Independence Thresholds: Balancing Tractability and Practicality in Soft Real-Time Stochastic Analysis*

Rui Liu+, Alex F. Mills++, and James H. Anderson+

+Department of Computer Science, University of North Carolina at Chapel Hill
++Department of Operations and Decision Technologies, Indiana University Kelley School of Business

Abstract

The issue of stochastic response-time analysis is considered in the context of soft real-time multiprocessor schedulers. For such analysis to yield tractable, closed-form results, it is inevitably necessary to assume that execution times are probabilistically independent. However, stochastic dependencies among tasks are often common in actual systems. To enable closed-form analysis results to be applied to such systems, the concept of an independence threshold *is introduced. Such a threshold is a "tunable" per-task parameter that can be adjusted to control the extent of dependency in task execution times as assumed in analysis; such thresholds can even be applied in settings where explicit dependencies exist among tasks through resource sharing. A method is presented for setting independence thresholds in which measured task execution times are subjected to known statistical independence tests. This method is applied in a case study involving MPEG decoding. In this case study, the usage of independence thresholds enabled up to a 3.5-fold reduction in provisioned task execution times compared to a worst-case provisioning without compromising analysis assumptions.*

1 Introduction

The multicore revolution has led to considerable scheduling-related work pertaining to multiprocessor real-time systems. The vast majority of such work has focused on ensuring hard real-time (HRT) deadline constraints. Unfortunately, applying such work in practice has proven to be somewhat problematic due to various complexities that arise in multicore-based systems. For example, cross-core interactions through shared hardware such as buses, caches, and memory controllers can be difficult to handle efficiently. In contrast, such complexities are much less of a concern when supporting soft real-time (SRT) applications for which less stringent timing guarantees are needed. Thus, focusing on the SRT case may yield more immediate practical benefits.

One major difference between HRT and SRT applications is that the former must necessarily be provisioned on a worst-case basis, while for the latter, an average- or near-average-case provisioning may likely be more suitable. Unfortunately, obtaining tractable, closed-form schedulability-analysis results in the latter case almost always entails assuming that execution times are probabilistically independent,[1] and such assumptions are often violated in practice. Thus, we have the following dilemma: while shifting focus from the HRT case to the SRT case obviates many of the complexities affecting multicore-based real-time systems, such a shift introduces new analysis limitations related to independence assumptions. In this paper, we address this dilemma by presenting a new approach for easing such limitations.

Related work. Before describing our main contributions, we first present a brief review of prior work on SRT multiprocessor scheduling and stochastic schedulability analysis. Due to space limitations, this overview emphasizes research of direct relevance to the work presented herein.

Edgar and Burns [10] were perhaps the first to apply statistical analysis to the determination of task execution times. Their work, which focused on *worst-case execution times* (*WCETs*), was later refined by Burns et al. [6] and by Bernat et al. [2], who proposed the idea of *probabilistic worst-case execution times* (*pWCETs*), for which independence can be assumed. These authors also showed that pWCETs can be empirically determined. In contrast to this prior work, which focuses on worst-case provisionings for HRT systems, our main focus here is enabling average- or near-average-case provisionings for SRT systems. Many later studies were conducted involving the estimation of pWCETs (e.g., [8, 12]). Additionally, Maxim et al. [15] presented uniprocessor fixed-priority response-time analysis for tasks provisioned via pWCETs that have probabilistically determined minimum inter-arrival times.

Our work builds directly upon prior work of Mills and Anderson [16, 17], who presented stochastic analysis for bounding expected task response times. Their work built upon prior work on global multiprocessor schedulers that showed that a variety of such schedulers are capable of ensuring bounded deadline tardiness (and hence bounded response times) with-

*Work supported by NSF grants CNS 1016954, CNS 1115284, CNS 1218693, and CNS 1239135.

[1]Throughout this paper, the term *independence* is used to refer to probabilistic independence. We emphasize this because this term usually has a different meaning in the real-time scheduling literature.

out schedulability-related utilization loss [9, 14]. In this prior tardiness-related work, WCETs were used. This leads to a worst-case system provisioning, which is problematic for many SRT systems due to the pessimism involved. Mills and Anderson showed that by encapsulating each real-time task within a deterministically provisioned server that is ensured bounded tardiness, expected task tardiness can be upper-bounded by the sum of two terms: a deterministic component pertaining to server tardiness, and a stochastic component pertaining to the task's execution using its server's budget allocations, which can be provisioned based on the encapsulated task's average execution time [17]. The analysis yielding this result does not require actual execution-time distributions; rather, a task's execution time is characterized by specifying a mean and standard deviation, both of which can be determined through measurement. While this work was motivated by prior work on global scheduling, it can be applied in any context where server response times are bounded. A similar server-based approach was later used by Palopoli et al. [18] in work that focused on computing deadline-miss probabilities.

As with most stochastic analysis, that of Mills and Anderson requires independence assumptions. To provide some leeway in this regard, they allow the jobs (i.e., invocations) of the same task to be grouped into *windows*, with one or more jobs per window, and provide response-time bounds on a per-window basis. Dependencies are allowed within such windows, but not across windows. Highly dependent jobs can be dealt with by defining such windows to be large, but since response times are bounded on a per-window basis, such bounds become large as well. As discussed next, we present an alternative way of dealing with dependencies in this paper that does not suffer from this shortcoming.

Independence thresholds. When applying stochastic analysis with independence assumptions to predict the behavior of an actual system, one is essentially making a "bet" that such assumptions are not egregiously violated in practice. Applying the concept of an "independence threshold" as proposed herein increases the odds that such a bet will pay off. An *independence threshold* is nothing more than a per-task deterministic execution-cost parameter. A task's actual execution cost is modeled in analysis as the sum of two costs: a random cost that is bounded from above by the deterministic independence threshold (for which independence is not assumed) and an additional random cost for which independence is assumed. Such a modeling approach has two benefits. First, system designers can control the extent of any *assumed* independence. Second, if extreme worst-case behavior is rare, then it may be possible to set a task's independence threshold close to a much lower average-case value without compromising analysis assumptions. Thus, a wasteful worst-case system provisioning can be potentially avoided.

