The Safe and Effective Application of Probabilistic Techniques in Safety-Critical Systems

Kunal Agrawal, Sanjoy Baruah
Washington University in St. Louis

Zhishan Guo
University of Central Florida

Jing Li[*]
New Jersey Institute of Technology

ABSTRACT

The use of randomized algorithms in safety-critical systems is investigated. Under the vast majority of circumstances, randomized algorithms out-perform deterministic ones on average; however, it is not obvious how one goes about establishing the correctness of safety-critical systems that use such algorithms. The approach advocated in this work is to exploit the fact that many safety standards allow for small probabilities of failure of even the most critical functionalities. We explore the use of *concentration bounds* — probabilistic bounds on the likelihood of the performance of a randomized algorithm deviating from its expected performance — to bound the probability of failure of systems that incorporate randomized algorithms, thereby showing compliance with safety standards that allow for small probabilities of failure. We illustrate the use of the proposed approach on several examples that both explain how the approach is to be applied, and demonstrate the benefits of doing so.

KEYWORDS

safety-critical systems, safety standards, randomized algorithms, failure probabilities, concentration bounds

ACM Reference Format:
Kunal Agrawal, Sanjoy Baruah, Zhishan Guo, and Jing Li. 2020. The Safe and Effective Application of Probabilistic Techniques in Safety-Critical Systems. In *ICCAD '20: IEEE/ACM International Conference On Computer Aided Design, Nov 02–05, 2020, CA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3400302.3415674

1 INTRODUCTION

The past few decades have witnessed a burst of activity within the research community to explore the applicability of probabilistic and statistical techniques to the analysis of timing properties of safety-critical systems — see [17, 18] for surveys (citing 136 and 134 references, respectively, a large number of which are directly related to probabilistic analysis). We consider this to be a very welcome development: as safety-critical systems have gotten increasingly more complex with regards to both their software and hardware components and as the manner in which such systems interact with the environment has become increasingly more sophisticated, continued dependence upon the kinds of deterministic worst-case analysis techniques that have traditionally been used for the design and verification of software in safety-critical systems is resulting in increasingly *in*efficient implementations of such systems.

Despite the presence of this large body of research results, however, the adoption of probabilistic techniques for software development and analysis has been rather slow in industrial practice for safety-critical systems, particularly in domains that are subject to statutory certification requirements. We believe that this is due in large part to a fundamental mismatch between prior practice in these domains, which are firmly rooted in worst-case analysis, and some of the foundational notions that underpin the new probabilistic ideas and results. In this research, we seek out and explore the potential applicability of, probabilistic and statistical analysis techniques that are less at odds with current industrial practice and may, therefore, be more likely to gain acceptance.

SIL	Low demand mode prob. failure on demand	Continuous/High demand mode prob. failure per hour
1	$\geq 10^{-2}$ to $< 10^{-1}$	$\geq 10^{-6}$ to $< 10^{-5}$
2	$\geq 10^{-3}$ to $< 10^{-2}$	$\geq 10^{-7}$ to $< 10^{-6}$
3	$\geq 10^{-4}$ to $< 10^{-3}$	$\geq 10^{-8}$ to $< 10^{-7}$
4	$\geq 10^{-5}$ to $< 10^{-4}$	$\geq 10^{-9}$ to $< 10^{-8}$

Table 1: IEC 61508: Permitted Failure Probabilities

Observe that *safety standards* are one prominent place where probabilistic specifications do appear in current practice in safety-critical systems verification. Consider, for example, IEC 61508, an international standard published by the International Electrotechnical Commission titled *Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems.* This standard is generic for safety-related systems, from which several industry-specific standards (such as ISO 26262 standard, in widespread use in the automotive industry) are derived. Recognizing that zero risk can rarely be achieved in practice, IEC 61508 applies a probabilistic failure approach to account for the impact of failures. It defines Safety Integrity Levels (SILs) 1 through 4, with 1 being the lowest and 4 the highest, and associates an allowable *probability of failure* with each SIL — see Table 1. Each functionality is then assigned an SIL, and it is required to demonstrate that the probability of failure for that functionality will not exceed the allowable thresholds. Several other safety standards also similarly specify allowable failure probabilities even for safety-critical functionalities: the more safety-critical the functionality, the smaller the allowable failure probability. *Our efforts at integrating probabilistic techniques into safety-critical system design and analysis seek to leverage off this pre-existing use of the concept of probability in current safety standards.*

The system certification process typically commences well before system deployment, and must, therefore, be performed, at least initially, using *models* of the run-time behavior that is likely to be encountered by the actual system during operation. To have high

[*]All authors contributed equally to this research.

confidence that the conclusions regarding safety that are drawn on the basis of such analyses will hold for the actual system during run-time, the models of run-time behavior must be conservative: there should be widespread agreement that the models do indeed capture all relevant aspects of actual run-time behavior that may arise. Obtaining such agreement is, in addition to its technical aspects, an inherently social and cultural process in the sense that multiple interested parties — system builders, certification authorities, customers, and other stakeholders whose safety may be compromised if the system malfunctions — must, based upon evidence and argumentation, come to agree that a particular model is indeed appropriate for use in validating safety. In our opinion, this is one of the most significant barriers to the adoption of many wonderful probability-based research ideas that have been proposed: they require the acceptance of new models, that are inherently statistical or probabilistic in nature, of the run-time behavior of systems.

A cyber-physical system (CPS) may be looked upon [3] as comprising three components: (i) *software* that executes on some (ii) *computing platform*, and interacts with the (iii) *environment* within which it operates using sensors (for input) and actuators (for output). Many ideas surveyed in [17, 18] for integrating probabilistic concepts into system analysis and design advocate for the modeling of the run-time behavior of the platform, and/or of the interaction with the environment, via probabilistic models — examples include probabilistic worst-case execution time (probabilistic WCET) [8, 9] and probabilistic modeling of execution time profiles [7] (both of which assume a probabilistic model for the platform), and the stochastic modeling of job arrivals [2, 14, 31] (which is essentially a probabilistic model of the environment that gives rise to event-triggered jobs). In contrast, the approach we explore in this paper for applying probabilistic concepts in safety-critical systems analysis and design is to do so in a controlled manner, in the first component comprising a CPS — the software — *only*. Specifically, we restrict the application of probability to the explicit use of the algorithmic technique of **randomization** within specific algorithms in the software, while continuing to model the rest of the CPS — the platforms upon which these algorithms execute and the interaction with its environment — using worst-case models. We emphasize that in our proposed approach probabilistic techniques are limited to randomized algorithms: the models of input/environment assumed in the analysis of these algorithms continue to be worst-case ones, as has traditionally been the case with safety-critical systems.

