (when the underlying system has worst-case utilization bounded by the number of processors), and an expected, or average-case, tardiness bound (when the underlying system has average-case utilization bounded by the number of processors). The average-case tardiness bound also yields a bound on the quantiles or percentiles of the tardiness distribution.

In this work, a job may only run when its task's server executes, and jobs that do not complete by the time their execution budget runs out "steal" from the budget of the next job of the same task. These ideas are not new in the real-time systems literature, as the server scheme we use is similar to bandwidth reservation. For example, Abeni and Buttazzo used a bandwidth reservation strategy to schedule soft real-time task systems on a uniprocessor, where successive jobs of the same task have independent randomly distributed ex-

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

ecution times [1]. However, only the case where worst-case utilization is less than one was considered. Similarly, Brandt et al. developed the resource allocation/dispatching (RAD) model for uniprocessors, where tasks could have different worst-case execution times and periods from job to job; the RAD model led to the development of the rate-based-earliest-deadline scheduler [3]. However, the RAD model assumes that for real-time tasks, these parameters (e.g, worst-case execution times) are known from job to job, in order to check whether it is feasible to increase or decrease a task's utilization. Baruah et al. showed that constant bandwidth servers (CBS) could be implemented on a multiprocessor to provide a quality-of-service guarantee for each task [2], and Pellizoni and Caccamo subsequently incorporated the reclaiming of unused resources in a multiprocessor CBS [15], but only hard real-time tasks with worst-case execution times were considered in these research efforts. Finally, Leontyev and Anderson developed a multiprocessor bandwidth reservation scheme that encapsulates both hard and soft real-time tasks [9], but again only for tasks with worst-case execution times.

In this paper, we present a novel analysis of tardiness when a server-based scheduling algorithm is used to schedule task systems fitting the SRT-SED model: in particular, such task systems may be over-provisioned in the worst case; to our knowledge, this possibility has not been considered in any previous work on multiprocessors. Unlike the papers mentioned above, our analysis does not place any constraint on worst-case utilization, does not require the scheduler to know the execution time of the job being scheduled, and does not require worst-case execution times to be known or even to exist (the only information needed is average-case execution times and variance in execution times).

This paper also differs significantly from [13], where 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 a much more general class of soft real-time systems can be analyzed. Moreover, unlike in [13], 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.

Main Contribution. We derive expected response time bounds for tasks whose execution costs are stochastic, under the scenario where those tasks are executed on simple sporadic servers. The expected response time bound can be applied to determine expected tardiness bounds and probabilistic deadlines. Our task system specification is general enough that certain types of dependence can be modeled within its framework. Our tardiness bounds are applicable if the *average-case* total utilization is less than the system's capacity, even if the system is over-utilized in the worst case or does not have a worst case. We also separate the deterministic tardiness analysis from the stochastic analysis in such a way that if better deterministic tardiness analysis becomes available in the future, the results of this paper can immediately be applied without having to repeat any of the stochastic analysis.

Organization. The remainder of this paper is organized as follows. In Section 2, we give a formal definition of the system model. Section 3 consists of a number of technical results that deal with the uncertainty in a stochastic system. Section 4 derives the multiprocessor tardiness bounds. These bounds are the main results of the paper. In Section 5, we discuss considerations necessary for the implementation of our scheduling approach, and in Section 6, we remark on possible uses and future directions for our model.

2 System Model

We consider a system $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ consisting of n tasks. A *task* is a possibly infinite sequence of jobs. *Jobs* are demands for processing time, each with an associated deadline. Jobs of the same task must be executed sequentially and in FIFO order. However, in this paper, we do not assume that successive jobs of the same task are identical.

We consider the above system under different combinations of the following assumptions, which are illustrated below via several examples.

Assumption G. Assume that for each task $\tau_i \in \tau$, we can find a set of time points $\{t_{i,0}, s_{i,1}, t_{i,1}, s_{i,2}, t_{i,2}, \ldots\}$ (which may be finite or countably infinite) and a constant p_i such that for all $j \geq 1$ the following four conditions hold:

- $t_{i,j} - s_{i,j} \leq p_i$,
- $s_{i,j+1} - s_{i,j} \geq p_i$,
- no job of τ_i is released in $[t_{i,j-1}, s_{i,j})$, and
- no job of τ_i released in $[s_{i,j}, t_{i,j})$ has a deadline later than $t_{i,j}$.

Assumption D. Denote the total amount of processing time demanded by jobs of τ_i released in $[s_{i,j}, t_{i,j})$ by $X_{i,j}$. There exists a constant e_i such that for all tasks $\tau_i \in \tau$, and for all $j = 1, 2, \ldots$, $X_{i,j} \leq e_i$.

Assumption S. Define $X_{i,j}$ as above. For all tasks $\tau_i \in \tau$, the sequence $\{X_{i,j}, j = 1, 2, \ldots\}$ is a sequence of independent and identically distributed random variables with mean $\bar{e}_i$ and variance σ_i^2.

We will assume that Assumption G (a general assumption about the analyzed SRT-SED task system) holds throughout the paper. When Assumption D (the deterministic case) or Assumption S (the stochastic case) are needed, this will be stated explicitly. Note that the major contribution of this paper is the set of results for the stochastic case; the results for the deterministic case are included as a basis for comparison and for the sake of completeness; determinism is always a special case of a stochastic system, namely, when variance is zero.

To illustrate the types of task systems that are special cases of SRT-SED, we provide a few examples.

Sporadic Task System. Consider the typical constrained-deadline sporadic task system where each $\tau_i \in \tau$ has period P_i and worst-case execution time C_i. We can see that this system satisfies Assumptions G and D by setting $s_{i,j}$ to be the release time of the jth job of sporadic task τ_i, setting $t_{i,j}$ to be the deadline of the jth job of τ_i, setting $p_i = P_i$, and setting $e_i = C_i$.

Sporadic Stochastic Task System. Consider the task system introduced in [13], where each $\tau_i \in \tau$ has a period P_i and an execution time that is independent and identically distributed with mean $\bar{e}_i$ and variance σ_i^2. The definition of this task system makes no assumption that there is a worst-case execution time, so Assumption D is not satisfied; however, we can see that this system satisfies Assumption G by setting $s_{i,j}$ to be the release time of the jth job of sporadic task τ_i, setting $t_{i,j}$ to be the deadline of the jth job of τ_i (which must be implicit or constrained), and setting $p_i = P_i$. Because the execution times of successive jobs are independent and identically distributed random variables, and only one job is released between $s_{i,j-1}$ and $s_{i,j}$, for all j, $\bar{e}_i$ and σ_i^2 satisfy Assumption S.