This concept is illustrated in Fig. 1, which shows a sequence of measured execution times for a task and three possible independence thresholds for that task. By setting the threshold to $A = 0$ ms, the system designer is assuming that all execution times for this task are independent. At the other extreme, a setting of $C = 7$ ms corresponds to a deterministic worst-case provisioning for this task (provided the measurement process revealed the true worst case), so no independence is required. Between these two extremes, a setting of $B = 3$ ms requires independence to be assumed only for the portion of this task's execution time that exceeds 3 ms. These three choices represent three different levels of confidence the system designer may have concerning the independence of task execution times.

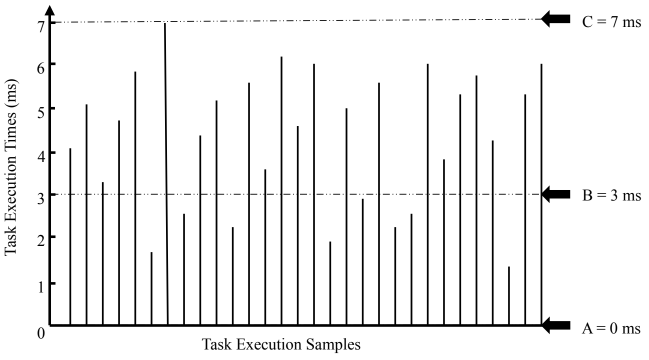

Figure 1: Three different independence thresholds.

Contributions. Our contributions are fourfold. First, we introduce the notion of an independence threshold as a simple and elegant way of balancing the conflicting desires of having tractable stochastic response-time analysis and being able to provision systems without resorting to worst-case or near-worst-case assumptions. Second, we show how to integrate such thresholds in the server-based stochastic analysis of Mills and Anderson [17]; such an integration can even be adapted to allow explicit resource sharing among tasks (i.e., critical sections). To our knowledge, the issue of resource sharing in the context of stochastically provisioned SRT systems has not been considered before. Third, while an independence threshold can be viewed as a tunable design parameter, we present a measurement process, based on known statistical independence tests, that can be applied to set such thresholds so that observed runtime dependencies are effectively eliminated. Fourth, we present a case study involving MPEG decoding tasks in which these results were applied. In this case study, the usage of independence thresholds enabled up to a 3.5-fold reduction in provisioned task execution times without compromising analysis assumptions.

Organization. After providing needed background (Sec. 2), we more formally introduce the concept of an independence threshold (Sec. 3), present the above-mentioned response-time analysis, measurement process, and experimental results (Secs. 4–6), and then conclude (Sec. 7).

2 Background

In this section, we introduce the task-system model and scheduling scheme we use. The majority of this section is adapted from [17].

2.1 Stochastic Sporadic Tasks

We consider a system $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ consisting of n tasks scheduled on m processors. A *task* is a possibly infinite sequence of jobs. *Jobs* are demands for processing time. Jobs of the same task must execute sequentially and in FIFO order. Each task $\tau_i \in \tau$ is a sporadic task specified with a period p_i and a relative deadline d_i, where $d_i \leq p_i$. The j^{th} job of τ_i, $\tau_{i,j}$, has a release time $r_{i,j}$, absolute deadline $d_{i,j} = r_{i,j} + d_i$, and execution time $X_{i,j}$, which is a random variable. If $\tau_{i,j}$ completes execution at time t, then its *response time* is $t - r_{i,j}$, and its *tardiness* is $\max\{0, t - d_{i,j}\}$. (If a job is tardy, then the release time of its successor is unaltered.) For now, we assume that tasks share no resources other than the processors upon which they are scheduled; however, we eliminate this assumption later in Sec. 3.

2.2 Simple Sporadic Servers

The scheduling scheme considered in this paper involves a server abstraction. Each task $\tau_i \in \tau$ runs on a unique simple sporadic server T_i, which carefully controls the amount of time that τ_i is allowed to execute.

Let $T = \{T_1, T_2, \ldots, T_n\}$ be a system of simple sporadic servers. Each such server T_i has period p_i and execution budget b_i. Permissible values of b_i are specified later in Cor. 1 in Sec. 3. 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 is also the period of τ_i). T_i is *backlogged* if and only if there is pending work of τ_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 global earliest-deadline-first (GEDF) scheduler is used, then the budget of T_i is consumed whenever it has one of the m earliest deadlines.

Note that server T_i always has a fixed budget b_i, while τ_i's execution times are random variables $\{X_{i,j}, j = 1, 2, \ldots\}$. The manner in which the server T_i schedules its contained task τ_i is explained next.

2.3 Scheduling Example

Fig. 2(b) shows an example with two 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. Job $\tau_{1,1}$ is released at time 0 and has execution cost 4, job $\tau_{1,2}$ is released at time 6.3 and has execution cost 1.5, and job $\tau_{1,3}$ is released at time 11.3 and has execution cost 2. Job $\tau_{2,1}$ is released at time 0 and has execution cost 0.8, and job $\tau_{2,2}$ is released at time 3 and has execution cost 1.7. The schedule is not shown after time 13.

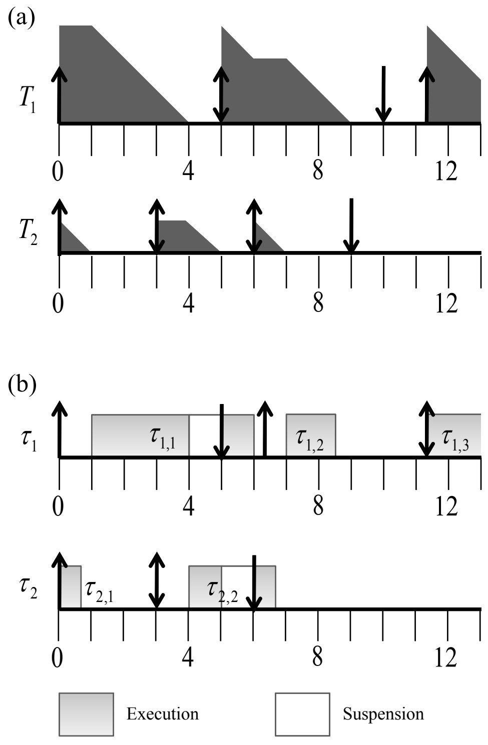

Figure 2: Example servers and sporadic stochastic tasks on a uniprocessor. **(a)** For servers T_1 and T_2, $\uparrow$ denotes replenishment time and $\downarrow$ denotes deadline; the budget is shaded for each server. T_1 and T_2 are scheduled using EDF. **(b)** For the sporadic stochastic tasks τ_1 and τ_2, $\uparrow$ denotes release time and $\downarrow$ denotes deadline; actual execution times are shaded, and suspensions are shown in white.