Other researchers have also investigated the applicability of randomization in analyzing safety-critical systems. For instance, Davis et al. [16] explain some of the conceptual underpinnings of probabilistic WCET by means of an elegant thought-experiment involving hypothetical hardware that behaves in a time-randomized manner; this thought-experiment is repurposed in [18, Section 6] to illustrate the applicability and limitations of Extreme Value Theory (EVT) [19] to probabilistic WCET analysis. Moving beyond thought-experiments, randomized cache-replacement policies that are implemented in both hardware and software have been explored quite thoroughly in the real-time systems literature (see [18, Sections 6.1–6.2] for a survey). Most of this prior work on randomization in safety-critical systems analysis seeks to enable measurement-based characterization of run-time behavior in a probabilistic manner by intuitively making it extremely unlikely that worst-case behavior will occur and therefore increasing the likelihood that a limited number of observations will encompass the vast majority of behaviors (i.e., will witness the "tail" of the distribution of behaviors — see the discussion in [18, Section 6] concerning the use of EVT). In contrast, the approach to incorporating probability that we are proposing here does not require any additional measurement-based characterization of system behavior, but rather assumes worst-case behavior. We illustrate via a contrived (toy) example.

EXAMPLE 1. *Suppose that we were using the well-known* quicksort *algorithm [23] to sort a* 100*-element array storing values that are unknown prior to run-time, and the bottleneck operation is comparing two elements. We, therefore, wish to determine the number of comparisons needed, to provide an adequate time-budget allowing the completion of sorting. As we will see in Section 2, it is known that quicksort on a* 100*-element array may, in the worst case, require as many as* $(100 \times 99)/2 = 4950$ *comparisons; however, under the assumption that each of the* 100! *possible permutations of the input is equally likely, the average or* expected *number of comparisons is merely* 648*. Unfortunately, since the array is given at run-time, assuming that the values in it appear as a random permutation is not necessarily valid*[1]*. But a randomized version of quicksort, which randomly permutes the array prior to sorting, would indeed have the expected number of comparisons be* 648 *regardless of the input.*

The approach advocated in this paper additionally uses concentration bounds*, which are able to specify quantitative limits on the deviation of random variables from their mean values, to derive results of the form "the probability that the number of comparisons exceeds* 755 *is no larger than* 10^{-1}*;" we are able to offer this guarantee without making any assumptions whatsoever regarding the initial contents of the array. Thus, if the requirement for sorting is assigned SIL-1 according to the IEC 61508 standard and is a "low demand" functionality, we conclude from Table 1 that it suffices to assign it a time-budget that is adequate for performing* 755 *comparisons. We can similarly show that if the SIL level is 2, 3, or 4, the number of comparisons that need to be accommodated in the time-budget allocated to the sorting routine is* 860, 966, *and* 1007*, respectively. Thus, even at the highest SIL, only about a fifth (*$1007/4950 \approx 0.2034$*) of the worst-case-deterministic number of comparisons need be accommodated.* □

Contribution. The major contribution of this work is a proposed approach, centered upon the use of ***concentration bounds***, for incorporating probabilistic and statistical techniques into the analysis of safety-critical systems that, as illustrated in the example above, for certain problems is able to provide probabilistic bounds on *worst-case* behavior that are significantly better than deterministic worst-case bounds, without making any probabilistic assumptions whatsoever on the execution platform or the interaction with the environment — the only use of probability lies within randomized algorithms, and the only dependence on probabilistic phenomena is the assumed availability of a good random number generator. By not requiring any change in the mind-set of current practice in safety-critical systems validation and certification (since our use of

[1]Any such assumption, even if well-founded, needs to be further justified based on arguments regarding the actual run-time conditions under which the algorithm is executed. That is, one must argue that the run-time environment can be represented by a *model* that is not a worst-case one. As mentioned earlier, getting such models accepted by entities responsible for certification often represents a non-trivial challenge.

probabilities is entirely consistent with their use in many widely-used safety standards), we believe there is a greater likelihood of this approach being adopted by practitioners.

Organization. We organize the remainder of the paper as follows. In Sections 2–4, we explain our proposed approach and illustrate its advantages and potential wide applicability by demonstrating its use on three rather different problems — sorting unknown arrays (Section 2), provisioning data-buffers to ensure no overflow (Section 3), and parallel job scheduling (Section 4). Section 5 provides experimental evidence of the safety and effectiveness of our proposed technique on a real platform. We place our work in the context of other related work in Section 6 and conclude in Section 7.

2 RANDOMIZED QUICKSORT

Although sorting is not a particularly central problem in safety-critical systems, we start by discussing the quicksort algorithm [23] because this algorithm provides an excellent and easily-understood vehicle for introducing the relevant concepts that underpin our proposed approach for incorporating probabilistic techniques in the analysis and design of safety-critical systems. The quicksort algorithm may be briefly described in the following manner:

(1) choose some element in the array as a "pivot" element;
(2) compare each element in the array to the pivot element, to obtain three sub-arrays of elements that are respectively *(i)* smaller than, *(ii)* equal to, and *(iii)* greater than the pivot element; and
(3) recursively sort the first and third sub-arrays.

Quicksort has many desirable properties, including the fact that it can be implemented as an "in-place" sorting algorithm (only a constant amount of additional memory is needed); however, in the worst case, the number of comparisons needed is quadratic in the size of the array. This worst-case behavior occurs if the element chosen as the pivot (both initially, and in each recursive call) is the unique smallest or largest element in the array/ sub-array being sorted; if this were to happen, the total number of comparisons needed to sort an array of n elements is easily seen to be equal to

$$(n-1)+(n-2)+(n-3)+\cdots+2+1=\frac{\mathbf{n(n-1)}}{\mathbf{2}}.$$

In **randomized** quicksort, the pivot element is chosen at random, with each element in the array being equally likely to be selected; the remaining two steps of the algorithm are unchanged.[2] Although the worst-case number of comparisons remains $(n(n-1)/2)$ — this would happen if the randomly-selected pivot at each recursive call happens to be the largest or smallest element in the sub-array being sorted during that recursive call — it should be intuitively clear that this is very unlikely. Indeed, the expected (average) number of comparisons is known to be far smaller: letting random variable Q_n denote the number of comparisons made by randomized quicksort on an n-element array, there is an easily-accessible text-book proof in [13, Section 7.4.2] showing that $E[Q_n]$, the expected value of Q_n, is bounded as follows:

$$E[Q_n]<\sum_{i=1}^{n-1}\sum_{k=1}^{n}\frac{2}{k}=\sum_{i=1}^{n-1}O(\log n)=O(\mathbf{n}\log\mathbf{n})$$

[2]Note that all use of probabilistic concepts is therefore made exclusively under the control of the sorting algorithm; analysis of randomized quicksort makes only worst-case assumptions about the initial state of the array to be sorted.

A rather more detailed study of randomized quicksort [27] has yielded a tighter bound [27, p 478] on the value of $E[Q_n]$, with equality (in place of the "<" in the bound above):

$$E[Q_n]=2(n+1)H_n-4n \tag{1}$$

(Here $H_n \overset{\text{def}}{=} \sum_{i=1}^{n}(1/i)$ denotes the n'th ***harmonic number***.)

