Iterative Robust Multiprocessor Scheduling *

Shreya Adyanthaya[1], Marc Geilen[1], Twan Basten[1,2], Jeroen Voeten[2,1], Ramon Schiffelers[3,1]
[1]Eindhoven University of Technology, Eindhoven, The Netherlands
[2]TNO-ESI, Eindhoven, The Netherlands
[3]ASML, Veldhoven, The Netherlands
{s.adyanthaya, m.c.w.geilen, a.a.basten, j.p.m.voeten}@tue.nl, ramon.schiffelers@asml.com

ABSTRACT

General purpose platforms are characterized by unpredictable timing behavior. Real-time schedules of tasks on general purpose platforms need to be robust against variations in task execution times. We define robustness in terms of the expected number of tasks that miss deadlines. We present an iterative robust scheduler that produces robust multiprocessor schedules of directed acyclic graphs with a low expected number of tasks that miss their deadlines. We experimentally show that this robust scheduler produces significantly more robust schedules in comparison to a scheduler using nominal execution times on both real world and synthetic test cases.

1. INTRODUCTION

Trends in embedded platform design show a shift from predictable application-specific platforms to general purpose platforms with low predictability. General purpose solutions are cost effective, and with the introduction of multi-cores, they can meet the requirements of embedded applications. The main drawback of general purpose platforms is that they exhibit variations in the execution timings of applications. This unpredictability calls for robustness in the application schedules on these platforms. Robustness of a schedule is the measure of its tolerance to variations in the application execution.

We target real time applications consisting of tasks with deadlines. Examples are the servo control systems consisting of control tasks with strict latency requirements. These are modeled as Directed Acyclic Graphs (DAGs) that are mapped onto execution platforms of homogenous general purpose processors [17]. In classical real time scheduling, robustness of schedules is guaranteed by assuming worst-case execution times. A static order schedule is robust if it meets all its deadlines (is feasible) in the worst-case. Application execution timings on general purpose platforms exhibit variations in which the most-likely (nominal) execution times are typically closer to the best-case execution times than to the worst-case execution times. In general purpose platforms, worst case timings arise due to, for instance, cache misses, interrupts, branch predictions, memory access delays or hardware failures. Recovery from these require significant amounts of time. Hence, the worst-case execution times are typically relatively large. Scheduling for the worst case then needs additional platform resources to accommodate the worst case. Moreover, it is very unlikely that all tasks execute close to their worst-case execution times. Hence the traditional worst case scheduling is not practical in these situations and we thus need a stochastic scheduling mechanism that takes these execution time variations into account and produces schedules that are robust in nature.

We present an iterative approach that starts from a list scheduler with a non-stochastic heuristic. It then repeatedly performs robustness analysis followed by list scheduling with a new stochastic robustness heuristic. We quantify robustness using a metric based on the expected number of tasks that miss their deadlines in the schedule. We estimate this metric using an existing analysis technique that effectively combines analytical reasoning with simulations [3]. We extend this analysis with a new metric that quantifies the task-level impact on expected deadline misses. This metric is used in an iterative highest robustness impact first heuristic to guide the stochastic list scheduler during the iterations. Our contributions are:

1. the overall iterative robust scheduling approach,
2. the task-level robustness metric,
3. and the stochastic list scheduling heuristic

We test the approach on 1000 synthetic DAGs and our scheduler produces better schedules in 76% of the test cases with 12% improvement in robustness, on average, compared to a non-robust scheduler. We also apply the scheduler on video conference applications as well as a time critical application of controllers for wafer scanners where it shows from 3% upto 79% robustness improvement.

The paper is organized as follows. Section 2 summarizes related work. Section 3 introduces the preliminaries. Section 4 gives the problem statement and the flow of the solution approach. Section 5 describes the extended robustness analysis and the new metric. Section 6 details on the scheduling heuristic and Section 7 elaborates the iterative robust scheduler that uses the heuristic. Experiments are given in Section 8 and conclusions are drawn in Section 9.

*RTNS'15, November 4-6, 2015, Lille, France.

2. RELATED WORK

Task execution time variations that impact schedule robustness can arise due to multiple factors, resulting in different models of variation. In [4][10], variations are modeled as external perturbations and their effects on system performance are used to study robustness. We model variations in task execution times which are internal to the system, due to the general purpose processing (cache misses, branch predictions, memory access delays). External perturbations do not play a role. Variations can be modeled using deterministic parameters such as min/max bounds and expected values, allowing for slack based robust scheduling techniques [5]. We use the richer model of probability density functions. Often, normal distributions are used [6][8]. However in most practical cases, variations are not symmetric and are skewed in nature. We base our robustness analysis technique on realistic skewed bounded PERT distributions of execution times [3].

Robustness can be quantified using different metrics such as slack based metrics [8]. Stochastic metrics include more information. Deadline miss probabilities of tasks [21] is an example metric. This does not give a global measure of schedule robustness if there are multiple tasks with different deadlines. Global schedule robustness can be quantified using the probability that the makespan is within specific bounds [18]. This corresponds to all tasks having the same deadline. Our work considers task level deadlines which can differ within a graph.

Multiprocessor real-time scheduling with fixed execution times has been studied extensively, a survey of which is presented in [9]. In [8], robust scheduling using slack introduces temporal slack to each task such that a certain level of uncertainty can be absorbed without rescheduling. Here, normal distributions are used to describe machine breakdown behaviours and their parameters are used to calculate the amount of slack added per task. Task ordering is not changed. Introducing unnecessary slack can adversely effect meeting latency requirements. In our work, we alternatively determine stochastic task robustness and derive metrics that allow us to produce different task orderings that result in more robust schedules. A partitioning heuristic [11] takes execution time overruns into account and aims to assign tasks to resources such that the amount of allowable overrun per task is maximized. Its focus is on the assignment problem using fixed priority and deadline monotonic priority assignment. Our work finds robust ordering of tasks for a given binding on resources and aims to maximize probabilities of meeting task deadlines. This is often seen in real time applications such as control applications where tasks in interconnected control loops are mapped on same processors to reduce communication overhead.