Fig. 2(a) shows budget changes for two servers, scheduled via the uniprocessor earliest-deadline-first (EDF) scheduler. T_1 corresponds to τ_1. It has period 5 (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 Fig. 2 to illustrate some important properties of the considered scheduling approach.

Initial replenishment. Both servers are replenished at time 0 since they are both eligible (having never been replenished before) and backlogged (since 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 time 0.8, T_2 continues its consumption, even though this means that the processor is idle. This prevents server instances from experiencing self-suspensions (which can complicate schedulability analysis).

Task suspensions and resumptions. 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. A server's budget is replenished only when it is eligible *and* backlogged. Thus, its next replenishment may be after its next 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 then.

The server scheme described above is used in this paper because of its simplicity. However, our results are also applicable under a variety of other server schemes that use more complex budget management rules.

3 Independence Thresholds

The *independence threshold* of task τ_i is a non-negative fixed parameter h_i. For a job $\tau_{i,j}$ of τ_i, the portion of its execution time exceeding h_i is denoted by $Y_{i,j} = \max\{0, X_{i,j} - h_i\}$. In order to apply the response-time analysis of Mills and Anderson [17], we assume the existence of a sequence of non-negative independent and identically distributed (i.i.d.) random variables $E_{i,j}$, where $j = 1, 2, \ldots$, with mean $\overline{e}_i$ and variance σ_i^2, where $\mathsf{P}(E_{i,j} \geq Y_{i,j}) = 1$. Moreover, we assume that, for any $l \neq i$, $E_{i,j}$ and $E_{l,k}$ are independent. Note that these independence assumptions apply only to the $E_{i,j}$ random variables and do not require either the actual job execution times or the $Y_{i,j}$ random variables to be independent of one another. An illustration of our notation is given in Fig. 3. As discussed later in Sec. 5.2, we upper-bound the $Y_{i,j}$ random variables by the $E_{i,j}$ random variables because the former may fail to be independent, due to the fact there are two cases in the expression $\max\{0, X_{i,j} - h_i\}$.

In applying the response-time analysis reviewed below, we use $E_{i,j} + h_i$ as the execution time of $\tau_{i,j}$ instead of $X_{i,j}$. Thus, $\tau_{i,j}$'s execution time consists of a deterministic part h_i and a stochastic part $E_{i,j}$. In general, analysis pessimism will be reduced if h_i can be set as low as possible. The desire to support the existence of independent $E_{i,j}$ random variables is the key determining factor when selecting an appropriate value for h_i. Later, in Sec. 5, we describe a measurement process that can be used to set h_i and determine $\overline{e}_i$ and σ_i^2, which are needed in the response-time analysis.

4 Response-Time Analysis

Suppose that a scheduling algorithm A (such as GEDF) is used to schedule the server system T that ensures bounded tardiness, i.e., for each server T_i, there exists a constant $B_i(T, A)$ such that any arbitrary instance $T_{i,j}$ has tardiness at most $B_i(T, A)$. Also, let $Z_{i,1}, Z_{i,2}, \ldots$ be i.i.d. random variables, where $Z_{i,j}$ upper bounds the execution time of job $\tau_{i,j}$, and $Z_{i,j}$ and $Z_{l,k}$ are independent for any $l \neq i$. Then, the expected response time of each stochastic sporadic task τ_i can be bounded by using Cor. 2 in [17]. Letting $\overline{Z}_i$ denote the mean of the sequence $Z_{i,1}, Z_{i,2}, \ldots$ and letting $Var(Z_i)$ denote its variance, this corollary is as follows, with minor modifications due to notational differences.

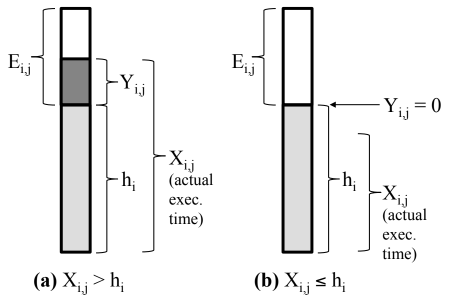

Figure 3: Depiction of our notation for the cases **(a)** $X_{i,j} > h_i$ and **(b)** $X_{i,j} \leq h_i$.

Corollary 1. *If there exists a set of budgets* $\{b_i, b_2, \ldots, b_n\}$ *such that* $b_i > \overline{Z}_i$ *for all* $\tau_i \in \tau$, *and* T *is scheduled on a multiprocessor platform with algorithm* A *as described above, then the expected response time of any job* $\tau_{i,j}$ *is less than*

$$\left(\frac{Var(Z_i)}{2b_i(b_i - \overline{Z}_i)} + 3 \right) p_i + B_i(T, A). \tag{1}$$

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

$$\left(\frac{Var(Z_i)}{2b_i(b_i - \overline{Z}_i)(1-q)} + 3 \right) p_i + B_i(T, A). \tag{2}$$

The quantile result is important because it allows a system designer to determine a bound that will hold any desired percentage of time (e.g., the 0.9-quantile is a threshold that will be met in at least 90% of cases). From a practical standpoint, this allows the more conservative user to introduce additional pessimism by specifying a higher quantile.