Multiframe Task System. Consider a task system where successive jobs of τ_i cycle deterministically through K_i different execution modes, as in [14]. Let P_i denote the period of the cycle, and let all relative deadlines be constrained to at most P_i. Furthermore, suppose that each execution mode is characterized by a worst-case execution time $C_{i,k}, k = 1, 2, \ldots, K_i$. We can see that this system satisfies Assumptions G and D by setting $s_{i,j}$ to be the release time of the (jK_i)th job (that is, the release time of the first job in the cycle), setting $t_{i,j}$ to be the deadline of the $((j+1)K_i - 1)$th job of τ_i (that is, the deadline of the last job in the cycle), setting p_i equal to P_i, and setting $e_i = \sum_{k=1}^{K_i} C_{i,k}$ (that is, the worst-case execution time of the entire cycle).

Stochastic Multiframe Task System. Consider a multiframe system like that above, but where each execution mode is characterized by a different random execution time (that is, the execution time is drawn from a different distribution for each mode), such that the execution time in mode k in one cycle is independent of the execution time in any mode in any other cycle. Note that execution times may be dependent *within* a cycle; for example, the execution time in mode k may be correlated with the execution time in mode $k-1$ within the same cycle.

We can see that this system satisfies Assumption G by setting $s_{i,j}$ to be the release time of the (jK_i)th job (that is, the release time of the first job in the cycle), setting $t_{i,j}$ to be the deadline of the $((j+1)K_i - 1)$th job of τ_i (that is, the deadline of the last job in the cycle), and setting p_i equal to P_i. To show that this system satisfies Assumption S, note that the assumption of independence between cycles guarantees that the total execution time of one cycle is independent of the total execution time of any other cycle, and the fact that execution times in each mode are identically distributed guarantees that the total execution time of the cycle is identically distributed. Furthermore, the variance of the total execution time of the cycle can be written as a function of the variance of the execution time of each mode and the covariance between execution times of different modes (see, e.g., [16] for details).

2.1 Independence

In order for Assumption S to be satisfied, execution time demands in successive $[s, t)$ time intervals must be independent from one another. Many applications of real-time systems have jobs with clear dependence on one another. Such dependence may result explicitly from the type of application (e.g., in a video decoding application, if a frame takes a long time to decode because it is part of a scene with significant movement, then its successor is more likely to also take a long time to decode) or they may result from characteristics of the system (e.g., if a job runs quickly due to a warm cache, then its successor is more likely to also run quickly if it requires access to the same set of data). We note that our system definition does *not* require execution times of *each job* to be independent of one another. However, allowing arbitrary dependence would result in completely intractable analysis—in fact, complete independence is commonly assumed in many queueing applications (for example, independence of successive service times) for the purpose of tractability. The system model presented in this paper lies somewhere between the most true-to-life model (which would be intractable), and the somewhat unrealistic model that calls for independence of every job's execution time (which was used in prior work on real-time systems [1, 13]). Moreover, because the intervals in which dependence is permitted can be set by the user, there is an interesting design trade-off: longer intervals result in a more accurate model at the expense of larger tardiness bounds (as we shall see). In some applications, these intervals may occur naturally. In other cases, these intervals may have to be imposed by the system designer: e.g., for dependencies arising from cache affinity, the beginning of a new independent interval corresponds in practical terms to a cold cache. In the latter case, the system designer would need to calculate the execution time demand by assuming a cold cache at the beginning of each interval.

2.2 Simple Sporadic Servers

The scheduling scheme we will use in this paper involves a server abstraction. 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 τ_i is allowed to execute. This will therefore ensure that worst-case execution times do not appear in the average-case 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 b_i. We will discuss conditions that must be satisfied by b_i in the next section. 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 (recall that p_i was defined in Assumption G). T_i is *backlogged* if and only if there is pending work of τ_i. We call $T_{i,j}$ the jth *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, where m is the number of processors.

2.3 Scheduling Example

Figure 1(b) shows an example with sporadic stochastic tasks τ_1, with period 5 and execution cost drawn from some distribution with mean 2, and τ_2, with period 3 and execution cost drawn from some distribution with mean 0.75. In this model, we let $\tau_{i,j}$ denote the jth job of task τ_i. $\tau_{1,1}$ is released at time 0 and has execution cost 4, $\tau_{1,2}$ is released at time 6.3 and has execution cost 1.5, and $\tau_{1,3}$ is released at time 11.3 and has execution cost 2. $\tau_{2,1}$ is released at time 0 and has execution cost 0.8, and $\tau_{2,2}$ is released at time 3 and has execution cost 1.7. The schedule is not shown after time 13. As we discussed above, the sporadic stochastic task model satisfies the assumptions required for task systems in our paper.

Figure 1(a) shows two servers. T_1 corresponds to τ_1. It has period 5.0 (the same as τ_1), and budget 3. T_2 corresponds to τ_2. It has period 3 and budget 1. We 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 Figure 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 τ_1 and τ_2 release jobs at that time). The deadlines are set to 5 (for T_1) and 3 (for T_2).

Consumption Rule. 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 idleness in τ. Although $\tau_{2,1}$ finishes executing at 0.8, T_2 continues its

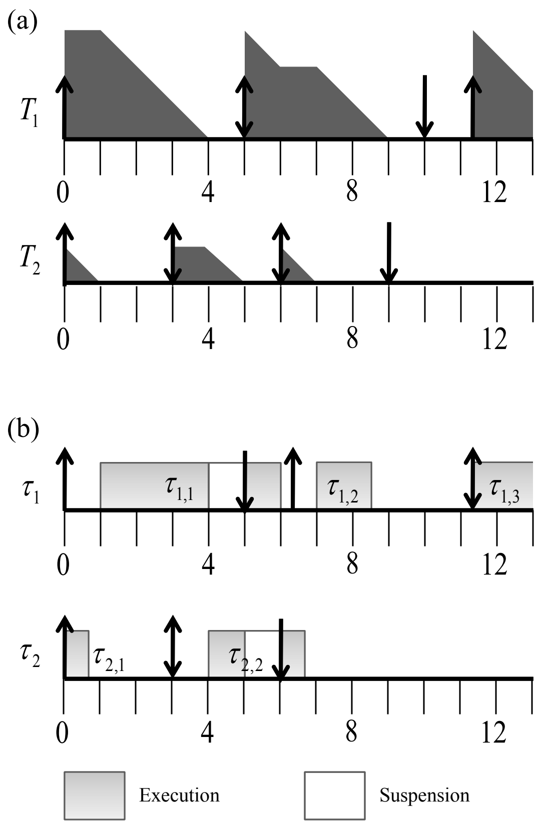

Figure 1: Example servers and sporadic stochastic tasks on a uniprocessor. **(a)** For servers T_1 and T_2, $\uparrow$ denotes replenishment time and $\downarrow$ denotes deadline; the budget is shaded for each server. T_1 and T_2 are scheduled using EDF. **(b)** For the sporadic stochastic tasks τ_1 and τ_2, $\uparrow$ denotes release time and $\downarrow$ denotes deadline; the actual execution times are shaded, while suspensions are shown in white.

consumption, even though this means that the processor is idle. This prevents server instances from experiencing self-suspensions. Such self-suspensions potentially could be allowed, at the expense of needing to impose utilization constraints to bound server tardiness [11]. However, we do not consider that possibility in this paper.