Recent work on robust scheduling [15] looks at non-preemptive scheduling under task duration uncertainty. It presents a hybrid offline/online technique to meet hard real time guarantees with limited idle time insertion called Precedence Constraint Posting. Bounds and expected values are used instead of probability distributions to deal with uncertainty. Tasks are assigned a common deadline corresponding to global real-time constraints. The scheduling aims to reduce makespan. In our context, each task has its own deadline and reducing makespan does not guarantee all tasks meeting their deadlines. As meeting task latency requirements is our primary goal, we quantify robustness in terms of the expected number of task deadline misses and provide a scheduling heuristic to improve this. We use probability distributions for task execution times to analyze schedule robustness allowing for more accurate analysis in comparison to using bounds alone.

Other recent work on scheduling precedence constrained stochastic tasks on heterogenous cluster systems performs list scheduling based on stochastic bottom levels and stochastic dynamic levels [14]. It aims to reduce schedule makespan and does not address robust scheduling under task deadline constraints. The expected schedule makespan is measured by assuming that tasks have normal execution time distributions and using Clark's equations to estimate the max of distributions due to task synchronization [6]. Our work allows skewed task execution times distributions by adopting and refining the robustness analysis of [3]. It combines analytical results with limited simulations. It overcomes the drawback of Clark's equations which are not accurate when performing max operations on distributions with large differences in their standard deviations [16]. Our analysis forms part of the iterative robust scheduling loop.

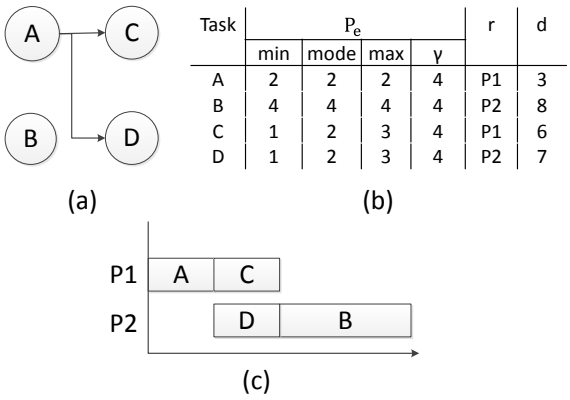
Effective online scheduling approaches that deal with execution time uncertainty include reservation-based scheduling such as the Constant Bandwidth Server (CBS) introduced in [1]. Using a task model which comprises hard real time and soft real time tasks, they first schedule hard real time tasks based on latency requirements and consequently use reservation methods for soft real time tasks. These tasks are assigned dedicated processor bandwidth that can be reclaimed by other tasks on early completion. However in our task model, all tasks are soft real time and the scheduling aim is to reduce the number of possible deadline misses by making a robust static order schedule. The scheduling is performed on design time or during machine start-up time [2] whenever the application graph is available. The static order schedule thus produced cannot be changed online and tasks are run using a dispatcher in the same order. This is typical in throughput and latency driven applications that try to minimize any scheduling overhead during run-time of the system.

3. PRELIMINARIES

3.1 Applications and Schedules

An *application* is a DAG, $G = (T, D)$, with a finite set T of tasks and a set $D \subseteq T^2$ of task dependencies. A multiprocessor platform consists of processors called *resources*, represented by R.

Let $\mathbb{D}$ be the set of all probability density functions such that if $d \in \mathbb{D}$, then $d : \mathbb{R}^{\geq 0} \rightarrow \mathbb{R}^{\geq 0}$. A *task* $A \in T$ is a tuple $A = (P_{e_A}, r_A, d_A) \in \mathbb{D} \times R \times \mathbb{R}^{\geq 0}$. Here, $r_A \in R$ is the resource in R that A is bound to, P_{e_A} is the probability density function of the continuous execution time distribution of A, with e_A denoting the random variable for the execution time of A, and d_A is the deadline of A. Tasks have a unique most-likely execution time, called the *nominal* execution time which is given by the peak of the distribution (assuming single peaked distributions). In this work, task execution time distributions are obtained by profiling and we assume that task execution times vary independently. An alternative method to obtain execution time distributions is to use static probabilistic worst-case execution time analysis [12]. Dealing with task execution time correlations



Task	P_e				r	d
	min	mode	max	γ		
A	2	2	2	4	P1	3
B	4	4	4	4	P2	8
C	1	2	3	4	P1	6
D	1	2	3	4	P2	7

Figure 1: Example to illustrate list scheduling

is left for future work.

A *dependency* $(A, B) \in D$ between tasks A and B states that B can begin its execution only after the completion of A. A *static-order schedule* S is a mapping $S : R \rightarrow T^*$ from the set of resources to ordered lists of tasks such that all data dependencies are respected. Given a schedule, the tasks have stochastic start and completion times due to their random execution times. For task A, s_A^S and c_A^S denote the random variables for the start and completion time in S, respectively. The start time of the first task scheduled on a resource, without any predecessors, is zero with probability 1. Completion times are derived by adding the task execution times to the corresponding task start times. The probability density functions for the start time and the completion time of A are denoted by $P_{s_A}^S$ and $P_{c_A}^S$ respectively. The nominal start and completion times of a task are derived by using nominal execution times for all tasks in the graph. A schedule is *feasible* if all its tasks meet their deadlines under nominal execution times. Note that during run-time, tasks are scheduled in the same order as in the static order schedule without pre-emptions. As such, even though the binding is fixed, dependencies between tasks on different processors make the scheduling problem more challenging than uni-processor scheduling.