Recall our claim, in Example 1, that the expected number of comparisons performed by randomized quicksort upon a 100-element array is 648. This was obtained from Equation 1 by setting n to 100: H_{100}, the 100th harmonic number, is ≈ 5.1874, and hence we get

$$E[Q_{100}]=2\times101\times5.1874-400,\ \text{or}\ \approx\mathbf{648}.$$

In addition to the tight bound of Expression 1 above on the expected number of comparisons, a further result in [27] provides tight bounds on the probability of the number of comparisons deviating from the expected number: for any $\epsilon>0$, the probability that the actual number of comparisons performed exceeds ϵ times the expected number of comparisons, $\epsilon\times E[Q_n]$, decreases exponentially with increasing ϵ:

$$Pr\Big(\Big|Q_n-E[Q_n]\Big|\geq\epsilon\times E[Q_n]\Big)\leq n^{-(2+o(1))\epsilon\ln\ln n} \tag{2}$$

from which we conclude that

$$Pr\Big(\Big|Q_n-E[Q_n]\Big|\geq\epsilon E[Q_n]\Big)\leq n^{-2\epsilon\ln\ln n} \tag{3}$$

with the bound becoming tighter as n gets larger (since an $o(1)$-term approaches zero in value as $n\to\infty$).

For a given permissible probability of error δ, we can compute the value of ϵ needed to ensure that the RHS of Expression 3 is $\leq\delta$:

$$\epsilon\geq\frac{\ln(1/\delta)}{2\times\ln n\times\ln\ln n} \tag{4}$$

The claim we had made in Example 1, that the probability of the number of comparisons made by randomized quicksort upon an array of one hundred elements exceeding 755 is no larger than $10^{-}1$, was obtained from Expression 4 by setting $\delta = 10^{-1}$; doing so yields a value of 0.1637 for ϵ, and therefore a value of

$$(1+\epsilon)\times E[Q_{100}]\ =\ 1.1637\times648\ \approx\ \mathbf{755}$$

is a safe upper bound on the number of comparisons that need to be accommodated for SIL-1 functionalities, as claimed. The number of comparisons for SIL-2, SIL-3, and SIL-4 can be similarly obtained by setting δ in Expression 4 to 10^{-2}, 10^{-3}, and 10^{-4}, respectively.

3 BUFFER SIZING

We now examine a problem of provisioning buffers in distributed safety-critical embedded systems as our next illustrative example of the safe and effective application of probabilistic methods in safety-critical systems. Consider a node in a system that has n incoming flows, all of which must be processed at the node. We model this processing problem in the following manner:

- A total of at most one unit of flow arrives at the node (across all n flows) per unit time, and is stored in buffers (one per flow) until processed at the node.
- The node can process up to one unit of flow, *all of which is taken from a single buffer*, during each time unit.

The buffer-provisioning problem in safety-critical systems is to determine the minimum size of the buffer that must be provided for each incoming flow to *make the assurance that no data of any incoming flows are lost before being processed.* This is equivalent to determining an upper bound on the maximum *backlog* that can build up in any buffer. It was shown [1] that the node can guarantee a $\Theta(\log n)$ bound on the backlog of any flow by employing the greedy strategy of processing, at each step, one unit of flow from the incoming flow with the currently largest backlog (ties broken arbitrarily); it was also shown that this bound is tight in the sense that no (deterministic) algorithm can guarantee a $o(\log n)$ backlog. Hence, to guarantee no loss on any flow, we must provision each incoming flow with a buffer that can hold $\Theta(\log n)$ flow.

Bender et al. [6] recently proposed a randomized algorithm with a much smaller expected backlog, provided that the node is able to process a bit more than one unit of flow per time unit (equivalently, that the incoming flows sum to a bit less than one per time unit). Specifically, under the assumption that the total flow arriving across all n flows per unit time is bounded from above by $(1-\epsilon)$ for some $\epsilon \leq \frac{1}{2}$ (while the node continues to be able to process one unit of flow per unit time), they proved that the expected backlog is $\Theta(\log \log n)$ in the worst case. Note that this represents an exponential improvement over the $o(\log n)$ lower bound for deterministic algorithms, and argues strongly for the use of randomization.

For a given choice of ϵ and a selected buffer-size k, concentration bounds are also derived in [6] on the probability of the backlog exceeding k in any particular round: it is shown that this probability $F(k,\epsilon)$ is bounded from above as follows (here e denotes the base of the natural logarithms, ≈ 2.718):

$$F(k,\epsilon) = k \cdot e^{-k/3} + \frac{e^{k\epsilon^2/6}}{1 - e^{\epsilon^2/6}} \tag{5}$$

We can use the result of Expression 5 to provision buffers for functionalities that have verification/ certification requirements specified according to safety standards that permit non-zero failure probabilities. Suppose, for example, that some functionality needs to be shown to have a probability of failure no more than 10^{-3}; by choosing $\epsilon = 1/11 \approx 0.091$ (equivalently, ensuring that the node can process 10% more flow than the maximum cumulative inflow during each time unit), we may conclude that a buffer size equal to 8127 suffices (since $F(8127, 0.091) \leq 10^{-3}$). Similarly, if the error probability were required to be $\leq 10^{-6}$, then a buffer size equal to 13142 suffices. In this manner, the size of the buffer to be provisioned is determined entirely by the specified allowable failure probability: while a functionality with a specified failure probability of 10^{-6} should be provided a buffer of size 13142 on each incoming flow, it is wasteful to do so for a functionality with a failure probability of 10^{-3} (since buffers of size 8127 would suffice).

Remark. Note that the failure probability computed in Expression 5 does *not* depend on the number of incoming flows n: it depends only on the buffer size per flow and the amount by which the processing capacity of the node exceeds the incoming flow.

4 MULTICORE FEDERATED SCHEDULING

In this section, we consider a problem that arises in multicore real-time systems. We first describe the problem and its importance, and existing theoretical results. However, some important pragmatic considerations were not adequately accounted for; as a consequence, they do not always perform satisfactorily in practice. Next, we discuss some modifications that have subsequently been made to the theoretical solutions; although these modifications do significantly enhance performance under most circumstances, we show that the worst-case performance may be poorer than of the original. We then describe a randomized version of the modified solution and show how it can be integrated into the safety-critical systems.