Suspensions. 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 6, 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 τ_1 can execute until that time.

Table 1: Summary of Notation

Notation	Definition
$[s_{i,j}, t_{i,j})$	jth interval during which dependence is allowed for τ_i
$X_{i,j}$	execution requirement for τ_i over $[s_{i,j}, t_{i,j})$
b_i	budget of server T_i
$T_{i,j}$	jth instance of T_i
$R_{i,j}$	jth replenishment time of T_i
$D_{i,j}$	deadline of $T_{i,j}$
(j)	largest index k such that $R_{i,k} < t_{i,j}$
$W_{i,j}$	remaining work of τ_i with deadline $\leq t_{i,j}$ not completed by $D_{i,(j)+1}$

2.4 Remaining Work Process

We define the *remaining work process* $W_{i,j}$ to be the amount of work of τ_i due to jobs with deadline at most $t_{i,j}$ that has not completed by the time the budget of server instance $T_{i,(j)+1}$ has been exhausted, where $(j) = \max\{k : R_{i,k} < t_{i,j}\}$. That is, (j) is the index of the last server instance that replenishes T_i before $t_{i,j}$. We will use the parenthetical function in conjunction with all other server-related notation. For example, $R_{i,(j)}$ is the replenishment time of $T_{i,(j)}$ and $D_{i,(j)}$ is the deadline of $T_{i,(j)}$.

To maintain consistency in the analysis, we define $W_{i,0} = 0$ because no work of τ is released prior to $t_{i,0}$, which is defined to be time 0.

Note that in general $T_{i,(j)}$ is not the jth replenishment of T_i, because replenishments of T_i may occur more frequently than time points in the set $\{s_{i,j}, j = 0, 1, \ldots\}$. This will happen if the interval $(t_{i,j-1}, s_{i,j})$ is long enough and T_i is backlogged during that interval.

A full summary of notation is given in Table 1.

Lemma 1. *Any job of τ_i with a deadline no later than $t_{i,j}$ completes no later than the actual time at which the server instance $T_{i,k}$ completes, where $k = (j) + 1 + \lceil W_{i,j}/b_i \rceil$.*

Proof. Take an arbitrary job with deadline no later than $t_{i,j}$. Either $\tau_{i,j}$ completes by the time $T_{i,(j)+1}$ completes, or it does not. If it does, the result is immediate, because $T_{i,(j)+1}$ completes before any later instance of T_i (in particular, $T_{i,k}$).

Otherwise, there is some work remaining for our job when $T_{i,(j)+1}$ completes. The amount of such work is no more than $W_{i,j}$, because $W_{i,j}$ by definition includes *all* work due to jobs of τ_i released prior to $t_{i,j}$ that did not complete by the time $T_{i,(j)+1}$ completed.

Because all work of τ_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 $W_{i,j}$ additional time units following the completion of $T_{i,(j)+1}$. Because each instance of T_i completes b_i time units of work on τ_i when there is pending work of τ_i, this means that the remaining work of $\tau_{i,j}$ will complete no later than the time when $T_{i,k}$ completes. □

3 Analysis of the Remaining Work Process

This section consists of technical results that lead to a bound on the expected value of the remaining work process.

Lemma 2. *The remaining work process satisfies*

$$W_{i,0} = 0 \tag{1}$$
$$W_{i,j} = \max\{0, W_{i,j-1} + X_{i,j} - Y_{i,j} b_i\}, \text{ for } j = 1, 2, \ldots \tag{2}$$

where $Y_{i,j}$ is the number of times T_i is replenished in $[t_{i,j-1}, t_{i,j})$.

Proof. First, observe that prior to the first release, there is clearly no remaining work, so (1) holds.

Next, by definition, $X_{i,j}$ is the amount of work that may *add* to the remaining work that is counted in $W_{i,j}$ but not in $W_{i,j-1}$, because it is the amount of work that is demanded by τ_i in $[s_{i,j}, t_{i,j})$ (remember that no jobs are released in $[t_{i,j-1}, s_{i,j})$). Finally, observe that by the time $T_{i,(j)+1}$ completes, either an additional $Y_{i,j} b_i$ units of work will have been allocated to τ_i, or there will be no more remaining work of τ_i; hence, either we subtract $Y_{i,j} b_i$ from the remaining work and are left with positive remaining work, or else we subtract all the remaining work, and are left with zero (justifying the $\max$ operation). □

Define the process $Z_{i,j}$ as follows:

$$Z_{i,0} = 0 \tag{3}$$
$$Z_{i,j} = \max\{0, Z_{i,j-1} + X_{i,j} - b_i\}, \text{ for } j = 1, 2, \ldots. \tag{4}$$

The next lemma relates this new process with the remaining work process. Note that when we describe the relationship between two random quantities as $A \leq B$, we mean "the probability that A is at most B is 1."

Lemma 3. *$W_{i,j} \leq Z_{i,j}$ for all i, j.*

Proof. First, $W_{i,0} = Z_{i,0} = 0$. Now, proceed by induction. Suppose that $W_{i,j-1} \leq Z_{i,j-1}$. Observe that if $W_{i,j} > 0$ then $Y_{i,j} \geq 1$, because T_i will be replenished at least once in $[t_{i,j-1}, t_{i,j})$ when there is any remaining work of τ_i. Thus

$$\begin{aligned} W_{i,j} &= \max\{0, W_{i,j-1} + X_{i,j} - Y_{i,j} b_i\} \\ &\leq \max\{0, W_{i,j-1} + X_{i,j} - b_i\} \\ &\leq \max\{0, Z_{i,j-1} + X_{i,j} - b_i\} \\ &= Z_{i,j}. \end{aligned}$$

By induction, the result holds. □

Lemma 4. *If $\mathsf{E}(X_{i,j}) < b_i$, then $\{Z_{i,j}, j = 1, 2, \ldots\}$ has a limit Z_i, as $j \to \infty$.*

Proof. In [10], analytical results are presented concerning processes having the form of $Z_{i,j}$. It is shown there that as j increases, the distribution of $Z_{i,j}$ has a limit if and only if $\mathsf{E}(X_{i,j} - b_i) < 0$. □