3.2 List Scheduling with EDDF Heuristic

A *list scheduler* schedules a DAG by maintaining a list of enabled tasks at each scheduling step. A task is enabled when all its predecessors have been scheduled. Repeatedly, the scheduler chooses an enabled task to be scheduled next, based on a ranking heuristic. The iterative robust scheduling mechanism presented in this paper is compared with the list scheduler with an *earliest due-date first* (EDDF) heuristic, that ranks tasks according to earliest due-dates. A *due-date* is an upper bound on the completion time of the task which upon being missed renders the schedule infeasible. In other words, if a task misses its due-date at least some future task including itself will miss its deadline. The EDDF heuristic has been statistically shown to outperform classical methods in producing feasible schedules for directed acyclic graphs with fixed binding and deadlines [2].

Due-dates are used during scheduling to avoid infeasible branches in the search space at early stages by making better scheduling choices. In [2], an efficient and linear approach is used to compute tight task due-dates by traversing backwards starting from the leaf nodes of the task graph. Due-dates are computed using deterministic nominal execution times of tasks.

To understand this scheduler consider the task graph in Figure 1(a). The attributes of the tasks are given in Figure 1(b). Due-dates of task A, B, C and D are computed using the approach of [2] to be 3, 8, 6 and 7 respectively. In this case, they are simply equal to their deadlines. In the beginning, only A and B are enabled since they do not have any predecessors. Based on the earliest due-date first heuristic, A is chosen to be scheduled. Thereafter, C and D become enabled along with B. Again based on due-dates, C is scheduled followed by D and then B resulting in the schedule shown in Figure 1(c).

3.3 Robustness Analysis

We define *robustness* for static order multiprocessor schedules as follows.

Definition 1 *Robustness of a static-order schedule S is measured by the expected value of the number of tasks that miss their deadlines in S. Let random variable X represent the number of tasks that miss deadlines in the schedule. From the completion time distributions of the tasks in T, the expected value of X is computed as follows. Note that the lower this value, the higher the robustness.*

$$E[X] = \sum_{A \in T} \mathbb{P}[c_A^S > d_A] \tag{1}$$

The deadline miss probability of a task A, given by the term $\mathbb{P}[c_A^S > d_A]$ in the above equation, is computed using the derivative $\int_{d_A}^{\infty} P_{c_A}^S(t)dt$. To compute this, we need to obtain its completion time distribution from its execution time distribution. We obtain execution time distributions by profiling and then use the curve fitting approach of [3] to obtain modified PERT distributions. A modified PERT distribution is a variant of the Beta distribution and is represented using four parameters, namely the minimum value, maximum value, most likely (mode) value and an intensity parameter (γ) which specifies the concentration of the distribution around the mode. Once we have the execution time distributions, we need to determine the completion time distributions. Due to data dependencies between tasks on the same and across resources, obtaining completion time distributions requires max-plus operations to be performed on execution time distributions. It is computationally intractable to compute the exact max of distributions. Simulations have the drawback of missing rare events. We adopt the *robustness analysis* technique of [3]. This approach overcomes the complexity of performing max-plus operations on distributions by using three steps.

1. It uses deterministic max-plus algebra on the best-case and worst-case execution time values of tasks to compute their best-case and worst-case completion times in one run through the graph.

2. It then uses limited simulations by drawing samples from the execution time distributions to obtain completion time samples. Each simulation corresponds to

one run through the graph and returns one completion time sample per task.

3. Finally, we use curve fitting which takes as input the best-case and worst-case completion time obtained in step 1 and the histogram of the mass from step 2 to obtain the modified PERT distribution of the completion time of each task.

Task execution times are assumed to vary independently as mentioned earlier. However, due to data dependencies between tasks there can also be correlations between completion times of tasks. Since we generate samples by walking through the tasks and dependencies in the graph in step 2, correlations due to data dependencies are implicitly taken into account in these completion time samples. Since we analyze a static order schedule, there are no scheduling anomalies. After estimating completion time distributions, Equation 1 gives schedule robustness.

4. PROBLEM DEFINITION AND SOLUTION OVERVIEW

4.1 Problem Statement

Given an application with tasks bound to a set of resources, find the static-order multiprocessor schedule with the lowest expected value of the number of tasks that miss deadlines.

Producing a schedule with the maximum likelihood of meeting an optimization criterion under processing time uncertainty such as total flow time is proven to be NP-hard in [7]. Our problem is a generalization of this as we can connect all leaf tasks in the DAG to an additional task which has the flow time as its deadline. Hence, our problem is NP-hard and finding the optimal robust schedule is computationally intractable. We aim to design a stochastic robust scheduler that, starting from a certain schedule (produced using a non-stochastic scheduler in our case) uses an efficient scheduling heuristic to improve schedule robustness.

4.2 Iterative Robust Scheduler: Overview

Our scheduling approach iterates between a scheduling step and a robustness analysis step. The scheduling step uses list scheduling with a heuristic to improve robustness. List scheduling is very efficient (not considering the ranking function) and commonly used for scheduling DAGs [13]. Our approach uses a deterministic scheduler that produces a starting schedule using nominal execution times of tasks. Given this schedule, our work aims to iteratively produce more robust schedules. The idea is to use the robustness analysis of the schedule obtained in an iteration to guide the scheduler towards a more robust schedule in the next iteration.

Figure 2 illustrates the flow of the iterative robust scheduler. It begins with schedule S_0 obtained from the list scheduler with the EDDF heuristic. We perform robustness analysis of S_0, involving computation of the task completion time distributions, to compute a robustness *impact metric* per task. These metrics obtained from S_0 are used to refine the list scheduling towards robustness with an *iterative highest robustness impact first* (IHRIF) heuristic resulting in schedule S_1. This list scheduler with IHRIF heuristic differs from the EDDF list scheduler of the initial iteration only