In both (1) and (2), the first summand accounts for tardiness that arises due to task τ_i's stochastically varying execution times; the second summand, $B_i(T, A)$, is due to the (deterministically bounded) tardiness experienced by the server T_i. Note that we cannot replace $Z_{i,j}$ by $X_{i,j}$ because independence is not assumed of the $X_{i,j}$ variables. However, we can replace it by $E_{i,j} + h_i$ because the $E_{i,j}$ variables do satisfy the given independence assumptions, h_i is a constant, and $E_{i,j} + h_i \geq X_{i,j}$. Given that h_i is a constant, it follows that $\overline{Z}_i = \overline{e}_i + h_i$, and $Var(Z_i) = \sigma_i^2$. Using these substitutions, we can bound the expected response time of any job of task τ_i using (1), and we can bound the q-quantile of the response-time distribution using (2).

Allowing job windowing. Cor. 1 is derived from an expected response-time bound obtained for a more general task model considered in [17] in which each task's jobs are partitioned into fixed-length windows that may contain multiple jobs each, and response times are bounded on a per-window basis; Cor. 1 considers the special case where each window includes exactly one job. As noted in Sec. 1, the window-based model used in [17] allows dependencies among jobs within windows (but not across windows). This gives us two methods for potentially dealing with dependencies: the usage of independence thresholds as proposed here, and the usage of job windows as proposed in [17]. Both methods can in fact be used together, though we do not state a counterpart of Cor. 1 for this possibility due to space constraints. In Sec. 6, we experimentally evaluate both methods.

Allowing resource sharing. The introduction of lock-based resource sharing can cause dependencies in two ways. First, jobs may experience *blocking times* when they wait to access shared resources. Second, information exchanged through such resources may cause execution-time variations, e.g., by influencing the code paths a job takes. Under suspension-oblivious blocking analysis [5] (wherein blocking time is analytically viewed as computation time), dependencies due to blocking can be eliminated by always provisioning critical sections (CSs) and the blocking times they induce on a worst-case basis. In actual systems, care is often taken to ensure that CS lengths are short, in which case accounting for them on a worst-case basis should have only minor impact. Execution-time variations outside of CSs can be dealt with by applying the idea of an independence threshold to the non-CS portion of a task's execution, but properly setting such a threshold becomes more complicated in this case, as discussed in Sec. 5.4. The general idea of dealing with CS-related costs on a worst-case basis and non-CS execution times stochastically is not limited to using independence thresholds. For example, the same idea can be applied when job windowing is used. We revisit the issue of resource sharing later in the context of our experiments.

5 Measurement Framework

An independence threshold is a tunable parameter that can be set to any non-negative value. However, for the analysis described in the prior section to be valid, the independence assumptions given in Sec. 3 should be respected. In this section, we explain how to use known statistical tests to ensure that these assumptions are respected (at least, to the extent possible using known tests). For simplicity, we assume here that the windowing technique mentioned in the prior section is not being used (or more precisely, each window includes exactly one job) and that critical sections do not exist. However, the methods discussed here can be easily extended to properly set independence thresholds when jobs are grouped into windows (in which case, a task's independence threshold would be set by considering per-window execution times) and can be adapted to apply when critical sections exist (though the latter can introduce some technical challenges with respect to measurement, as discussed in Sec. 5.4).

We assume that independence thresholds are set by applying statistical tests to measured task execution times. The statistical tests used in this section are *hypothesis tests*. Each tries to reject a certain *null hypothesis* $\mathcal{H}_0$ and has an associated *significance level*, which is a fixed probability of wrongly rejecting $\mathcal{H}_0$, if it is in fact true. We need tests for validating both independence and the property of being identically distributed. We consider these issues separately.

5.1 Testing for Independence

The fundamental idea of probabilistic independence between two random variables is that if information about the realized value of one random variable is known, then this knowledge should not change the distribution of the other random variable (in other words, knowing the value of one random variable is not "useful" in predicting the value of the other random variable). It is desirable that tests for independence be "distribution-free": such tests can be applied to reason about data for which the underlying distribution is unknown. For a sequence of random variables, we can perform a distribution-free test for independence by applying the *runs test*, which has two variants, as discussed next. For both variants, the null hypothesis $\mathcal{H}_0$ states that the values in the tested sequence are independent of one another. This hypothesis is tested by exploiting certain statistical properties of runs.

Runs of "above and below the mean." Consider a sequence of observations from two complementary and fixed-probability events a and b. That is, $\mathsf{P}(a)$ and $\mathsf{P}(b)$ are fixed for all observations and $\mathsf{P}(a) + \mathsf{P}(b) = 1$. A *run* is defined as an unbroken (sub)sequence of the same event. For example, in the sequence

$$a\ a\ a\ a\ b\ b\ a\ a\ b\ b\ a\ a\ a\ b\ a\ b\ a\ a\ a,$$

there are nine runs: five runs of a, namely, $a\ a\ a\ a$, $a\ a$, $a\ a\ a$, a, and $a\ a\ a$, and four runs of b, namely, $b\ b$, $b\ b$, b, and b. As discussed in [3], for a sequence of a's and b's that is drawn independently, the total number of a-runs and b-runs follows a known distribution, which motivates defining a test statistic for randomness based on the number of runs. Specifically, let n_a be the number of a's, n_b be the number of b's, and R denote the total number of runs in the sequence under consideration. For example, in the sequence above, $n_a = 5$, $n_b = 4$, and $R = 9$. Following [3], *if the sequence consists of independent variables*, then R approximately follows a Normal distribution with mean μ_R and variance σ_R^2, given by

$$\mu_R = \frac{2n_a n_b}{n_a + n_b} + 1 \quad \text{and} \quad \sigma_R^2 = \frac{2n_a n_b(2n_a n_b - n_a - n_b)}{(n_a + n_b)^2(n_a + n_b - 1)}.$$

Knowing this, to test the hypothesis that the variables in the sequence are independent, we can compute the test statistic

$$\frac{R_{\text{observed}} - \mu_R}{\sqrt{\sigma_R^2}},$$

where R_{observed} is the number of observed runs in the tested sequence. Using a standard Normal table, we can use this test statistic to compute a P-value. Generally speaking, in statistical tests, a *P-value* is the probability of observing such a test statistic if the null hypothesis is true (in this case, if the variables are independent). The choice of 0.05 is commonly used as a significance level, meaning the assumption of independence will be rejected if the P-value is less than 0.05.