Lemma 5. *[6, p. 474] Suppose that $\mathcal{A}_n$, $n = 1, 2, \ldots$ is drawn independently from a distribution with mean $\mu_\mathcal{A}$ and variance $\sigma^2_\mathcal{A}$, and $\mathcal{B}_n$, $n = 1, 2, \ldots$ is drawn independently from a distribution with mean $\mu_\mathcal{B}$ and variance $\sigma^2_\mathcal{B}$. Assume $\mu_\mathcal{A} > \mu_\mathcal{B}$. Let $V_0 = 0$, and $V_n = \max\{0, V_{n-1} + \mathcal{B}_n - \mathcal{A}_n\}$. Let $V = \lim_{n\to\infty} V_n$ be the limit (in distribution) of V_n. Then*

$$\mathsf{E}(V) \le \frac{\sigma^2_\mathcal{A} + \sigma^2_\mathcal{B}}{2\mu_\mathcal{A}(1 - \mu_\mathcal{B}/\mu_\mathcal{A})}.$$

Lemma 6. *If $\mathsf{E}(X_{i,j}) < b_i$, then $\mathsf{E}(Z_i) \le \frac{\sigma_i^2}{2(b_i - \bar{e}_i)}$.*

Proof. (4) is an example of Lindley's recursion, which describes the waiting time in a queue: the waiting time of the jth customer is equal to the waiting time of the $(j-1)$st customer, plus the service time of the $(j-1)$st customer, minus the amount of time between the two customers' arrivals (because the jth customer does not wait during that time, while the $(j-1)$st customer may). In other words, if there were a queueing system where service times were $\{X_{i,1}, X_{i,2}, \ldots\}$ and customers arrived every b_i time units, then the waiting time of the jth customer would be $Z_{i,j}$. By applying the known upper bound given in Lemma 5 to the limiting distribution of this process, we find that the expected value of Z_i is at most

$$\frac{\sigma_i^2/b_i}{2(1 - \bar{e}_i/b_i)}, \tag{5}$$

which after simplifying yields the result (note that, in applying Lemma 5, $\sigma^2_\mathcal{A} = 0$ because b_i is a constant). □

We now use Lemma 4, i.e., the fact that Z_i has a limiting distribution. This distribution will be important from a theoretical standpoint because it will allow us to create yet another process, which will still upper-bound $W_{i,j}$ but which behaves like Z_i for all j, not just as $j \to \infty$.

$\{Z_{i,j}, k = 1, 2, \ldots\}$ is a Markov process because its future evolution depends on its history only through its present state. $Z_{i,j}$ depends only on $Z_{i,j-1}$, and $X_{i,j}$, which is independent of everything else. Therefore, $Z_{i,j}$ does not depend on $Z_{i,j-2}, Z_{i,j-3}$, etc.

Let $\pi_{i,j}(x)$ be the probability density function of $Z_{i,j}$. The kernel of $Z_{i,k}$ [12, p. 59] is a probability transition function $K(\cdot)$ that satisfies

$$\pi_{i,j}(x) = \int K_i(x, y)\pi_{i,j-1}(y)dy. \tag{6}$$

As k increases, $Z_{i,k}$ approaches its limit Z_i. Z_i has probability density function $\pi_i(x)$, which does not depend on j. Therefore, $\pi_i(x)$ satisfies

$$\pi_i(x) = \int K_i(x, y)\pi_i(y)dy. \tag{7}$$

This leads us to the formulation of a process $\{\tilde{Z}_{i,j}, j = 1, 2, \ldots\}$, which has distribution $\pi_i(x)$ for all j:

$$\tilde{Z}_{i,0} \text{ is distributed according to the pdf } \pi_i(x) \tag{8}$$

$$\tilde{Z}_{i,j} = \max\{0, \tilde{Z}_{i,j-1} + X_{i,j} - b_i\}, \text{ for } j = 1, 2, \ldots. \tag{9}$$

Lemma 7. *$Z_{i,k} \le \tilde{Z}_{i,k}$, for all i, k.*

Proof. First, observe that $Z_{i,0} = 0$, while $\tilde{Z}_{i,0}$ is drawn from the non-negative probability distribution $\pi_i(\cdot)$. Therefore, $Z_{i,0} \le \tilde{Z}_{i,0}$, so the lemma holds for $k = 0$.

Now, suppose that the lemma holds for some arbitrary $j-1$, i.e., $Z_{i,k-1} \le \tilde{Z}_{i,j-1}$. We will show that the lemma holds for j.

$$\begin{aligned}\tilde{Z}_{i,j} &= \max\{0, \tilde{Z}_{i,j-1} + X_{i,j} - b_i\}\\ &\ge \max\{0, Z_{i,j-1} + X_{i,j} - b_i\}\\ &= Z_{i,j}.\end{aligned}$$

The result follows by induction. □

Lemma 8. *$\mathsf{E}\left(\tilde{Z}_{i,j}\right) = \mathsf{E}(Z_i)$ for all j.*

Proof. We show that $\tilde{Z}_{i,j}$ has distribution $\pi_i(\cdot)$ for all j. The result immediately follows, because Z_i has the same distribution.

First observe that $\tilde{Z}_{i,0}$ is drawn from the distribution $\pi_i(\cdot)$, by definition, so the claim holds for $j = 0$. Now, suppose that $\tilde{Z}_{i,j-1}$ has distribution $\pi_i(\cdot)$ for some arbitrary $j-1$. Then the same holds for j:

$$\begin{aligned}\pi_{i,j}(x) &= \int K_i(x, y)\pi_{i,j-1}(y)dy\\ &= \int K_i(x, y)\pi_i(y)dy\\ &= \pi_i(x).\end{aligned}$$

Here, the first equality is simply a restatement of (6), the second equality is a result of the inductive hypothesis, and the third equality follows from (7). The result follows by induction. □

By combining Lemmas 3, 6, 7, and 8, we have the following result, which is the main result of this section.

Theorem 1. *If $\mathsf{E}(X_{i,j}) < b_i$, then*

$$\mathsf{E}(W_{i,j}) \le \frac{\sigma_i^2}{2(b_i - \bar{e}_i)}.$$

The result of Theorem 1 makes intuitive sense: as the variance in execution time demand increases, on average there will be more work that "spills over" into the next instance of T_i; moreover, as the budget b_i gets larger relative to $\bar{e}_i$, there would be more "room for error" to accommodate jobs of τ_i that run longer than average.