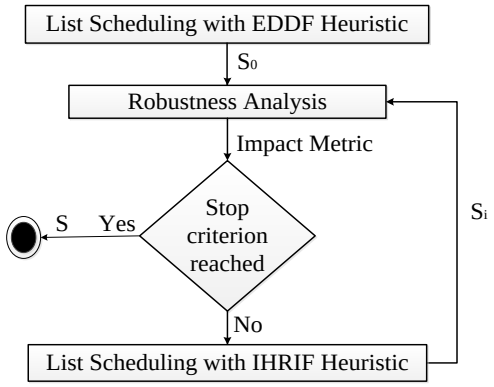


Figure 2: Flow of Iterative Robust Scheduler

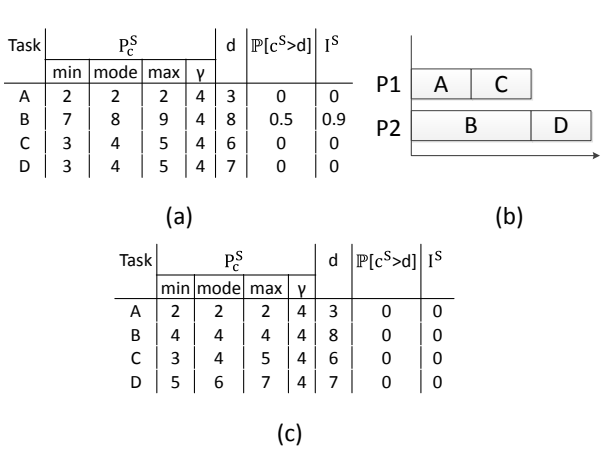


Figure 3: Example to illustrate the Iterative Robust Scheduling flow

in the heuristic that determines the scheduling choices. The construction of the schedule, however, is done using nominal execution times of tasks for scheduling speed and simplicity's sake. This is because maintaining and updating start and completion time distributions during schedule construction is very expensive. In the next iteration, the impact metric computed from S_1 is used in the list scheduling heuristic to obtain S_2. This repeats until either the process converges with no schedule changes or until a fixed number of iterations.

To illustrate the flow of the scheduler, we reuse the example from Figure 1(a). The initial schedule S_0 is the one produced using list scheduling with the earliest due-date first heuristic and is given in Figure 1(c). We perform robustness analysis on this schedule. The expected value of deadline misses is 0.5 due to the deadline miss probability of B. The task completion time distributions, deadline miss probabilities and impact metrics of the tasks in S_0 are given in Figure 3(a). The computation of the impact metric is elaborated in Section 5.2. Tasks A, C and D have zero impact since the completion time distributions of their dependent

tasks are within their own deadlines. Task B has a positive impact since its deadline is in the middle of its completion time distribution. In the next iteration, list scheduling is performed using the highest impact first heuristic. As a result, B gets priority and is scheduled first resulting in the schedule S_1 given in Figure 3(b) and the corresponding task details in Figure 3(c). The expected value of deadline misses of this new schedule obtained after performing robustness analysis is 0 implying that it has better robustness than the initial schedule.

The explanation of the iterative robust scheduler is divided into three sections. Section 5 elaborates on the refinement of the robustness analysis to compute the impact metric. Section 6 explains the IHRIF heuristic which uses the impact metric computed from the refined robustness analysis and Section 7 details on the list scheduler with the IHRIF heuristic.

5. REFINED ROBUSTNESS ANALYSIS: IMPACT METRIC

The robustness analysis, briefly explained in Section 3.3, uses task deadline miss probabilities to compute the schedule robustness using Equation 1. We refer to the schedule-level notion of robustness as *global robustness* and task-level notion of robustness as *local robustness*. Since list scheduling is performed on a per task basis, the global notion of robustness is insufficient to construct a robust schedule. For construction, we require a local notion of robustness that is consistent with and aids in the improvement of the global robustness. Hence, we refine the current robustness analysis with a new impact metric per task that provides this local notion of robustness.

5.1 Impact Metric

Given a schedule S, this metric quantifies the impact of a task on the expected number of deadline misses in S. We use task deadline miss probabilities to extract the impact metric per task. The hypothesis is that schedule robustness can be improved by scheduling highest impact tasks first. The rationale behind the heuristic is that delaying a task which has a high impact on the deadline misses of its future tasks will cause an increase in the expected deadline misses.

Definition 2 (Impact Metric) *The impact metric of a task A at a given completion time c_A^S in a schedule S, denoted by $I_A^S : \mathbb{R}^{\geq 0}$, is the derivative of the expected value of number of task deadline misses in S over the delay in c_A.*

To estimate the impact metric of a task, we define the dependent set of A as follows.

Definition 3 (Dependent set) *Dependent set of task A, denoted by DS_A, is the set of tasks including A and all tasks that have a direct or indirect dependency from A in the DAG. These do not include dependencies due to scheduling order.*

Since the scheduling of task A influences the scheduling of the tasks in DS_A, we need to compute the expected value of the number of tasks that miss deadlines in DS_A to compute I_A^S. Let random variable X_A represent the number of tasks from DS_A that miss deadlines in S. The expected value of X_A is given by

$$E[X_A] = \sum_{B \in DS_A} \mathbb{P}[c_B^S > d_B] \tag{2}$$

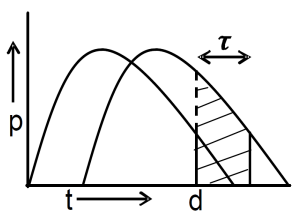


Figure 4: Increase in deadline miss probability due to delaying a task

The value of $E[X_A]$ as a function of the completion time of A is denoted by the function $v_{E[X_A]} : \mathbb{D} \rightarrow \mathbb{R}^{\geq 0}$. The derivative of $v_{E[X_A]}$ gives the impact metric representing the impact on $E[X_A]$ caused by delaying c_A^S by a small amount of time. To express the shifting of a probability density function by an amount of time $\tau \in \mathbb{R}^{\geq 0}$, we define ${}_\tau d : \mathbb{R}^{\geq 0} \rightarrow \mathbb{R}^{\geq 0}$ such that ${}_\tau d(t) = d(t-\tau)$ for all $t \in \mathbb{R}$. Using this, ${}_\tau c_A^S$ represents the random variable for the completion time of A after the shift of τ and ${}_\tau P_{C_A}^S$ represents the corresponding probability density function. This in turn causes an increase of 'at most' τ in the completion times of tasks in DS_A. From Equation 2, we deduce that this may cause the expected value of X_A to increase. Hence, the impact metric of A in S is captured as follows.

$$I_A^S = \lim_{\tau \rightarrow 0} \frac{v_{E[X_A]}({}_\tau c_A^S) - v_{E[X_A]}(c_A^S)}{\tau} = \frac{dv_{E[X_A]}({}_\tau c_A^S)}{d\tau} \tag{3}$$

In the next subsection we explain the computation of this derivative.