The runs test can be generalized to apply to a sequence of more than two events. For example, suppose that we draw 16 integers from the range $\{0, 1, \ldots, 9\}$ and get the sequence

$$3\ 8\ 2\ 0\ 1\ 2\ 3\ 4\ 5\ 4\ 6\ 2\ 9\ 1\ 3\ 4,$$

which has mean 3.5625. We can replace each digit in this sequence by a "+" if its value is at least the mean and a "-" if its value is less than the mean. This transforms the sequence to the "binary" sequence

$$- + - - - - - - + + + + - + - - + .$$

From here, we can treat the binary sequence as observations of constant events and follow the testing procedure discussed above. Due to its simplicity, this approach has been applied previously to test the independence of sample data [19].

Runs of "ups and downs." Another widely adopted runs-based independence test involves redefining the definition of a run to be "an unbroken sequence of increasing or decreasing observations" [3]. Under this alternative definition, there are two events that are observed from a sample: a value is greater than its predecessor (an "up"), and a value is at most its predecessor (a "down"). An unbroken sequence of increasing values is an "up-run," and an unbroken sequence of decreasing values is a "down-run." Let R denote the total number of up-runs and down-runs. Similarly to before, R can be used to define a test statistic for independence. From [3], if the sample size is n, then R approximately follows a Normal distribution with mean μ_R and variance σ_R^2, which are given by

$$\mu_R = (2n-1)/3 \quad \text{and} \quad \sigma_R^2 = (16n-29)/90.$$

As before, we can calculate the test statistic $\frac{R_{\text{observed}} - \mu_R}{\sqrt{\sigma_R^2}}$ in order to determine a P-value, and a low P-value (generally less than 0.05) results in rejecting the independence assumption.

Although they have different definitions of a run, both runs tests use "the number of runs" to test for independence. In general, "too few runs" indicates a tendency for high and low values to cluster, while "too many runs" indicates a tendency for high and low values to alternate. These two different approaches have different underlying assumptions (and have been misapplied in the literature). For the "above/below" runs test, the probability that a value in a sample is at most the mean of the sample is fixed across the sample. However, in the "up/down" runs test, the probability that a sample value is greater than or at most the previous value is not fixed. Thus, when applying these two tests, caution is needed to choose the right form of the mean and variance of the number of runs. In general, the up/down runs test is the preferable variant due to its more sophisticated usage of data. This test has been used previously to test for independence in task execution times [8].

Rank von Neumann ratio test. A more sophisticated version of the runs test takes into account the magnitude of the differences, rather than just the direction (i.e., up/down), by examining the *rank* of each value (where rank 1 is the smallest value and rank n is the largest value). This so-called *rank von Neumann ratio test* has more power to detect dependence than the runs tests, making it ideal for small samples, but its P-value cannot be determined using the standard Normal distribution [1]. For this reason, we do not implement it in this paper, although we discuss the issue of *power* later.

5.2 Applying Independence Tests to Determine Independence Thresholds

When applying independence thresholds to Cor. 1, it would be desirable to define a job $\tau_{i,j}$'s execution time as

$$h_i + Y_{i,j}, \tag{3}$$

where $Y_{i,j}$ is $\max\{0, X_{i,j} - h_i\}$ (recall Sec. 3). Any dependence among such execution times would arise due to the second summand (as the first summand is a constant). Unfortunately, any independence assumption would likely be rejected when defining execution times in this way. In particular, the second summand in (3) may be 0 for many jobs, and the ordering of these 0's among non-zero values may not appear random. However, as discussed in Sec. 4, we only need to find a sequence of random variables $E_{i,j}$, which upper bounds $Y_{i,j}$. To accomplish this, our measurement framework involves subjecting the subsequence of the sequence $Y_{i,1}, Y_{i,2}, Y_{i,3}, \ldots$ consisting of only *positive* values to an independence test. We then assume in applying Cor. 1 that each job $\tau_{i,j}$ for which $Y_{i,j} = 0$ has an execution time of $h_i + E_{i,j}$, where $E_{i,j}$ is drawn from the same distribution as the random variables in the tested (positive-only) subsequence. That is, in defining execution times, we allow the term $Y_{i,j}$, when 0, to be replaced by a positive value. This introduces some pessimism, but such a substitution is valid because Cor. 1 only provides an upper bound on response times.

5.3 Testing for Identical Distribution

The *two-sample Kolmogorov-Smirnov test* (*KS-Test2*) determines if two samples are identically distributed by evaluating the difference between their empirical cumulative distri-

```
$t_1 \leftarrow \max\{x_{i,1}, x_{i,2}, x_{i,3}, \ldots, x_{i,N}\}$
$t_0 \leftarrow \min\{x_{i,1}, x_{i,2}, x_{i,3}, \ldots, x_{i,N}\}$
$t \leftarrow (t_0 + t_1)/2$
$x'_i \leftarrow \{\max\{0, x - t\} : x \in x_i\}$
$x^+_i \leftarrow \{x \in x'_i : x > 0\}$
while true do
    if $x^+_i$ passes both the runs test and KS-Test2 then
        if $t - t_0 < 0.01$ then
            return $h_i \leftarrow t$
        else
            $t_1 \leftarrow t$
            $t \leftarrow (t_1 + t_0)/2$
            Update $x'_i$ and $x^+_i$ accordingly
        end if
    else
        $t_0 \leftarrow t$
        $t \leftarrow (t_1 + t_2)/2$
        Update $x'_i$ and $x^+_i$ accordingly
    end if
end while
```

Figure 4: Measurement procedure for setting independence thresholds. Either runs test can be used.

bution functions [7]. To test if a data set is identically distributed, we randomly select two samples of equal size from the data set and apply KS-Test2 to these two samples. This "selection-compare" process is performed multiple times for multiple sample sizes. If we use a significance level of 0.05 in KS-Test2, then it will reject one out of 20 pairs of samples on average even if they are identically distributed. To avoid such over-rejecting, we apply the Bonferroni correction, which uses a significance level of $0.05/k$ when examining k pairs of samples [20]. For instance, for a data set of size 10,000, we could select two random samples of size 500, 1,000, 2,000, and 5,000 respectively, and then apply KS-Test2 to each pair of samples of the same size using a significance level of 0.0125. The data set under testing can be considered identically distributed if all pairs of samples pass KS-Test2.