4 Response Time and Tardiness Bounds for Multiprocessor Scheduling

A number of global scheduling algorithms, such as GEDF, can schedule the system of *servers* T on a multiprocessor with bounded tardiness, under the mild conditions that

$$b_i \leq p_i \, \forall i, \text{ and}$$
$$\sum_{i=1}^{n} b_i/p_i \leq m,$$

where m is the number of processors. In particular, any global scheduling algorithm with *window-constrained priorities* has bounded tardiness under these two conditions [8]. Let A be such a global scheduling algorithm. Then the system T of servers can be scheduled using A on a multiprocessor with bounded tardiness, i.e.,

$$\exists B(T, A) = \{B_1(T, A), B_2(T, A), \ldots, B_n(T, A)\}$$

such that the tardiness of an arbitrary instance $T_{i,j}$ (i.e., the amount of time by which $T_{i,j}$ exceeds its deadline) is at most $B_i(T, A)$. For example, when $m \geq 2$,

$$B_i(T, \text{GEDF}) = \frac{\sum\limits_{T_k \in E_{\max}} b_k - b_{\min}}{m - \sum\limits_{T_k \in U_{\max}} \frac{b_k}{p_k}} + b_i, \tag{10}$$

where $E_{\max}$ is the set of $m - 1$ servers with largest execution budgets, $U_{\max}$ is the set of $m - 1$ servers with largest utilization (b_i/p_i), and $b_{\min}$ is the smallest budget of any server (note that somewhat tighter bounds for GEDF have been proved [4, 5]). In the special case where $m = 1$, earliest-deadline-first (EDF) is optimal, and $B_i(T, \text{EDF}) = 0$. Note that we do not claim that GEDF is the best way to schedule the server instances in every case; it is simply an example of a scheduling algorithm that could be used.

Lemma 9. *When T is scheduled on a multiprocessor using algorithm A, the completion time of $T_{i,\lceil(j)+1+W_{i,j}/b_i\rceil}$ is no later than $D_{i,(j)+1} + \lceil W_{i,j}/b_i \rceil p_i + B_i(T, A)$.*

Proof. In the case where $W_{i,j} = 0$, the lemma simply states that $T_{i,(j)+1}$ completes by time $D_{i,(j)+1} + B_i(T, A)$, which is true by definition.

In the case where $W_{i,j} > 0$, we have $R_{i,(j)+2} = p_i + R_{i,(j)+1}$ (because T_i is eligible and backlogged), and successive instances of T_i are released p_i time units apart until all the remaining work of τ_i, including the work in $W_{i,j}$, is completed. Therefore, $T_{i,\lceil(j)+1+W_{i,j}/b_i\rceil}$ is released $\lceil W_{i,j}/b_i + 1 \rceil p_i$ time units after $T_{i,(j)+1}$ is, i.e., at time $R_{i,(j)+1} + (\lceil W_{i,j}/b_i \rceil) p_i$.

Because T is scheduled using an algorithm with bounded tardiness, $T_{i,\lceil(j)+1+W_{i,j}/b_i\rceil}$ completes no later than its deadline plus $B_i(T, A)$. Since its deadline is p_i time units later than its release time, the instance completes no later than time $D_{i,(j)+1} + \lceil W_{i,j}/b_i \rceil p_i + B_i(T, A)$. □

Lemma 10. *When $W_{i,j} > 0$, $D_{i,(j)+1} < t_{i,j} + 2p_i$.*

Proof. $W_{i,j} > 0$ means that some work of τ_i with deadline at most $t_{i,j}$ remains when $T_{i,(j)}$ completes. Hence, by the replenishment rule, $T_{i,(j)+1}$ is replenished at time $R_{i,(j)} + p_i$ (as soon as it is eligible). Then, because server deadlines are implicit, $D_{i,(j)+1} = R_{i,(j)+1} + p_i = R_{i,(j)} + 2p_i$. By the definition of (j), $R_{i,(j)} < t_{i,j}$, so the result follows. □

Lemmas 9 and 10 lead to the three main results of this section. Theorem 2 gives an upper bound on the actual response time of a job given that all the execution demands are known; Theorem 3 gives an upper bound on the actual response time of a job under Assumption D; Theorem 4 (the major result of this paper) gives an upper bound on the expected response time of a job under Assumption S.

The first result is a response time bound, expressed in terms of the remaining work; that is, if we calculate the values of $\{W_{i,j}, j = 1, 2, \ldots\}$, which requires knowing the values of $\{X_{i,j}, j = 1, 2, \ldots\}$, we can calculate an upper bound on the response time of any job of τ_i.

Theorem 2. *If τ_i runs on the server task T_i and T is scheduled on a multiprocessor using algorithm A, which has bounded tardiness, then the response time of any job released in $[s_{i,j}, t_{i,j})$ is less than*

$$\left(\left\lceil \frac{W_{i,j}}{b_i} \right\rceil + 3 \right) p_i + B_i(T, A).$$

Proof. From Lemma 1, we know that any job of τ_i with deadline no later than $t_{i,j}$ completes no later than the actual time at which the server instance $T_{i,(j)+1+\lceil W_{i,j}/b_i \rceil}$ completes. Lemma 9 tells us that $T_{i,(j)+1+\lceil W_{i,j}/b_i \rceil}$ finishes no later than time $D_{i,(j)+1} + \lceil W_{i,j}/b_i \rceil p_i$, which by Lemma 10 is less than $t_{i,j} + 2p_i + \lceil W_{i,j}/b_i \rceil p_i$. Because any job released in $[s_{i,j}, t_{i,j})$ was released no earlier than $s_{i,j}$, and because $t_{i,j} - s_{i,j} \leq p_i$, the result follows. □

The result of Theorem 2 is not particularly useful in a practical sense unless all execution times are known. However, we can use Assumption D, Assumption S, and the result of Theorem 1 to obtain two useful response time bounds.

Theorem 3. *Suppose τ satisfies Assumption D. If there exists a set of budgets $\{b_1, b_2, \ldots, b_n\}$ such that $e_i \leq b_i$ for all $\tau_i \in \tau$, and such that T is multiprocessor schedulable using some algorithm A that ensures bounded tardiness, and if τ_i runs on the server task T_i, then the response time of any job of τ_i released during $[s_{i,j}, t_{i,j})$ is less than*

$$3p_i + B_i(T, A).$$

Proof. This theorem follows immediately from Theorem 2 and the observation that in (2), if Assumption D holds, then $X_{i,j} \leq e_i$ and $W_{i,j} = 0$ for all j. □

Theorem 4. *Suppose τ satisfies Assumption S. If there exists a set of budgets $\{b_1, b_2, \dots, b_n\}$ such that $\bar{e}_i < b_i$ for all $\tau_i \in \tau$, and such that T is multiprocessor schedulable using some algorithm A that ensures bounded tardiness, and if τ_i runs on the server task T_i, then the expected response time of any job of τ_i released during $[s_{i,j}, t_{i,j})$ is less than*

$$\left(\frac{\sigma_i^2}{2b_i(b_i - \bar{e}_i)} + 4\right) p_i + B_i(T, A).$$

Moreover, the q-quantile of the response time distribution is less than

$$\left(\frac{\sigma_i^2}{2b_i(b_i - \bar{e}_i)(1-q)} + 4\right) p_i + B_i(T, A).$$