The problem considered, and its (theoretical) solution. Due to a combination of multiple trends in real-time computing, including the increasing computational demand of safety-critical applications and the wide adoption of multicore and multiprocessor platforms, a large body of research has been performed on multiprocessor real-time scheduling (see [15] for a survey). Such research has included the development of new models that are based upon directed acyclic graphs (DAGs) ([4, 5]) for representing real-time workloads, in a manner that maximally exposes internal parallelism in the workload and thereby enables the exploitation of such parallelism by multicore scheduling algorithms. Multicore real-time scheduling algorithms have been specially designed to exploit the parallelism exposed in these models; our focus here is on the *federated* scheduling paradigm [26]. Upon systems of *constrained-deadline* and implicit-deadline[3] sporadic DAG tasks, schedulability analysis for federated scheduling reduces to the problem of determining whether each invocation of the DAG is guaranteed to complete within its deadline upon a given number of cores. The reduced problem is essentially equivalent to the widely-studied *makespan minimization* problem from traditional scheduling theory: schedule a precedence-constrained collection of jobs upon an identical multicore platform to minimize makespan. This problem has long been known [33] to be NP-hard in the strong sense. However, Graham's *list scheduling* algorithm [21], which greedily constructs a work-conserving schedule by executing at each time instant an available job, if any are present, upon any available core, performs reasonably well. It was shown [21] that list scheduling makes the following guarantee: if $\mathcal{S}_{\text{opt}}$ denotes the smallest makespan with which a particular DAG can be scheduled upon m cores, then the schedule generated by list scheduling this DAG upon m cores will have a makespan no larger than $(2 - \frac{1}{m}) \times \mathcal{S}_{\text{opt}}$. This result, in conjunction with a hardness result in [24] showing that determining a schedule for a DAG with makespan $\leq \frac{4}{3}\mathcal{S}_{\text{opt}}$ remains NP-hard in the strong sense[4], suggests that list scheduling is a reasonable algorithm to use, and most run-time scheduling algorithms that are used for scheduling DAGs use some variant of list scheduling.

An upper bound on the makespan of a schedule generated by list scheduling is easily stated. Let W denote the cumulative worst-case execution time of all the jobs (i.e., nodes) in a given DAG, and L denote the maximum cumulative execution time of any sequence of precedence-constrained jobs in the DAG. It has been shown [21] that the makespan of the schedule generated by list scheduling on m cores is guaranteed to be no more than $\frac{W}{m} + L \times \left(1 - \frac{1}{m}\right)$.

[3] See, e.g., [4, page 15] for definitions of constrained- and implicit-deadline tasks.

[4] In fact, assuming a reasonable complexity-theoretic conjecture that is somewhat stronger than P $\neq$ NP, a result of Svensson [30] implies that a polynomial-time algorithm for determining a schedule of makespan $\leq 2\mathcal{S}_{\text{opt}}$ for all m is ruled out.

An issue that arises in implementation. As seen above, list scheduling has very good theoretical properties: it runs in efficient polynomial time and provides a solution to an NP-hard problem that is no more than a fact of two of the optimal solution, and furthermore it is unlikely that any other polynomial-time algorithm will be able to provide a better approximation. From an implementation perspective, however, list scheduling suffers from a serious drawback: it requires that during run-time all jobs eligible to execute (because their predecessor jobs have completed execution) be maintained in a centralized queue, and cores that are idling are required to get a job to execute from this centralized queue instantly.[5] While this may not be an issue when the number of cores is relatively small, it has been observed that the centralized queue rapidly becomes a bottleneck resource as the number of cores increases.

Dealing with this implementation issue. The obvious solution to the problem of the bottleneck centralized queue is to replace it with a *distributed* queue: rather than keeping all eligible jobs in a single queue, multiple queues are maintained (typically, one per core). Upon completing a job and thereby rendering some other jobs eligible to execute, a core adds these newly-eligible jobs to its own queue. Similarly, upon discovering that it is idle, a core looks first at its own queue for a job to execute. If the idled core discovers that its own queue is empty, then the second idea behind the solution — *work stealing* — comes into play: the core visits the queues of other cores one by one, and "steals" a job from the first non-empty queue (if there is any) it encountered.

Distributed queues with work-stealing have been observed to perform very well in practice; however (as with quicksort) one can construct pathological cases where its performance would actually be *poorer* than when using a single centralized queue. For example, if all eligible jobs always end up placed in the same queue, from which all the other cores must repeatedly steal — this queue becomes a bottleneck as in centralized queueing, and we additionally have to pay the overhead of maintaining multiple queues and failed steal attempts of visiting cores with empty queues. (See [20, Theorem 19] for a formalization and details of this argument.) To make such pathological worst-case behavior unlikely, work-stealing is usually implemented in a randomized manner in the sense that a core that needs to steal a job chooses from the other cores' queues uniformly at random. This version — randomized work-stealing across distributed queues — has been widely used in industrial parallel runtime systems. This approach has additionally been adapted for use in real-time systems [25], but with the caveat that its use is restricted to soft-real-time systems: "$\cdots$ *[randomized] work-stealing may not be suitable for hard real-time tasks*" [25, page 203].

In the remainder of this paper, we use previously-derived concentration bounds for makespan under randomized work-stealing [32] to demonstrate that scheduling strategies based on randomized work-stealing can, in fact, be used in safety-critical systems; we further validate this assertion via extensive experiments. Via our proposed usage of concentration bounds, federated scheduling with randomized work-stealing becomes compliant with the current standards that require the guarantee of not exceeding the allowable probability of failure in the safety-critical systems. In contrast, previous work [25] can only bound the *expected response time* and cannot provide any probabilistic worst-case guarantee.

Let us define the constant Φ as follows and $e \approx 2.718$:

$$\Phi = 2\Big/\left(1 - \log_2\left(1 + \frac{1}{e}\right)\right) \tag{6}$$

Note that the value of Φ is a bit less than 3.65.

Let random variable S_{rws} denote the makespan of the schedule generated when a DAG with cumulative WCET W and WCET of the longest chain of precedence-constrained jobs L is scheduled by a randomized work-stealing scheduler. It was proved in [32] that $E[S_{\text{rws}}]$, the expected value of S_{rws}, is bounded as follows:

$$E[S_{\text{rws}}] \le \left(\frac{W}{m} + \Phi \times L + 1\right) \tag{7}$$

Comparing with the worst-case makespan bound of list scheduling, this expected makespan is, in fact, larger: this is the price we pay for choosing to implement a distributed queue rather than a centralized one. (Recall that this choice was necessitated by the need to not have a centralized queue be a potential bottleneck during run-time.)

Since expected values are not always enough for safety-critical systems: we also need concentration bounds specifying probabilities on deviation from the expected values. Such a concentration bound for makespan was derived in [32]:

$$P\left(S_{\text{rws}} \ge \left(\frac{W}{m} + \Phi \times L + 1\right) + \Phi \times \log_2 \frac{1}{\epsilon}\right) \le \epsilon \tag{8}$$

This essentially states that for any $\epsilon > 0$, the probability that the makespan of the schedule would exceed the expected value in Expression 7 by an amount $\ge (\Phi \times \log_2 \frac{1}{\epsilon})$ is no greater than ϵ.

How should we be using this concentration bound in the design and implementation of safety-critical systems? We first illustrate via an example and subsequently detail how this concentration bound can be used to obtain a resource-efficient implementation.