5.4 Measurement Process for Independence Thresholds

Let $x_i = x_{i,1}, x_{i,2}, x_{i,3}, \ldots, x_{i,N}$ be a sequence of N sample execution times of task τ_i. To find the lowest suitable independence threshold for τ_i, we perform a binary-search-like operation as shown in Fig. 4.

The procedure in Fig. 4 keeps t_0 as a lower bound for h_i and updates its value as the search proceeds. The search terminates when the difference between the measured h_i and t_0 is less than 0.01. Recall that a lower value of h_i means that the system designer has more confidence in the independence of τ_i's execution times, and such a lower value will yield a less pessimistic response-time bound. This procedure returns the lowest value of h_i for a given precision. In addition, it does not require the existence of τ_i's WCET.

Limitations of statistical tests. The statistical tests introduced in this section involve examining the validity of the null hypothesis. For example, the runs test is based on the null hypothesis that the sequence being tested is independent. Informally speaking, the test looks for evidence through the number of runs to prove that the null hypothesis is wrong. If the runs test cannot find enough evidence (the number of runs is neither too many nor too few), then it will fail to reject the independence assumption. If the data set is small and/or the level of dependence is weak, then the runs test could accept the sequence as independent only because there is not sufficient data to show otherwise. The ability of a test to correctly reject the null hypothesis is known as *power*. Thus, these tests must be applied with care. Generally speaking, it is desirable to have as large a sample as possible to increase the power of the test.

Note that, while using budgeted servers temporally isolates different tasks to a large degree, our measurement procedure for setting independence thresholds does not assess dependencies across tasks but only within the jobs of the same task. Dependencies across tasks could arise either implicitly (e.g., through contention involving shared hardware such as caches) or explicitly (e.g., through shared data structures). The implicit case can be dealt with by using a stress-inducing workload in the measurement process that creates contention for hardware resources and thus gives conservative estimates (e.g., using a methodology similar to that used in [4]). The explicit case is trickier and would involve properly accounting for how explicit interactions influence per-job execution times, e.g., due to variations in code paths taken. This is a complicated issue that we defer for further study.

Despite these limitations, recall that the focus here is the provisioning of SRT systems with less stringent timing requirements than HRT systems. Moreover, we are attempting to provide guidance and reduce assumed risks in provisioning such systems. We are not claiming that all risks can be eliminated. For our purposes, the tests we have introduced would likely give acceptable results in many practical scenarios.

6 MPEG Case Study

In this section, we illustrate the application of our results by presenting the results of a case study we conducted involving MPEG decoding. We selected MPEG as a case study for several reasons. First, video decoding times vary from frame to frame with MPEG. Second, decoding times are higher for scenes with significant movement. Thus, a frame is more likely to take a long time to decode if its predecessor took a long time to decode, i.e., per-frame decoding times are not independent. Third, with MPEG decoding there is a significant difference between the worst-case per-frame decoding time and the average case. This motivates wanting to avoid a worst-case provisioning.

Experimental setup. A system of 12 tasks was considered in our experiments, where each task decodes a separate high-definition movie trailer. (A similar workload was considered in [13].) The hardware platform used to trace frame decoding times has two six-core Xeon X5650 processors (running

Table 1: Computed independence thresholds and other deduced information for 12 MPEG decoding tasks (all times are in ms).

Video	h_i	$\bar{e}_i$	σ_i^2	$h_i + \bar{e}_i$	WCET
1	29.06	5.35	43.23	34.41	64.10
2	17.54	6.71	34.38	24.25	67.90
3	31.23	3.71	14.28	34.94	52.49
4	22.62	4.60	17.48	27.22	45.38
5	26.63	5.36	41.35	31.99	78.33
6	16.20	3.34	7.05	19.54	39.76
7	17.63	7.18	49.41	24.81	71.47
8	25.60	3.32	8.11	28.92	37.32
9	17.71	4.19	8.50	21.90	38.70
10	12.63	1.42	1.33	14.05	19.09
11	24.72	4.90	29.38	29.62	59.29
12	12.81	6.54	35.96	19.35	73.03

at 2.67 GHz), giving the system 12 cores in total. Each processor has a 12 MB L3 cache shared among its six cores. The considered platform has a NUMA (nonuniform memory access) memory architecture, with one 12 GB memory module per processor. Each job of each task decodes one frame of video. In determining per-job execution times, tasks were run in isolation and all videos were preloaded into memory to avoid page faults. (In applying our measurement procedure, such choices are left to the system designer.) The videos used in this case study all have a constant frame rate of 23.98 frames per second (fps). Thus, each decoding task has a period of $1/(23.98 \text{ fps}) \approx 41.70$ms.

Independence threshold selection. For each video decoding task, we determined the lowest independence threshold possible (without job windowing) using the proposed measurement procedure. We used the "up/down" runs test to test for independence. The typical movie trailer contains approximately 3,200 frames, which translates to a sample size of approximately 3,200 as well. Table 1 shows the resulting values obtained for h_i, as well as $\bar{e}_i$ and σ_i^2, for each video. The table also lists the resulting average-case provisioning, as given by $h_i + \bar{e}_i$, and the observed WCET for each video. Observe that the provisioned execution-time reduction afforded by the usage of independence thresholds is considerable. On average across all videos, a two-fold reduction was enabled, without violating the independence assumptions used in our response-time analysis. The greatest reduction was by a factor of approximately 3.5.