Proof. We simply take the expected value of the response time bound given in Theorem 2, using the result of Theorem 1. Hence, the expected response time of $\tau_{i,j}$ is less than

$$\begin{aligned}
&\mathsf{E}\left(\left(\left\lceil \frac{W_{i,j}}{b_i} \right\rceil + 3\right) p_i + B_i(T, A)\right) \\
\leq &\mathsf{E}\left(\left(\frac{W_{i,j}}{b_i} + 4\right) p_i\right) + B_i(T, A) \\
= &\left(\frac{\mathsf{E}\left(W_{i,j}\right)}{b_i} + 4\right) p_i + B_i(T, A) \\
\leq &\left(\frac{\sigma_i^2}{2b_i(b_i - \bar{e}_i)} + 4\right) p_i + B_i(T, A).
\end{aligned}$$

Note that while we can remove $B_i(T, A)$ from the expected value because it is deterministic, we cannot bring the expected value inside the ceiling operator because the ceiling operator is non-linear, so we must upper bound it by adding one.

To prove the quantile result, let $\mathcal{R}_{i,j}$ denote the response time of $\tau_{i,j}$, and let $\mathcal{R}_q$ be the q-quantile of the response time distribution of task i; that is, $\mathsf{P}\left(\mathcal{R}_{i,j} \leq \mathcal{R}_q\right) = q$. Then,

$$\begin{aligned}
&\mathsf{P}\left(\mathcal{R}_{i,j} \leq \left(\frac{\sigma_i^2}{2b_i(b_i - \bar{e}_i)(1-q)} + 4\right) p_i + B_i(T, A)\right) \\
\geq &\mathsf{P}\left(\left(\frac{W_{i,j}}{b_i} + 4\right) p_i + B_i(T, A) \leq \right. \\
&\qquad \left.\left(\frac{\sigma_i^2}{2b_i(b_i - \bar{e}_i)(1-q)} + 4\right) p_i + B_i(T, A)\right) \\
= &\mathsf{P}\left(W_{i,j} \leq \frac{\sigma_i^2}{2(b_i - \bar{e}_i)(1-q)}\right),
\end{aligned}$$

where the inequality comes from upper bounding R using Theorem 2 and the equality follows from cancelation. Now, Markov's inequality [16, p. 400] states that for any non-negative random variable Y, and for any $y > 0$,

$$\mathsf{P}\left(Y \leq y\right) \geq 1 - \frac{\mathsf{E}\left(Y\right)}{y}.$$

Using this result and Theorem 1

$$\begin{aligned}
&\mathsf{P}\left(W_{i,j} \leq \frac{\sigma_i^2}{2(b_i - \bar{e}_i)(1-q)}\right) \\
\geq &1 - \frac{\sigma_i^2}{2(b_i - \bar{e}_i)} \cdot \frac{2(b_i - \bar{e}_i)(1-q)}{\sigma_i^2} \\
= &q.
\end{aligned}$$

Combining these results, we conclude that

$$\mathsf{P}\left(\mathcal{R}_{i,j} \leq \left(\frac{\sigma_i^2}{2b_i(b_i - \bar{e}_i)(1-q)} + 4\right) p_i + B_i(T, A)\right) \geq q,$$

and hence

$$\mathcal{R}_q \leq \left(\frac{\sigma_i^2}{2b_i(b_i - \bar{e}_i)(1-q)} + 4\right) p_i + B_i(T, A).$$

□

In the expected response time bound of Theorem 4, the term $B_i(T, A)$ will not include worst-case execution times, because T consists of servers with defined execution budgets $\{b_i, i = 1, 2, \dots, n\}$ that depend on $\bar{e}_i$. In fact, worst-case execution costs never enter into our analysis in Theorem 4. This is a huge practical advantage: even if worst-case execution costs exist (which is not required), we do not have to know them in order to compute the bound. The bound depends on the distribution of execution times only through mean and variance.

Probabilistic Deadlines. The quantile result of Theorem 4 also implicitly gives a sufficient condition for checking whether the analyzed system meets a timing requirement specified as a *probabilistic deadline*. A probabilistic deadline is a pair (δ_i, η_i), and is satisfied if the probability that a job of τ_i has a response time exceeding δ_i is no more than η_i [1].

Corollary 1. *If*

$$\left(\frac{\sigma_i^2}{2b_i(b_i - \bar{e}_i)\eta_i} + 4\right) p_i + B_i(T, A) \leq \delta_i,$$

then τ_i meets probabilistic deadline (δ_i, η_i).

Corollary 1 follows by simply observing that if the $(1 - \eta_i)-$quantile is no more than δ_i, then the probabilistic deadline holds because the response time will exceed δ_i with probability at most η_i.

Improvements for Sporadic Stochastic Systems. In some special cases, we can tighten the bounds given in this section.

Corollary 2. *Suppose τ is a sporadic stochastic task system with implicit deadlines (that is, p_i is also the relative deadline of all jobs of τ_i). Then the expected tardiness of any job of τ_i is less than*

$$\left(\frac{\sigma_i^2}{2b_i(b_i - \bar{e}_i)} + 3\right) p_i + B_i(T, A).$$

Corollary 2 follows immediately from Theorem 4 by subtracting the relative deadline from the response time bound.

Remark. In the special case where τ is a system where all the demand from jobs of τ_i in the interval $[s_{i,j}, t_{i,j})$ occurs exactly at $s_{i,j}$, for all j, the analysis can be improved by p_i; that is, the bound given by Theorem 2 is becomes

$$\left(\left\lceil \frac{W_{i,j}}{b_i} \right\rceil + 2\right) p_i + B_i(T, A).$$

Hence, the corresponding response time bounds given by Theorems 3 and 4, and the tardiness bound given by Corollary 2 are also decreased by p_i. The quantile result proved in Theorem 4 and used in Corollary 1 can be improved similarly. This special case encompasses, for example, the sporadic task model and the sporadic stochastic task model used as examples in Section 2. We omit the proof of this result due to space considerations; however, it follows the same analysis, with $T_{i,(j)}$ replacing $T_{i,(j)+1}$ throughout.

5 Discussion

In this section, we discuss several considerations necessary for the implementation of our scheduling approach. In particular, we address the selection of execution budgets and window sizes (i.e., length of $[s_{i,j}, t_{i,j})$ intervals), and we compare the expected tardiness bounds derived in this paper to those in [13].