Example 2. *Suppose that we have a constrained-deadline DAG task with the following parameters: cumulative WCET $W = 150ms$, maximum cumulative WCET of any sequence of precedence-constrained jobs $L = 9ms$, and relative deadline $D = 68ms$, which is to be scheduled using the federated scheduling algorithm [26] upon a dedicated set of cores. Let us assume, as we did in Example 1, that the functionality of this task is characterized according to IEC 61508 (for which the permitted failure probabilities are specified in Table 1). If it were characterized as a low-demand SIL-2 task, then we see from Table 1 that the permitted failure probability is 10^{-2}. Substituting this value for ϵ in Expression 8 above, we have*

$$\begin{aligned} &P\left(S_{\text{rws}} \ge \frac{W}{m} + \Phi \times L + 1 + \Phi \times \log_2 100\right) \le 10^{-2} \\ \Leftarrow\ &P\left(S_{\text{rws}} \ge \frac{150}{m} + 3.65 \times 9 + 1 + 3.65 \times 6.644\right) \le 10^{-2} \\ \Leftrightarrow\ &P\left(S_{\text{rws}} \ge \frac{150}{m} + 58.09\right) \le 10^{-1} \end{aligned}$$

Since the relative deadline D of the task is $68ms$, we need $\frac{150}{m} + 58.09 \le 68$ from which we conclude that at least $\lceil 150/(68 - 58.09)\rceil = \lceil 15.14 \rceil$, i.e., **16** *cores are needed (and hence choose $m \leftarrow 16$).*
Proceeding similarly, we can show that if the task were of SIL-1, then we should assign it at least 7 cores. □

[5]There are in fact a couple of possible choices here: in *push* schedulers, a master thread dispatches work to cores as they become idle; in *pull* schedulers, cores access the centralized queue themselves to grab jobs as they need them.

We now explain the algorithm illustrated in the example above. We seek to determine the number of cores m to be devoted to a given DAG task under federated scheduling. The values of the task parameters W, L, and D are obtained from the task specifications. The value of ϵ is dictated by safety requirements: it specifies the permitted probability of failure for the task. Given these values, in order to satisfy the safety requirements, it suffices to ensure that

$$\begin{aligned} & \frac{W}{m} + \Phi \times L + 1 + \Phi \times \log_2 \frac{1}{\epsilon} \leq D \\ \Leftrightarrow\ & \frac{W}{m} \leq D - \left(\Phi \times L + 1 + \Phi \times \log_2 \frac{1}{\epsilon}\right) \\ \Leftrightarrow\ & m \geq \left\lceil W \Big/ D - \left(\Phi \times L + 1 + \Phi \times \log_2 \frac{1}{\epsilon}\right) \right\rceil \end{aligned} \tag{9}$$

Expression 9 specifies the minimum number of cores that need to be devoted to a task — the presence of ϵ in this expression reveals the dependence of this number on the permitted probability of failure.

5 EXPERIMENTAL EVALUATION OF RANDOMIZED WORK-STEALING

We empirically evaluate the performance of randomized work-stealing with widely-used DAG benchmark programs. Our results confirm that the makespan of a program executed by randomized work-stealing has a relatively small variation in practice, and that this makespan is bounded by the concentration bound in Expression 8. In fact, the vast majority of the measured makespans do not exceed the expected makespan calculated using Expression 7.

Experiment setup. We conduct the experiments on a 16-core machine with two Intel Xeon 3.1Ghz Processors E5-2687W (each with 8 cores), 64GB memory, and Linux version 3.5.0. We disable processor throttling, processor sleeping, and hyper-threading. We use parallel benchmark programs written in the Cilk Plus language and choose the Cilk Plus branch 4.9.0 with a randomized work-stealing scheduler as the parallel runtime system.

For a benchmark program, we use the profiling tool Cilkview [22] to measure the work W and the burdened span L', where W is the cumulative execution time of all the nodes in a DAG and L' is the execution time of the longest chain of precedence-constrained nodes embedded with the explicit overhead of bookkeeping parallel nodes in the Cilk Plus runtime system and the implicit overhead due to the migration of stolen nodes [22]. Note that Cilkview only measures the number of instructions, so we convert the instruction counts into running time by measuring the actual running time of W upon the same input. We use the measured work W and burdened span L' to calculate the expected makespan of a benchmark program on a given number of cores using Expression 7.

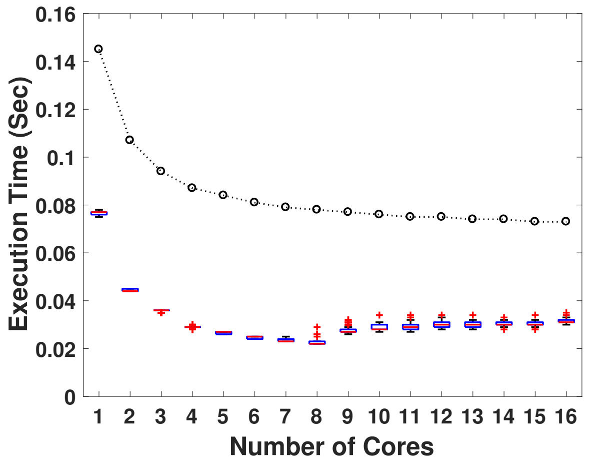
To obtain the actual makespan of a benchmark program on a given number of cores, we execute the program using the randomized work-stealing scheduler in Cilk Plus for 100 times and measure the execution times. We run each program with real-time priority to reduce the interference from the other system programs.

Benchmark programs. We use 6 programs with different properties and DAG structures, as described below, for evaluation.
(a) SC: Data Stream Clustering (SC) is from the PARSEC benchmark suite [10] and is designed for clustering continuously arriving data, as in multimedia applications and financial transactions. In PARSEC, SC was originally parallelized using parallel for-loops in pthread, OpenMP and TBB. We converted it into Cilk Plus using `cilk_for`. SC was run on 2048 data points, where each data point has 32 dimensions and 5 to 10 centers are allowed.
(b) Nbody: N-body Force simulation program (Nbody) is a scientific application in the problem based benchmark suite (PBBS) [28]. It calculates the motion of particles under the influence of mutual gravitational forces in a dynamic system. Nbody is parallelized in Cilk Plus and has a complex DAG structure. Nbody was run on 1383 3-dimension in-sphere points for 1000 rounds.
(c) Cholesky: This program, from the Cilk-5.4.6 release [29], uses the divide and conquer approach to performs Cholesky factorization of a matrix. Cholesky is parallelized using many `cilk_spawn` and `cilk_sync`, forming a general DAG structure. The Cholesky program was run on a matrix of size 3000×3000.
(d) Strassen: It is from the Cilk-5.4.6 release [29] and implements parallel matrix multiplication using Strassen's Algorithm. It recursively calculates using sub-matrices and is parallelized using `cilk_spawn` and `cilk_sync`. We run Strassen on matrices of size 512×512.
(e) LU: Similar to Cholesky, LU decomposition (LU) in the Cilk-5.4.6 release [29] also performs matrix factorization, but it has different requirements on the input and output matrices. It has abundant parallelism. We run LU on a matrix of size 2048×2048.
(f) Heat: This program from [29] simulates the heat diffusion using a Jacobi-type iteration. Its computation is repeatedly performed on a 2-dimension grid, where its parallelism comes from the recursive decomposition of 2D grids in stripes and is abundant. We run Heat on an input of 4096×1024 2D grid for 800 iterations.