5.2 Impact Metric Computation

In schedule S, any change in $E[X_A]$ due to delaying A is only caused by tasks in DS_A that experience an increase in their deadline miss probabilities due to the delay. To compute the derivative in Equation 3, we consider the delay to be infinitesimal. This in turn can cause only an infinitesimal shift in the completion times of tasks in DS_A. The increase in the deadline miss probability of a task $B \in DS_A$ for an infinitesimal shift in c_B^S is equal to the probability density at its deadline:

$$\lim_{\tau \rightarrow 0} (\mathbb{P}[{}_\tau c_B^S > d_B] - \mathbb{P}[c_B^S > d_B]) = P_{c_B}^S(d_B) \tag{4}$$

Here, $\mathbb{P}[c_B^S > d_B]$ is the deadline miss probability of B before the shift and $\mathbb{P}[{}_\tau c_B^S > d_B]$ gives the deadline miss probability after shifting by τ. The probability density at its deadline d_B is given by $P_{c_B}^S(d_B)$. How we arrive at this is illustrated in Figure 4 that shows a task's completion time distribution shifted by τ. This causes an increase in its deadline miss probability given by the shaded portion. For an infinitesimal increase $\tau \rightarrow 0$, this area collapses into a straight line of length equal to the probability density at the task deadline d.

Another important aspect of the computation is that shifting the completion time of A by τ will increase the deadline miss probability of task $B \in DS_A$ only if the slack between A and B is smaller than τ. This slack in S, denoted by

sl^S_{AB}, is the difference between the completion time of A and the start time of B taking dependent tasks in between into account. Taking the limit $\tau \to 0$ implies that task B will be affected iff $sl^S_{AB} = 0$. Note that sl^S_{AB} can never be smaller than 0. The probability of sl^S_{AB} being 0, denoted by $\mathbb{P}[sl^S_{AB} = 0]$, can be approximated by counting the number of times the slack equals zero in the simulations used in the robustness analysis. These probabilities, starting from immediate predecessors of leaf nodes, are propagated backwards through the schedule to get the probabilities also for tasks which do not have a direct dependency between them. Also, the probability of zero slack between a task and itself, given by $\mathbb{P}[sl^S_{AA} = 0]$, is 1. Using this, we compute the impact metric of A as follows.

$$I^S_A = \sum_{B \in DS_A} \mathbb{P}[sl^S_{AB} = 0] \cdot P^S_{c_B}(d_B) \tag{5}$$

Here, we see that the impact metric of a task is based heavily on its successors and can be computed only after the successors have been scheduled. Hence, computing I^S_A requires the complete schedule of all tasks in DS_A. Similarly any change in the scheduling of tasks in DS_A might require I^S_A to be recomputed. Therefore, the robust scheduling approach presented in this work uses an iterative mechanism and cannot be performed efficiently in one shot. We compute the impact metrics of tasks in a schedule generated in the previous iteration and use it for scheduling in the next iteration. The idea behind this approach is to progress towards a more robust schedule by performing robustness based scheduling decisions in an iteration, for high impact tasks detected in the schedule of the previous iteration.

To illustrate the impact metric computation we reuse the example schedule in Figure 3(b) but with tighter deadlines to get non-zero impacts. The execution time and completion time distributions, deadlines, and impacts of the tasks are in Figure 5(b). Impacts of B, C and D having no successors are equal to the densities of their completion distributions at their respective deadlines. The probability of zero slack between A and itself is 1. The probability of zero slack between A and C is also 1 (C always immediately follows A). The probability of zero slack between A and D is 0 (the start time of D depends on the completion time of B which is always greater than that of A). Hence, the impact of A is computed as follows.

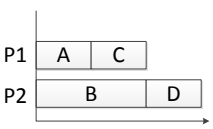
$$\begin{aligned} I^S_A &= \mathbb{P}[sl^S_{AA} = 0] \cdot P^S_{c_A}(d_A) + \mathbb{P}[sl^S_{AC} = 0] \cdot P^S_{c_C}(d_C) \\ &\quad + \mathbb{P}[sl^S_{AD} = 0] \cdot P^S_{c_D}(d_D) \\ &= 1 \cdot 0 + 1 \cdot 0.9 + 0 \cdot 0.9 = 0.9 \end{aligned}$$

Note that different deadlines can result in different impact values and different scheduling choices for the same graph. In the next subsection we elaborate on the IHRIF heuristic which uses the impact metric to guide the scheduling decisions.

6. IHRIF HEURISTIC

The iterative approach uses a list scheduler which chooses tasks from the list of enabled tasks using the IHRIF heuristic. The various components of the heuristic are as follows.

1) *Impact History*: During the robustness analysis of S, we only obtain the impact of A at its current completion



(a)

Task	P_e				r	P^S_c				d	I^S
	min	mode	max	γ		min	mode	max	γ		
A	2	2	2	4	P1	2	2	2	4	3	0.9
B	4	4	4	4	P2	4	4	4	4	5	0
C	1	2	3	4	P1	3	4	5	4	4	0.9
D	1	2	3	4	P2	5	6	7	4	6	0.9

(b)

Figure 5: Example to illustrate the Impact Metric Computation

time in S. At different decision points during scheduling, the completion time of A can be different from the ones observed in prior schedules. We need to be able to estimate impact values at all possible completion times. To do so, we maintain a history of all the impact values observed in prior iterations along with their corresponding nominal completion times. We use nominal completion times mainly to simplify the approach by removing the complexity of combining density functions in the impact history. With more iterations, we obtain more values and can approximate the impact metrics of tasks with changing completion times.