5.1 Execution Budgets

In the analysis, we assumed that there exist budgets $\{b_i, i = 1, 2, \ldots, n\}$ such that $\bar{e}_i < b_i$ for all i; however, there is certainly not a unique choice of such values. The bounds given in Theorems 2–4 depend on the values of $\{b_i, i = 1, 2, \ldots, n\}$ in a nonlinear way. Moreover, if A is a global algorithm with window-constrained priorities, as defined in [8], then as seen in (10), $B_i(T, A)$ depends on $\{b_i, i = 1, 2, \ldots, n\}$ in a way that is nonlinear, because $B_i(T, A)$ depends on the $m - 1$ or $m - 2$ largest values in $\{b_i, i = 1, 2, \ldots, n\}$. For these algorithms there is little hope for finding an optimal choice of $\{b_i, i = 1, 2, \ldots, n\}$ to minimize the tardiness bound. Hence, we will have to rely on a heuristic approach to determine execution budgets. For simplicity, we consider only sporadic stochastic task systems.

Proportional Execution Heuristic. A simple heuristic for assigning server task execution times uses the proposition that a server's execution budget should be proportional to its assigned sporadic stochastic task's average execution time. To determine the server execution budgets, we simply let $b_i = \min\{p_i, \alpha\bar{e}_i\}$ for some

$$1 < \alpha \leq \frac{m}{\bar{u}_{\text{sum}}}.$$

The first inequality is necessary because we must have $b_i > \bar{e}_i$ for all i for expected tardiness to be finite, and the second inequality is sufficient to guarantee $\sum_i b_i/p_i \leq m$, which is needed for T to be schedulable.

The proportional execution heuristic is simple, but it does not account for why we even need to have $b_i > \bar{e}_i$ in the first place—variance in execution times. If a certain sporadic stochastic task has little variation in its execution time from one job to the next, then its corresponding server task may need to have an execution time that is only slightly larger than what the sporadic stochastic task requires on average; on the other hand, another sporadic stochastic task whose execution times vary widely from job to job will require a server task with longer execution times in order to have a low tardiness bound. This intuition leads us to the variance-based heuristic.

Variance-Based Heuristic. To determine server execution budgets, set $b_i = \min\{p_i, \bar{e}_i + \beta\sigma_i\}$, with

$$0 < \beta \leq \frac{m - \bar{u}_{\text{sum}}}{\sum_i \frac{\sigma_i}{p_i}}.$$

Here, the first inequality is necessary because we must have $b_i > \bar{e}_i$ for all i for the expected tardiness to be finite, and the second inequality is sufficient to guarantee $\sum_i b_i/p_i \leq m$, which is needed for T to be schedulable.

To see why the two different heuristics might be useful for different types of task systems, we examine the impact of the variability of execution times on the expected tardiness bounds for the uniprocessor case ($m = 1$). Specifically, we look at the relationship between the mean and variance of execution times, treating other parameters as if they are constant. If we examine the bound given in Theorem 4, we notice that with the variance-based heuristic, assuming $\bar{e}_i + \beta\sigma_i \leq p_i$, the expected response time bound given in Theorem 4 is

$$\left(\frac{\sigma_i^2}{2(\bar{e}_i + \beta\sigma_i)(\bar{e}_i + \beta\sigma_i - \bar{e}_i)} + 2\right) p_i,$$

which reduces to

$$\left(\frac{1}{2\beta(\frac{\bar{e}_i}{\sigma_i} + \beta)} + 2\right) p_i.$$

Therefore, if we treat p_i as a constant, the size of this bound will be on the order of $\frac{\sigma_i}{\bar{e}_i}$. This ratio is known as the *coefficient of variation*, or c.v., of the execution times.

On the other hand, with the proportional execution heuristic, assuming $\alpha\bar{e}_i \leq p_i$, we have

$$\left(\frac{\sigma_i^2}{2\alpha\bar{e}_i(\alpha\bar{e}_i - \bar{e}_i)} + 2\right) p_i,$$

which reduces to

$$\left(\frac{\sigma_i^2}{2\alpha\bar{e}_i^2(\alpha - 1)} + 2\right) p_i.$$

If we treat p_i as a constant, the size of this bound will be on the order of $\frac{\sigma_i^2}{\bar{e}_i^2}$, or the square of the c.v.

This analysis suggests that when the c.v. is less than one (indicating low variability in execution times), the proportional execution heuristic is likely to perform better; on the other hand, when the coefficient of variation is more than one (indicating high variability in execution times), the variance-based heuristic is likely to perform better. As a basis for comparison, the exponential distribution has a c.v. of one. These conclusions fit with our intuition that as variance gets larger, it would be more important to include information about the variance in determining server budgets.

In the multiprocessor case, these results are less clear: $B_i(T, A)$ will often depend on $\{b_i, i = 1, 2, \ldots, n\}$ in ways that are hard to analyze. Nonetheless, intuition leads us to believe that it may still be useful to have more than one heuristic for determining $\{b_i, i = 1, 2, \ldots, n\}$.

5.2 Window Sizes

As we noted in Section 2, SRT-SED tasks can include certain types of dependence. In particular, any job of a task that is released within a given window $[s_{i,j}, t_{i,j})$ (for any $j \geq 1$) can have an arbitrary level of dependence on the execution time of another job of the same task released in the same window. However, it cannot depend on the execution time of any job released outside its window.

Although we assumed independence in the stochastic execution time demands between windows, the size of the window during which dependence is allowed can be specified. Unfortunately, specifying larger window sizes would result in a larger value of p_i, and hence a larger expected tardiness bound. Therefore, there is an analytical tradeoff between allowing dependence in longer intervals and having increased tardiness bounds.

In a real application, it is unlikely that successive jobs of a task will be perfectly independent of one another. Therefore, the user must make a qualitative decision about window sizes based on the application at hand. For example, in work with colleagues on MPEG video decoding in Linux, we found that while successive frames had clearly dependent decoding times, as the window size was increased, the correlation between demands in successive windows decreased, and when a window size of at least 50–100 frames was used, no significant correlation was observed [7]. Therefore, in such an application, the system designer will have to decide between using a smaller window size and accepting that the tardiness bound may not be as accurate due to violation of the assumptions, or using a larger window size and having a better timing guarantee. In the MPEG example, the better timing guarantee would potentially come at the expense of additional memory required to store decoded frames.

Table 2: Example Sporadic Stochastic Task System τ.

Task	p_i	$\bar{e}_i$	WCET	σ_i^2	$\bar{u}_i$
τ_1	4	3	25	1	0.75
τ_2	4	3	20	1	0.75
τ_3	5	3	30	4	0.60
τ_4	5	3	20	1	0.60
τ_5	8	2	15	1	0.25
τ_6	20	3	35	2	0.15
τ_7	20	2	25	1	0.10

Table 3: Expected tardiness bounds from this paper and [13] for tasks from Table 2. Server budgets based on proportional execution heuristic with $\alpha = 1.25$ (the largest value possible for T to be schedulable).