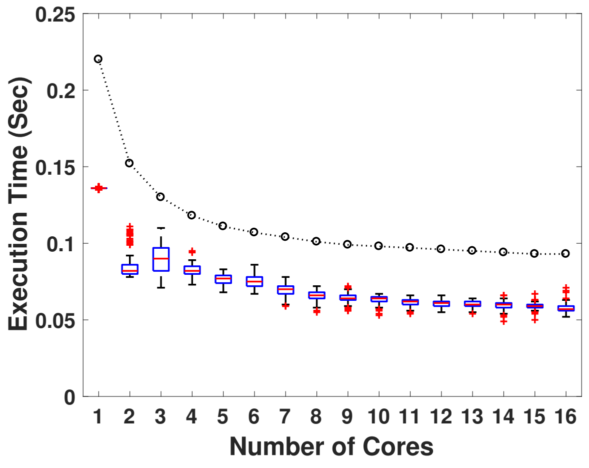
Outcome of the experiments. Figure 1 shows the results for the benchmark programs when executed on 1 to 16 cores by the randomized work-stealing scheduler in Cilk Plus. The expected makespans calculated using Expression 7 is displayed using the dotted lines with circles, and the measured actual makespans of 100 runs for each setting is presented using the boxplots.

We can see that the measured makespans are almost always smaller than the expected makespan in practice. For example, the actual makespans are only about 40% and 66% of the expected makespan for SC and Nbody, respectively. The Heat program has the closest distance between the two, where the measured makespans are about 98% of the expected value. The difference between the expected makespan and the average measured makespan is larger when the parallelism degree W/L' of the program is low. This is because the average measured makespan also follows the bound in Expression 7, but usually with a constant Φ smaller than 3.65 in practice. As SC and Nbody have smaller parallelism degrees of 4.1 and 5.9, respectively, their expected makespans deviate more from their average measured makespans. In contrast, Cholesky, Strassen, LU, and Heat have parallelism degrees of 27.99, 17.14, 89.74, and 343.55, respectively, all of which are larger than the maximum number of available cores.

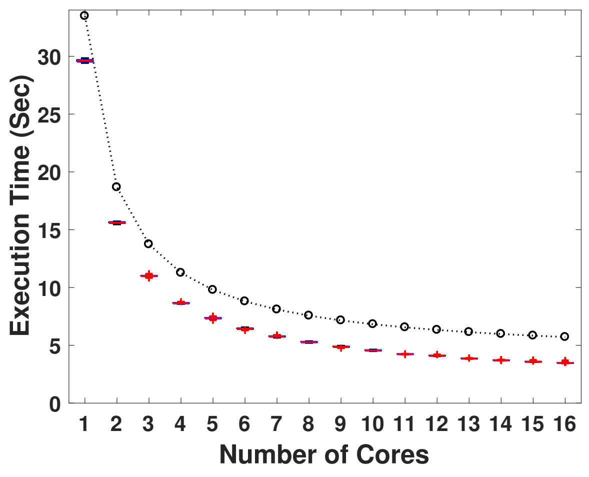
We would like to emphasize that *in none of our runs did the measured makespan exceed the concentration bound as computed according to Expression 8.* Indeed, of all the measurements for the different benchmark programs upon different numbers of cores, only in one — one of the 100 runs of Strassen upon 16 cores — did we observe a measured makespan exceeding the expected makespan. Even in this case, though, the concentration bound was not exceeded: with



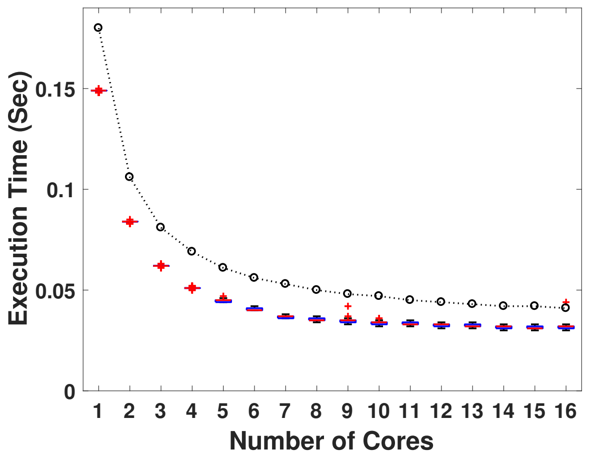
(a) SC



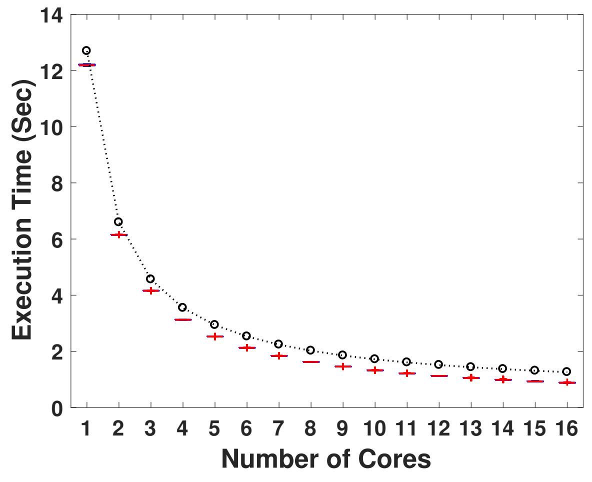
(b) Nbody



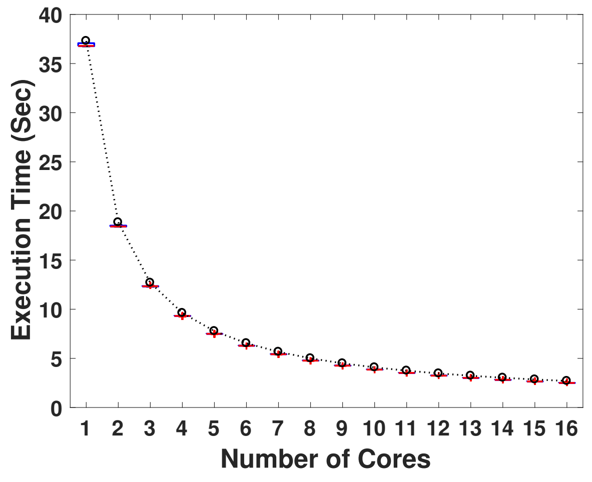
(c) Cholesky



(d) Strassen



(e) LU



(f) Heat

Figure 1: Expected makespan vs. actual makespan of a benchmark program executed on an increasing number of cores by a randomized work-stealing scheduler. The dotted lines with circles show the expected makespans calculated using Expression 7. The boxplots show the measured actual makespans of 100 repetitions for each setting, where the blue box gives the first, second, and third quartiles, and the red crosses are the outliers.