Budget selection. We now examine expected response-time bounds that can be computed for our system of decoding tasks. Such bounds will depend on how server budgets are determined. According to Cor. 1, if the server T_i can be scheduled with bounded tardiness, then the expected response time of τ_i will be bounded, provided T_i's budget is selected to satisfy $b_i > \overline{Z}_i$, which is equivalent to $b_i > h_i + \bar{e}_i$. This leaves some leeway in budget selection; the actual value selected for b_i will impact the expected response-time bound that results. Mills and Anderson [17] proposed two heuristics for assigning b_i values:

- *Proportional Execution Heuristic*: Here, b_i is set to $\min\{p_i, \alpha(\bar{e}_i + h_i)\}$, where α is a parameter satisfying
$$1 < \alpha \leq \frac{mp_i}{\bar{e}_i + h_i}.$$
- *Variance-Based Heuristic*: Here, b_i is set to $\min\{p_i, \bar{e}_i + h_i + \beta\sigma_i\}$, where σ_i is the standard deviation of $E_{i,j}$ and β is a parameter satisfying
$$0 < \beta \leq \frac{m - \sum_{i=1}^{n} \frac{\bar{e}_i + h_i}{p_i}}{\sum_{i=1}^{n} \frac{\sigma_i}{p_i}}. \quad (4)$$

The ranges of α and β ensure that $b_i > \bar{e}_i + h_i$ and $\sum_{i=1}^{n} b_i/p_i \leq m$, which is required for the server system T to have bounded tardiness on m processors.

As pointed out in [17], the proportional execution heuristic performs better when the variance of $E_{i,j}$ is small, while the variance-based heuristic is preferable when there is high variability in $E_{i,j}$.

Scheduling algorithm. Response-time bounds also depend on the scheduling algorithm used to schedule servers. A number of global scheduling algorithms, such as GEDF, can schedule the system of servers on a multiprocessor with bounded tardiness, under the mild conditions that

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

where m is the number of processors. For example, when $m \geq 2$, GEDF ensures the tardiness bound

$$B_i(T, \text{GEDF}) = \frac{\sum\limits_{T_k \in E_{\max}} b_k - b_{\min}}{m - \sum\limits_{T_k \in U_{\max}} \frac{b_k}{p_k}} + b_i, \quad (5)$$

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 utilizations (b_i/p_i), and $b_{\min}$ is the smallest budget of any server [9]. For simplicity, we consider the usage of GEDF here and the above tardiness bound, even though global algorithms with better tardiness bounds exist and better bounds for GEDF are known [11]. (These better algorithms and bounds are more complicated to explain, and the algorithm and bound we are assuming are sufficient to illustrate our analysis.) We also assume that the considered tasks are scheduled on $m = 11$ of the cores on our test system (in our test system, one core is dedicated to handling interrupts). Decoding times exhibit a large variance, so we used the variance-based heuristic in determining server budgets, giving β the largest value allowed by (4). The server budgets and expected response-time bounds (obtained from (5) and (1)) that result from these

Table 2: Server Budgets and expected response time bounds for 12 MPEG decoding tasks (all times are in ms).

Video	b_i	Response Time Bound
1	41.70	391.70
2	40.04	388.20
3	41.70	389.79
4	38.48	386.35
5	41.70	390.86
6	26.69	374.49
7	41.70	390.19
8	36.59	384.22
9	29.75	377.54
10	17.16	364.71
11	41.70	389.95
12	35.50	383.84

choices are given in Table 2. The response-time bounds can be used to provision output queues by comparing against the common per-task period of 41.70 ms.[2]

Comparison with a worst-case provisioning. As seen in Table 1, under a worst-case provisioning many of the decoding tasks are not schedulable, as their WCETs exceed their periods. In contrast, as seen in Table 2, by provisioning the system based on independence thresholds, all tasks are schedulable and all expected response times are bounded.

Independence threshold selection assuming job windows of size three. Following the same process as described above, we also determined independence thresholds assuming job windowing is used as suggested in [17], with three jobs per window. Recall that windowing is an alternative technique for dealing with dependencies—jobs within the same window may have dependencies. When windowing is used, all measurements and analyses are done on a per-window basis. Effectively, each such window has a period of 41.70 ms $\times$ $3 = 125.10$ ms. Comparing Tables 1 and 3, we can see that using windows of size three results in lower independence thresholds. In comparing these two tables, it is important to note that the thresholds in Table 3 apply to groups of three jobs, while those in Table 1 apply to individual jobs. Said another way, a period three times larger is assumed for Table 3, which impacts expected utilization. For example, from the data in Table 1, task τ_1 would have an expected utilization of $34.41/41.70 \approx 0.825$, while from the data in Table 3, its expected utilization would be $49.53/125.10 \approx 0.396$. Because the number of windows is now one third of the total number of frames, it is not clear whether these improvements come from a real decrease in execution-time dependencies or from having an insufficient number of sample values when applying the tests discussed in Sec. 5. Corresponding response-time bounds for the windowing case are given in Table 4. Comparing to Table 2, we see the price to be paid for windowing: expected response times are now bounded on a per-window basis and hence are much larger.

[2]Response-time bounds can be greatly reduced by using a scheduler like *global fair lateness* [11] with better tardiness bounds than GEDF to schedule servers, or by using better server tardiness analysis than covered here.

Table 3: Independence thresholds and other computed information for 12 MPEG decoding tasks assuming three jobs per window (all times are in ms).

Video	h_i	$\bar{e}_i$	σ_i^2	$h_i + \bar{e}_i$	WCET
1	14.94	34.59	255.40	49.53	135.03
2	53.50	17.27	14.28	70.77	114.97
3	16.34	37.15	401.10	53.49	110.43
4	0	47.51	239.58	47.51	109.58
5	43.38	14.11	172.97	57.49	123.65
6	30.94	17.47	77.81	48.41	77.82
7	0	53.82	356.57	53.82	148.34
8	63.56	4.00	18.38	67.56	80.93
9	70.57	5.07	20.90	75.64	100.06
10	24.60	11.41	24.94	36.01	49.57
11	42.25	15.06	185.24	57.31	150.93
12	39.91	14.05	129.22	54.96	113.73

Table 4: Server Budgets and expected response time bounds for 12 MPEG decoding tasks assuming three jobs per window (all times are in ms).