Task	b_i	$B_i(T,$ GEDF)	Expected Tardiness	Expected Tardiness [13]
τ_1	3.75	10.11	18.82	93.88
τ_2	3.75	10.11	18.82	88.88
τ_3	3.75	10.11	23.67	98.88
τ_4	3.75	10.11	21.00	88.88
τ_5	2.50	8.86	28.06	83.88
τ_6	3.75	10.11	57.22	103.88
τ_7	2.50	8.86	56.86	93.88

5.3 Comparison to Prior Work

In [13], the presented analysis was illustrated by considering an example system consisting of seven tasks. Since the SRT-SED task model developed in this paper generalizes the task model considered in [13], we can analyze the same system using the analysis presented herein. We consider this example system here, in order to provide a sense of the improvements the new analysis enables.

Consider the sporadic stochastic task system in Table 2 on a four-core platform. This same example was used to illustrate the results in [13]. Each task has a worst-case execution time greater than its period, so it would be considered unschedulable by [4, 8]. However, since the total expected utilization is 3.2, which is less than 4.0, and each task's expected utilization is less than one, τ is schedulable as a sporadic stochastic task system. Note that, in this example, worst-case execution times are roughly an order of magnitude greater than average-case times. In reality, such differences could be much more extreme.

The results of applying Theorem 4 are shown in Tables 3 and 4 for $\{b_i\}$ values determined with the proportional execution heuristic and the variance-based heuristic, respectively, with GEDF as the scheduling algorithm for T. These tardiness bounds are clearly superior to those given in [13], which are also shown in Tables 3 and 4. Under execution budgets assigned by the variance-based heuristic, the task with the largest expected tardiness, τ_6, is guaranteed not to miss its deadline by more than 56.67 time units on average; in [13], *no task* in τ could be proved to have an expected tardiness bound of less than 68.88 time units, *plus* its worst-case exe-

Table 4: Expected tardiness bounds from this paper and [13] for tasks from Table 2. Server budgets based on variance-based heuristic with $\beta = 0.59$ (the largest value possible for T to be schedulable.)

Task	b_i	$B_i(T,$ GEDF)	Expected Tardiness	Expected Tardiness [13]
τ_1	3.21	10.17	19.12	93.88
τ_2	3.21	10.17	19.12	88.88
τ_3	6.21	10.76	22.79	98.88
τ_4	3.21	10.17	21.35	88.88
τ_5	2.48	9.17	27.79	83.88
τ_6	4.21	10.42	56.67	103.88
τ_7	2.48	9.17	55.72	93.88

cution time. This contrast emphasizes the point that the expected tardiness bounds in [13] are on the order of worst-case execution times, while the expected tardiness bounds given by Theorem 4 are on the order of average execution times.

6 Conclusion

We considered a scheduling policy where a general class of tasks, defined as *soft real-time with stochastic execution demands*, or SRT-SED, are scheduled on simple sporadic servers, with predetermined execution budgets. The SRT-SED model generalizes the sporadic task model, the multiframe task model, and the task model used in [13] by allowing for dependence within intervals of bounded length. We derived expected tardiness bounds on a multiprocessor assuming that servers are scheduled using a global algorithm with bounded tardiness. The expected tardiness bounds in this paper improve on those given in [13] for the special case where each job's execution time is independent of every other job's execution time, because the bounds presented here do not depend on worst-case execution costs; indeed, such worst-case execution costs need not even exist. This is a major distinction in comparison to prior work.

The major analytical contribution of this paper was to separate the deterministic scheduling analysis, which is done with the server budgets, from the stochastic analysis. In other words, the terms $B_i(T, A)$ do not depend on the stochastic analysis but on existing deterministic analysis. Because of this separation, if better deterministic tardiness analysis becomes available in the future, then it can be used immediately with the results of this paper to yield better expected tardiness bounds.

A major practical implication of this paper for scheduling soft real-time systems is that the parameters needed to schedule the task system and determine a tardiness bound (mean and variance of execution times) can be easily estimated from observational data in an unbiased way. This reduces the need to perform timing analysis for tasks where bounded tardiness is acceptable (e.g., multimedia decoding). Such tasks can effectively be executed on simple sporadic servers in a predictable way.

References

[1] L. Abeni and G. Buttazzo. QoS guarantee using probabilistic deadlines. In *Proceedings of the IEEE Euromicro Conference on Real-Time Systems*, June 1999.

[2] S. Baruah, J. Goossens, and G. Lipari. Implementing constant-bandwidth servers upon multiprocessor platforms. In *Proceedings of the 8th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2002.

[3] S. A. Brandt, S. Banachowski, C. Lin, and T. Bisson. Dynamic integrated scheduling of hard real-time, soft real-time and non-real-time processes. In *Proceedings of the 24th IEEE Real-Time Systems Symposium*, 2003.

[4] U. C. Devi and J. H. Anderson. Tardiness bounds under global EDF scheduling on a multiprocessor. In *Proceedings of the 26th IEEE Real-Time Systems Symposium*, 2005.

[5] J. Erickson, S. Baruah, and U. C. Devi. Improved tardiness bounds for global EDF. In *Proceedings of the EuroMicro Conference on Real-Time Systems (ECRTS)*, 2010.

[6] D. P. Heyman and M. J. Sobel. *Stochastic Models in Operations Research*, volume 1. McGraw-Hill, 1982.

[7] C. J. Kenna, J. L. Herman, B. B. Brandenburg, A. F. Mills, and J. H. Anderson. Soft real-time on multiprocessors: Are analysis-based schedulers really worth it? Manuscript available at `http://www.cs.unc.edu/~anderson/papers.html`, 2011.

[8] H. Leontyev and J. H. Anderson. Generalized tardiness bounds for global multiprocessor scheduling. In *Proceedings of the 28th IEEE Real-Time Systems Symposium*, 2007.

[9] H. Leontyev and J. H. Anderson. A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees. *Real-Time Systems*, 43(1), 2009.

[10] D. V. Lindley. The theory of queues with a single server. *Mathematical Proceedings of the Cambridge Philosophical Society*, 48(2), 1952.

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

[12] S. Meyn and R. L. Tweedie. *Markov Chains and Stochastic Stability*. Cambridge University Press, 2009.

[13] A. F. Mills and J. H. Anderson. A stochastic framework for multiprocessor soft real time scheduling. In *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2010.

[14] A. Mok and D. Chen. A multiframe model for real-time tasks. In *Proceedings of the 17th IEEE Real-Time Systems Symposium*, 1996.

[15] R. Pellizzoni and M. Caccamo. M-CASH: a real-time resource reclaiming algorithm for multiprocessor platforms. *Real-Time Systems*, 40:117–147, 2008.

[16] S. Ross. *A first course in probability*. Prentice Hall, 6 edition, 2002.