$W = 150$ms, $L' = 9$ms and a permitted failure probability 10^{-2}, the concentration bound for makespan on 16 cores is 67.47ms, as calculated in Example 2, and is larger than the largest measured makespan of 47ms that is the only measurement larger than the expected makespan (which is 43ms). This observation gives us confidence that the concentration bounds computed according to Expression 8 are indeed safe upper bounds in practice

We also observed in our experiments that the makespan upon scheduling with randomized work-stealing is highly *predictable*, with relatively small variances between runs in practice. In particular, we note that Nbody and Streamcluster have the largest relative standard deviations (i.e., standard deviation divided by the mean) in their measured makespan on more than two cores, which are only 4% to 10% and 2% to 5%, respectively. The other programs all have relative standard deviations smaller than 2.6%.

6 RELATIONSHIP TO PRIOR WORK

As mentioned earlier, a vast body of research has been conducted within the real-time computing community on the application of probabilistic techniques to the analysis of real-time systems — the interested reader is encouraged to read the previously cited surveys [17, 18] for a comprehensive overview and critique. Much of this prior work adopts one of two approaches:

(1) *Modeling* some system behavior (of programs, platforms, or interaction with the environment) via probability distributions;
(2) Using results from probability theory (such as Extreme Value Theory (EVT) [19]) to *obtain a statistical estimate of worst-case behavior* by extrapolating from measurements made upon a limited set of controlled experiments.[6]

It is important to note that analysis methods falling under both these approaches *assume* that the underlying behavior is probabilistic in nature: the objective of the methods is to obtain some characterization or model of this probabilistic behavior. We believe that this is often a reasonable assumption (and characterizing it is therefore a reasonable objective); however, as we have pointed out earlier in Section 1, getting such assumptions accepted by the larger safety-critical systems community is no easy task and involves a significant social and cultural component in addition to the purely technical ones of determining the right underlying models (and choosing the right theoretical tools, such as EVT, that enable us to do so). We reiterate that the main difference between our work and these prior approaches is that we are making no additional modeling assumptions: all our probability is within our (randomized) algorithms, where we have complete control over all aspects of it, while our assumptions about the rest of the system – the platform, and the interaction with the environment – continue to be the worst-case ones that have traditionally found favor in the safety-critical computing community. Furthermore, our use of concentration bounds to compute probabilities of failure fits in very well with the existing use of probabilities in safety standards such as IEC 61508 (Table 1) and its derivative standards.

[6]Some research, such as [12], has made a strong case that even simpler results from probability theory such as the Central Limit Theorem (CLT) and the Law of Large Numbers (LoLN), are applicable to a large class of safety-critical systems. Applying CLT or the LoLN typically requires one to make stronger assumptions concerning the independence of different random variables and additionally require that they are drawn from identical underlying distributions.

7 SUMMARY AND CONCLUSIONS

We proposed an approach for incorporating probabilistic and statistical techniques into the implementation and analysis of safety-critical systems. This approach is centered upon encapsulating all probabilistic concepts within specific randomized algorithms and analyzing their performance under worst-case assumptions concerning the run-time behavior of the rest of the system. Furthermore, we only use randomized algorithms for which good concentration bounds are known: these bounds provide tight limits on the probability that relevant aspects of the algorithm's run-time performance will exceed specified limits. Since many widely used safety standards allow for low but non-zero probabilities of failure, the use of randomized algorithms with such concentration bounds renders our proposed approach more compliant with the current standards and methodologies in the safety-critical systems community than other proposed probabilistic approaches. As stated earlier, there is a strong social and cultural aspect of getting new approaches accepted into the inherently conservative domain of safety-critical systems certification. Thus, the proposed approach shows great promise in enhancing efficiency within the constraints of current practice on certification of safety-critical systems.

There is a fairly broad and deep **agenda for future research** that must be pursued in order to enhance the likelihood of adopting this approach. Theoretically, this approach suggests a rich research agenda for the community, which is strongly motivated by practical system-building considerations. Specifically, we plan to identify other randomized algorithms that can enhance resource-efficiency in safety-critical systems and derive their concentration bounds. Results that are already present in the traditional theoretical computer science literature are not always ready for our use — many of the published bounds tend to be derived asymptotically and stated in asymptotic (O, Θ, Ω) terms, ignoring constant factors. While concentration bounds for the three example problems we have considered exist in the literature, it took considerable effort for us to identify the most useful formulation of the bounds — the ones that give us the greatest resource-efficiency for a given failure probability. We had to obtain a deep understanding of the derivations of these pre-existing asymptotic bounds in order to understand the steps where approximations were being made, and where possible, come up with ways of reducing these constant factors.

From a system's perspective, our ongoing and future research efforts seek to demonstrate the applicability of randomized algorithms in safety-critical systems via proof-of-concept implementations. We plan to actually integrate the randomized algorithms into industrial-strength systems and conduct extensive experiments to illustrate their performance. By so doing, we seek to establish that the randomized algorithms are easy to integrate into real systems, and that system performance is indeed along the lines of what was predicted by the theoretical application of concentration bounds.

ACKNOWLEDGMENTS

This research was supported, in part, by the National Science Foundation (USA) under Grant Numbers CNS–1948457, CNS–1814739, CNS–1911460, CPS–1932530, CNS–1850851, OIA–1937833, CCF-1733873 and CCF–1725647, and in part by Northrop Grumman Corporation grant.

REFERENCES

[1] Micah Adler, Petra Berenbrink, Tom Friedetzky, Leslie Ann Goldberg, Paul Goldberg, and Mike Paterson. 2003. A Proportionate Fair Scheduling Rule with Good Worst-case Performance. In *Proceedings of the Fifteenth Annual ACM Symposium on Parallel Algorithms and Architectures* (San Diego, California, USA) *(SPAA '03)*. ACM, New York, NY, USA, 101–108. https://doi.org/10.1145/777412.777430

[2] A. Atlas and A. Bestavros. 1998. Statistical rate monotonic scheduling. In *Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No.98CB36279)*. 123–132. https://doi.org/10.1109/REAL.1998.739737

[3] Sanjoy Baruah. 2018. Predictability Issues in Mixed-Criticality Real-Time Systems. In *Principles of Modeling: Essays Dedicated to Edward A. Lee on the Occasion of His 60th Birthday*, Marten Lohstroh, Patricia Derler, and Marjan Sirjani (Eds.). Springer International Publishing, 77–87.