Definition 4 (Impact History) *The impact history of a task A is a function which maps an iteration i to a tuple of its nominal completion time and the corresponding impact metric at that iteration. It is denoted by the mapping $IH_A : \mathbb{N} \to (\mathbb{R}^{\geq 0}, \mathbb{R}^{\geq 0})$. The functions $left(IH_A(i))$ and $right(IH_A(i))$ give the nominal completion time and the impact metric of A at the i^{th} iteration.*

Using this function during scheduling, we need to deduce an *impact estimate* of A at the potential nominal completion time of A if it were to be scheduled next. This impact estimate enables us to approximate the impact metrics at completion times not observed in schedules of prior iterations. At each scheduling step we need to compute impact estimates for all enabled tasks. The IHRIF heuristic then ranks enabled tasks based on the highest impact estimate. At each scheduling step, the task with the highest impact estimate is scheduled.

Definition 5 (Impact Estimate) *The impact estimate of a task A, used during scheduling, is a function which maps a potential nominal completion time of A to the estimated impact metric at that time using the current impact history. It is denoted by the function $IE_A : \mathbb{R}^{\geq 0} \to \mathbb{R}^{\geq 0}$.*

We can derive impact estimates at any point in time from the points in the impact history. Since the impact computation performed during robustness analysis of a schedule is a random process, we can have multiple impact metric values for the same nominal completion time. These values can arise if we compute the impact of a task the position of which has not changed in multiple iterations. This could also happen if two different schedules result in the same nominal completion time for the task. To smoothen out the effects of these multiple values, we use a weighted distance average computation to compute impact estimate from the impact history. If CH_A is the set of all nominal completion times of A in the impact history and K the number of iterations

so far, then the following equation is used to compute IE_A at time t from its impact history.

$$IE_A(t) = \begin{cases} \text{avg}\{right(IH_A(k)) | left(IH_A(k)) = min(CH_A)\} & t < min(CH_A) \\ \text{avg}\{right(IH_A(k)) | left(IH_A(k)) = max(CH_A)\} & t > max(CH_A) \\ \text{avg}\{right(IH_A(k)) | left(IH_A(k)) = t\} & t \in CH_A \\ \frac{\sum_{k \in K} \frac{\Delta_k}{d_k} \cdot right(IH_A(k))}{\sum_{k \in K} \frac{\Delta_k}{d_k}} & \text{otherwise} \end{cases}$$

where $d_k = |left(IH_A(k) - t|$ is the distance from t to $left(IH_A(k))$ and $\Delta_k = \frac{left(IH_A(k+1)) - left(IH_A(k-1))}{2}$ denotes the range of values covered by the k^{th} entry. Multiplying the weight by Δ_k ensures that having multiple completion time entries close to each other does not bias the impact estimate in that region.

2) *Scheduling history (fallback choices)*: The impact history allows us to improve the estimation of the impact metrics used in the scheduling heuristic. Aside from this, we also want to improve the scheduling decisions by maintaining the *scheduling history*. This involves keeping the scheduling decisions of the best prior schedule in order to steer the search process of the heuristic towards global robustness improvement. Scheduling decisions are defined below.

Definition 6 (Scheduling Decisions) *Scheduling decisions of a scheduler, denoted by the function $SD : T \to \mathbb{N}$, is a function which maps a task to the step number at which the scheduler schedules the task.*

Maintaining scheduling history implies that we base our fallback choices on the *scheduling decisions* made in the most globally robust schedule observed so far in all prior iterations. This is the schedule with the lowest expected number of deadline misses. Fallback choices occur when all tasks in the enabled set have the same impact estimate. The reason for basing the fallback choices on the scheduling decisions of this schedule is to utilize the knowledge that these decisions have previously led to a globally more robust schedule.

3) *Impact Threshold*: An additional aspect of the heuristic is to limit the changes in the schedule to high impact tasks of each iteration. The rationale is based on the fact that the impact history is derived from schedules observed in prior iterations. If drastic changes are made early on during scheduling for low impact tasks, the schedule being formed becomes too different. This renders the impact estimate inaccurate for the subsequent high impact tasks which become visible to the scheduler only at the later scheduling steps. Therefore, we introduce an *impact threshold*, denoted by *it*, such that only impact values above a certain threshold are considered during scheduling. For tasks with impact values below the threshold, the scheduler uses the scheduling history explained earlier. This threshold is computed as a certain percentage, called *impact threshold percentage itp*, of the highest impact metric observed in the last iteration. By doing this the low impact tasks in a current iteration become visible above the threshold only in the later iterations, when the scheduler has already dealt with the current high impact tasks.

Algorithm 1: IterativeRobustScheduler()

Input : $G := (T, D), R, N_I, thp$
Output: S

1 $[S_0, SD_0] :=$ ListScheduler$_{\text{EDDF}}(G, R)$;
2 $IH = \phi$;
3 $[E[X_0], IH] :=$ RobustnessAnalysis(S_0, IH);
4 $th := \frac{thp}{100} \cdot \max_{A \in T} right(IH_A(0))$;
5 $S_{Best} := S_0$, $SD_{Best} := SD_0$, $i := 0$;
6 **repeat**
7 $\quad [S_i, SD_i] :=$ ListScheduler$_{\text{IHRIF}}(G, R, IH, it, SD_{Best})$;
8 $\quad [E[X_i], IH] :=$ RobustnessAnalysis(S_i);
9 $\quad th := \frac{thp}{100} \cdot \max_{A \in T} right(IH_A(i))$;
10 $\quad$ **if** $E[X_i] < E[X_{i-1}]$ **then**
11 $\quad\quad S_{Best} := S_i$, $SD_{Best} := SD_i$;
12 $\quad$ **end**
13 $\quad$ i:=i+1;
14 **until** $i \leq N_I$ *or* $S_i = S_{i-1}$;
15 **return** S_{Best};