Video	b_i	Response Time Bound
1	125.10	1098.87
2	90.50	1063.08
3	125.10	1099.98
4	125.10	1098.73
5	125.10	1098.46
6	94.47	1067.67
7	125.10	1099.68
8	89.94	1062.60
9	99.51	1072.14
10	62.08	1035.13
11	125.10	1098.55
12	111.31	1086.60

Allowing resource sharing. As discussed in Sec. 4, we can allow lock-based resource sharing (under suspension-oblivious analysis) in our model by treating all critical-section-related costs (i.e., critical section execution times plus blocking times) on a *worst-case* basis. In our MPEG case study, each job of each task actually adds its decoded frame to a lock-protected queue before terminating. Since the lock associated with this queue is acquired at the end of a job's execution, the introduction of lock-based sharing in this case cannot alter code paths taken within jobs (thus, most of the complexities associated with resource-sharing noted at the end of Sec. 5.4 do not arise). For the jobs under consideration, we found the worst-case per-job critical-section-related cost to be approximately 10 μs. This cost can be accounted for by adding it to the h_i values given in Tables 1 and 3, or equivalently, by requiring the h_i values to be at least 10 μs when applying the measurement procedure. Given the magnitude of the h_i values given in these tables, we can see that such a change would have a near-negligible impact. In fact, even if critical-section-related costs were several times larger, the impact would still be small. Of course, some systems might have worst-case critical-section-related costs so large that the benefits of a stochastic provisioning are negated.

Sample sizes. We remarked at the end of Sec. 5 that a large sample size is generally desirable when applying hypothesis tests. This begs the question: what is "large"? In the context of MPEG decoding, this would require much more study and is beyond the scope of this paper. Our intent in this section was not to report on a thorough study of the provisioning of MPEG decoding tasks, but rather to illustrate how our analysis may potentially be applied using MPEG as an example. Indeed, in this particular study, sample sizes were fundamentally limited by the lengths of the trailers we used.

7 Conclusion

We considered the scheduling of SRT systems with stochastic execution demands. We proposed the notion of an independence threshold as a way of achieving a reasonable balance between the conflicting goals of having tractable stochastic analysis and avoiding a worst-case system provisioning. We also explained how to use independence thresholds along with job windowing, and how to apply them when critical sections due to resource sharing exist. To our knowledge, this is the first paper to present an approach for dealing with critical sections when stochastic analysis is applied to SRT systems.

An independence threshold is a tunable parameter that can be arbitrarily set. If set lower, the system designer is accepting more risk that independence assumptions underlying analysis might be violated in practice. As a further contribution, we presented a measurement-based framework for properly setting such thresholds based on known statistical tests. Applying this framework can give system designers a sense of the extent of the risks they are taking.

For purposes of illustration, we applied this framework and our analysis in a case study involving MPEG video decoding. In this case study, the usage of independence thresholds enabled up to a 3.5-fold reduction in provisioned task execution times compared to the worst case. Moreover, the considered tasks have resource-sharing constraints, which were found to have a near-negligible impact on analysis.

The results of this paper provide a sound framework for provisioning and analyzing SRT applications on multiprocessors. This framework is ultimately measurement-based, and thus avoids the pitfalls of static timing analysis tools that arise in the context of multprocessor machines.

In future work, we intend to refine our measurement framework by obtaining results that indicate appropriate sample sizes in various use cases. Further work is also needed to ensure that execution-time variations caused by resource sharing are properly captured.

References

[1] R. Bartels. The rank version of von Neumann's ratio test for randomness. *Journal of the American Statistical Association*, 77(377):40–46, 1982.

[2] G. Bernat, A. Burns, and M. Newby. Probabilistic timing analysis: An approach using copulas. *Journal of Embedded Computing*, 1(2):179–194, 2005.

[3] J. V. Bradley. *Distribution-Free Statistical Tests*. Prentice-Hall, 1968.

[4] B. Brandenburg. *Scheduling and Locking in Multiprocessor Real-Time Operating Systems*. PhD thesis, University of North Carolina, Chapel Hill, NC, 2011.

[5] B. Brandenburg and J. Anderson. Optimality results for multiprocessor real-time locking. *31st RTSS*, 2010.

[6] A. Burns, G. Bernat, and I. Broster. A probabilistic framework for schedulability analysis. *3rd EMSOFT*, 2003.

[7] I. Chakravarti, R. Laha, and J. Roy. *Handbook of methods of applied statistics*, volume 1. John Wiley, 1967.

[8] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzetti, E. Quinones, and F. Cazorla. Measurement-based probabilistic timing analysis for multi-path programs. *24th ECRTS*, 2012.

[9] U. Devi and J. Anderson. Tardiness bounds for global EDF scheduling on a multiprocessor. *Real-Time Systems*, 38(2):133–189, 2008.

[10] S. Edgar and A. Burns. Statistical analysis of WCET for scheduling. *22nd RTSS*, 2001.

[11] J. Erickson, B. Ward, and J. Anderson. Fair lateness scheduling: Reducing maximum lateness in G-EDF-like scheduling. *Real-Time Systems*, 50(1):5–47, 2014.

[12] J. Hansen, S. Hissam, and G. Moreno. Statistical-based wcet estimation and validation. *WCET*, 2009.

[13] C. Kenna, J. Herman, B. Brandenburg, A. Mills, and J. Anderson. Soft real-time on multiprocessors: Are analysis-based schedulers really worth it? *32nd RTSS*, 2011.

[14] H. Leontyev and J. Anderson. Generalized tardiness bounds for global multiprocessor scheduling. *Real-Time Systems*, 44(1):26–71, 2010.

[15] D. Maxim and L. Cucu-Grosjean. Response time analysis for fixed-priority tasks with multiple probabilistic parameters. *24th RTSS*, 2013.

[16] A. Mills and J. Anderson. A stochastic framework for multiprocessor soft real-time scheduling. *16th RTAS*, 2010.

[17] A. Mills and J. Anderson. A multiprocessor server-based scheduler for soft real-time tasks with stochastic execution demand. *17th RTCSA*, 2011.

[18] L. Palopoli, D. Fontanelli, N. Manica, and L. Abeni. An analytical bound for probabilistic deadlines. *24th ECRTS*, 2012.

[19] P. Radojkovic, P. Carpenter, M. Moreto, A. Ramirez, and F. Cazorla. Kernel partitioning of streaming applications: A statistical approach to an np-complete problem. *45th MICRO*, 2012.

[20] J. Shaffer. Multiple hypothesis testing. *Annual Review of Psychology*, 46(1):561–584, 1995.