[4] Sanjoy Baruah, Marko Bertogna, and Giorgio Buttazzo. 2015. *Multiprocessor Scheduling for Real-Time Systems*. Springer Publishing Company, Incorporated.

[5] Sanjoy Baruah, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, Leem Stougie, and Andreas Wiese. 2012. A generalized parallel task model for recurrent real-time processes. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS 2012)*. San Juan, Puerto Rico, 63–72.

[6] Michael A. Bender, Martin Farach-Colton, and William Kuszmaul. 2019. Achieving Optimal Backlog in Multi-Processor Cup Games. *CoRR* abs/1904.02861 (2019). arXiv:1904.02861 http://arxiv.org/abs/1904.02861

[7] Guillem Bernat, Alan Burns, and Martin Newby. 2005. Probabilistic Timing Analysis: An Approach Using Copulas. *J. Embedded Comput.* 1, 2 (April 2005), 179–194. http://dl.acm.org/citation.cfm?id=1233760.1233763

[8] G. Bernat, A. Colin, and S. Petters. 2003. *pWCET: A tool for probabilistic worst-case execution time analysis of real-time systems*. Technical Report. The University of York, England.

[9] G. Bernat, A. Colin, and S. M. Petters. 2002. WCET analysis of probabilistic hard real-time systems. In *2002 IEEE Real-Time Systems Symposium (RTSS)*. 279–288. https://doi.org/10.1109/REAL.2002.1181582

[10] Christian Bienia. 2011. *Benchmarking Modern Multiprocessors*. Ph.D. Dissertation. Princeton University. http://parsec.cs.princeton.edu.

[11] Robert D. Blumofe, Christopher F. Joerg, Bradley C. Kuszmaul, Charles E. Leiserson, Keith H. Randall, and Yuli Zhou. 1995. Cilk: An Efficient Multithreaded Runtime System. In *Proceedings of the Fifth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (Santa Barbara, California, USA) *(PPOPP '95)*. ACM, New York, NY, USA, 207–216. https://doi.org/10.1145/209936.209958

[12] Alan Burns and David Griffin. 2011. Predictability as an emergent behaviour. In *Proceedings of the Workshop on Compositional Theory and Technology for Real-Time Embedded Systems*.

[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. 2009. *Introduction to Algorithms*. (third ed.). MIT Press.

[14] Liliana Cucu and Eduardo Tovar. 2006. A Framework for the Response Time Analysis of Fixed-priority Tasks with Stochastic Inter-arrival Times. *SIGBED Rev.* 3, 1 (Jan. 2006), 7–12. https://doi.org/10.1145/1279711.1279714

[15] Robert Davis and Alan Burns. 2011. A Survey of Hard Real-Time Scheduling for Multiprocessor Systems. *Comput. Surveys* 43, 4 (2011).

[16] Robert I. Davis, Alan Burns, and David Griffin. 2017. On the Meaning of pWCET Distributions and their use in Schedulability Analysis. In *Proceedings 2017 Real-Time Scheduling Open Problems Seminar (RTSOPS)*.

[17] Robert I. Davis and Liliana Cucu-Grosjean. 2019. A Survey of Probabilistic Schedulability Analysis Techniques for Real-Time Systems. *LITES* 6, 1 (2019), 04:1–04:53. https://doi.org/10.4230/LITES-v006-i001-a004

[18] Robert I. Davis and Liliana Cucu-Grosjean. 2019. A Survey of Probabilistic Timing Analysis Techniques for Real-Time Systems. *LITES* 6, 1 (2019), 03:1–03:60. https://doi.org/10.4230/LITES-v006-i001-a003

[19] Laurens de Haan and Ana F. Ferreira. 2006. *Extreme Value Theory: An Introduction*. Springer Publishing Company, Incorporated.

[20] Jeremey Fineman. 2005. *Provably good race detection that runs in parallel*. Master's thesis. Massachusetts Institute of Technology, Deptartment of Electrical Engineering and Computer Science,.

[21] R. Graham. 1969. Bounds on multiprocessor timing anomalies. *SIAM J. Appl. Math.* 17 (1969), 416–429.

[22] Yuxiong He, Charles E Leiserson, and William M Leiserson. 2010. The Cilkview scalability analyzer. In *22nd ACM symposium on Parallelism in algorithms and architectures (SPAA)*. 145–156.

[23] C. A. R. Hoare. 1962. Quicksort. *Computer Journal* (1962), 10–15.

[24] J. K. Lenstra and A. H. G. Rinnooy Kan. 1978. Complexity of Scheduling under Precedence Constraints. *Operations Research* 26, 1 (1978), 22–35.

[25] J. Li, S. Dinh, K. Kieselbach, K. Agrawal, C. Gill, and C. Lu. 2016. Randomized Work Stealing for Large Scale Soft Real-Time Systems. In *2016 IEEE Real-Time Systems Symposium (RTSS)*. 203–214. https://doi.org/10.1109/RTSS.2016.028

[26] Jing Li, Abusayeed Saifullah, Kunal Agrawal, Christopher Gill, and Chenyang Lu. 2014. Analysis Of Federated And Global Scheduling For Parallel Real-Time Tasks. In *Proceedings of the 2012 26th Euromicro Conference on Real-Time Systems (ECRTS '14)*. IEEE Computer Society Press, Madrid (Spain).

[27] C.J.H. McDiarmid and R.B. Hayward. 1996. Large Deviations for Quicksort. *J. Algorithms* 21, 3 (Nov. 1996), 476–507. https://doi.org/10.1006/jagm.1996.0055

[28] PBBS. 2014. Problem Based Benchmark Suite. http://www.cs.cmu.edu/~pbbs.

[29] MIT Supercomputing Technologies Group. 1998. Cilk 5.4.6 Reference Manual. http://supertech.lcs.mit.edu/cilk/manual-5.4.6.pdf.

[30] Ola Svensson. 2010. Conditional hardness of precedence constrained scheduling on identical machines. In *Proceedings of the 42nd ACM symposium on Theory of computing* (Cambridge, Massachusetts, USA) *(STOC '10)*. ACM, New York, NY, USA, 745–754. https://doi.org/10.1145/1806689.1806791

[31] B. Tanasa, U. D. Bordoloi, P. Eles, and Z. Peng. 2015. Probabilistic Response Time and Joint Analysis of Periodic Tasks. In *2015 27th Euromicro Conference on Real-Time Systems*. 235–246. https://doi.org/10.1109/ECRTS.2015.28

[32] Marc Tchiboukdjian, Nicolas Gast, Denis Trystram, Jean-Louis Roch, and Julien Bernard. 2010. A Tighter Analysis of Work Stealing. In *Algorithms and Computation*, Otfried Cheong, Kyung-Yong Chwa, and Kunsoo Park (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 291–302.

[33] J. Ullman. 1975. NP-complete scheduling problems. *J. Comput. System Sci.* 10, 3 (1975), 384 – 393.