7. ITERATIVE LIST SCHEDULING WITH IHRIF HEURISTIC

Algorithm 1 gives the overall iterative robust scheduler. N_I is the number of allowed iterations, SD_i the i^{th} iteration scheduling decisions, ListScheduler$_{\text{EDDF}}$() the list scheduler with EDDF heuristic, ListScheduler$_{\text{IHRIF}}$() the list scheduler with IHRIF heuristic, and RobustnessAnalysis() is the robustness analysis step. Line 1 produces an initial schedule using ListScheduler$_{\text{EDDF}}$() and Line 3 performs RobustnessAnalysis(). From the impact metrics obtained, impact threshold is computed in Line 4. We iteratively perform ListScheduler$_{\text{IHRIF}}$() and RobustnessAnalysis() in the loop between Lines 6-14 until stop criterion of Line 14. Line 9 recomputes impact threshold for the next iteration. If the schedule produced is better than previous best, Line 10-12 updates the best schedule and scheduling decisions. Line 15 returns the best schedule. Note that by construction the best result S_{Best} cannot be worse than the EDDF schedule.

For complexity analysis of the iterative algorithm, we consider its 3 major components: (1) ListScheduler$_{\text{EDDF}}$(), (2) ListScheduler$_{\text{IHRIF}}$() and (3) RobustnessAnalysis(). ListScheduler$_{\text{EDDF}}$() includes topological sorting of tasks and computation of due-dates. Topological sorting is linear in the number of tasks and dependencies. Due-date computation uses an efficient linear approach requiring one backward traversal of the graph. ListScheduler$_{\text{IHRIF}}$() differs only in the scheduling heuristic involving the computation of the impact estimate from the impact history, whose entries depends on the number of iterations. Since we need to repeat the computation for every task in the enabled set along with obtaining their potential completion times, this heuristic has a quadratic complexity of $O(N_I \cdot (|T| + |D|)^2)$. RobustnessAnalysis() obtains completion time distributions of tasks by linear max-plus computation of completion time bounds and a limited number of simulations, N_{Sim}. This requires the traversal of the schedule N_{Sim} times and the complexity is $O(N_{Sim} \cdot (|T| + |D|))$. There is a trade-off here between the number of simulations and accuracy of the analysis which can be exploited to reduce the scheduler run-times. RobustnessAnalysis() additionally performs the impact metric

computation per task as a product of two factors in Equation 5. Zero slack probability can be computed in one backward traversal through the graph. The second factor, probability density at the deadlines, takes constant time independent of graph size. Overall, since the impact metric of each task needs the sum of values obtained from all dependent tasks, this computation is quadratic in the number of tasks and dependencies. Lastly computation of expected deadline misses involves obtaining the deadline miss probabilities of constituent tasks and summing them up. The computation of these cumulative probabilities takes constant time independent of graph size.

8. EXPERIMENTAL RESULTS

In this section, we evaluate the iterative robust scheduler on real world applications and multiple synthetic test sets.

8.1 Real world applications

Video Conferencing Applications. With current advancements in video streaming, video conferencing on applications like Skype has become an integral part of daily life. Video and audio decoders are software components that convert a compressed video or audio into raw uncompressed form. The rate at which the decoders decompress frames has a direct impact on the quality of the video conference experience. The end to end delay can be seen as a latency requirement on the end task of the decoders. Missing these latency requirements results in jitter and failures experienced during the conference calls. We tested the robust scheduler on an application consisting of two audio and two video decoders. The graphs and the distributions are taken from documented Scenario Aware Dataflow Graphs (SADF) of video and audio decoders [20]. The graph of the four SADF combined consists of 1982 tasks, each performing a software function, mapped onto a platform consisting on three general purpose processors. A common deadline is assigned to the end tasks of all the four decoders. The expected value of deadline misses from the EDDF schedule is 0.53, consisting of the end tasks of the two video decoders having a deadline miss probability of 0.24 and 0.29 respectively. Starting from this schedule, the robust scheduler produces a schedule with an expected value of deadline misses equal to 0.11 using a threshold of 10% and 10 iterations. In this schedule, the end tasks of the video decoders have a deadline miss probability of 0.05 and 0.06 respectively. This is a significant robustness improvement of 79% experienced during the conference calls. The scheduler run-time was 4.25 minutes with 100 simulations in the robustness analysis.

A variant of the above application includes three video decoders with 603 tasks on a platform consisting of two homogenous general purpose processors. We assign fewer resources to utilize the parallelism in the application. Tight deadlines result in an expected value of deadline misses of 2.39 in the EDDF schedule. The robust scheduler yields a schedule with an expected value of 2.16 with a threshold of 10% and 10 iterations which is around 10% improvement in robustness. The scheduler ran for 1.33 minutes.

Lithography Machines. We have applied our scheduler on a system of ASML, the world's leading provider of lithography machines. These machines are highly complex and consist of a large number of control systems. We applied the robust scheduling approach to a critical control application for accurate movements and positioning in the machines. It has 4301 control tasks and 4095 dependencies, with a degree (number of dependencies of a task) ranging from 0 to 22, running with a frequency of 10kHz on a platform of 11 general purpose processors. The tasks range from simple addition operations to more complex clipping, filtering and matrix operations. Task execution time distributions are estimated from measurements made on the machine. Critical actuator tasks are assigned 30% of the processor budget as deadline corresponding to Input-Output delay constraints between sensing and actuation. The remaining tasks are assigned 70% to leave room for background processing. The expected number of tasks that miss deadlines in the EDDF schedule is around 458. A threshold percentage of 10% and 10 iterations gives a schedule with an expected value of deadline misses of around 445 which is approximately 3% robustness improvement. The scheduler runtime was 8.55 minutes.

8.2 Synthetic test cases

For further validation, we use multiple test sets consisting of 1000 DAGs generated using the random graph generator tool of SDF3 [19]. In the first test set, each graph consists of 100 tasks and each task has a degree ranging from 1 to 5 with an average of 2 and variance of 1. These tasks are bound to two to five resources such that highly interconnected sub-branches are mapped on the same resources to the maximum possible extent. This is to mimic the binding approach used in control applications wherein tasks in highly interconnected control loops are mapped on the same resources in order to minimize communication overhead.

Figure 6(a) compares the results obtained using the EDDF and the IHRIF scheduler. The horizontal axis represents the iterations in increasing order. The number of allowed iterations was chosen to be reasonably large at 10. The bottom portion of each vertical bar gives the number of graphs from 1000 that improved in robustness. The top portion gives the ones that showed no improvement. No improvement arises when IHRIF does not perform better than EDDF. In some of these cases the EDDF schedules may already optimally robust. We see an aggregated result of 76% more robust schedules at the end of 10 iterations. It can be shown that this is statistically a very significant result.

Since 6(a) gives aggregated results (taking the best of all schedules upto each iteration). From this, we cannot assess whether the individual iterations also produce significantly better results. In other words, we would also like to see if with more iterations, each iteration of the scheduler by itself also, on average, produces more robust schedules. From this we can deduce whether the schedules converge over iterations. Figure 6(b) compares each iteration of IHRIF individually with EDDF. Also per iteration, the number of more robust schedules gets significantly better in Figure 6(b). This validates our hypothesis that maintaining history in the iterative approach allows the scheduler to gather sufficient information over iterations to produce more robust schedules.

Figure 7 compares the expected number of deadline misses in schedules obtained from EDDF, the X-axis, and from IHRIF, the Y-axis. Points below the line correspond to cases where IHRIF improves over EDDF. The test set consists of a range of problems with low and high expected number of deadline misses from the EDDF schedules; we see improvement in most cases. The absolute improvement ranges from 0 to 6 tasks. The relative improvement compared to EDDF

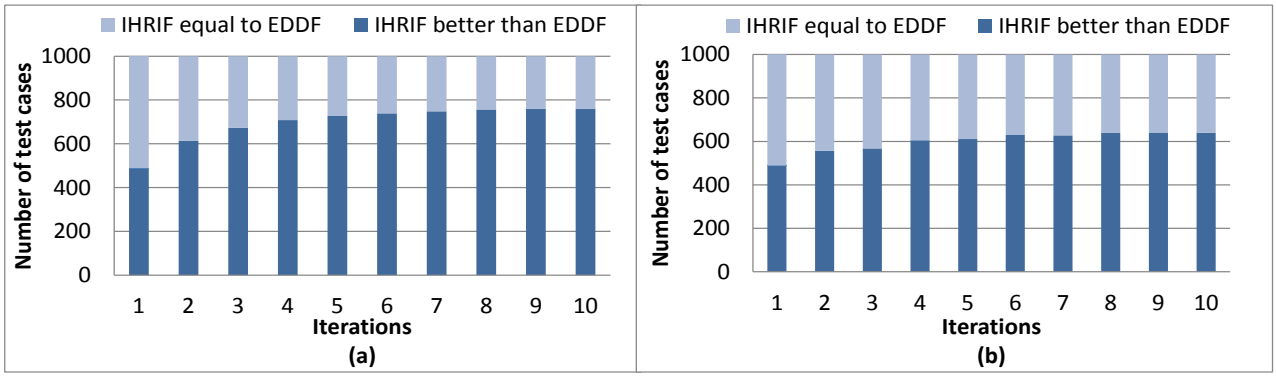


Figure 6: (a) Aggregated, and (b) per iteration schedule robustness improvements of IHRIF over EDDF

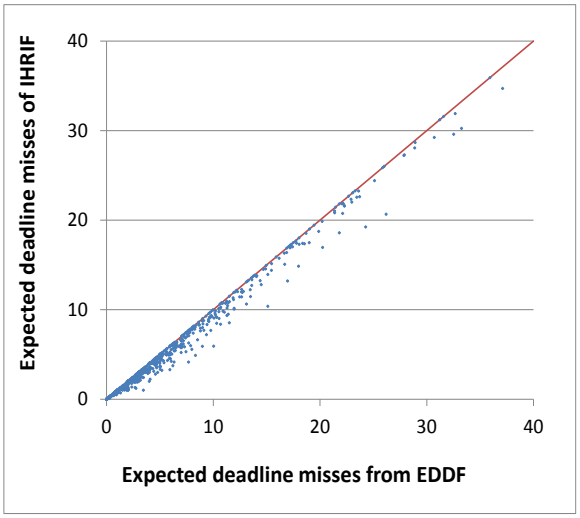


Figure 7: Comparison of the expected values of number of tasks that miss deadlines obtained from EDDF and IHRIF

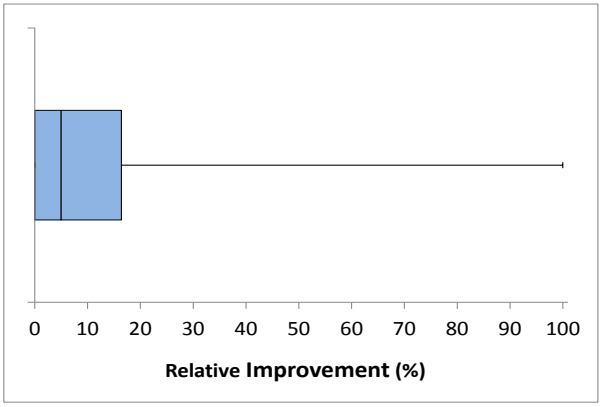


Figure 8: BoxPlot of relative improvement of IHRIF over EDDF

schedules ranges from 0-100% with an average of 12%, a median of 5% and a standard deviation of 18% (which is high due to some exceptional very high robustness improvement cases), see Figure 8. There is no tractable algorithm to find optimally robust schedules. Exhaustive state space search does not scale for DAGs with more than a few tasks. Hence, we cannot compare to optimal schedules. To test whether the approach is sensitive towards the binding mechanism used and to different task graph sizes, we also validated the approach on test sets with a first fit binding approach and on sets of larger task graph sizes with 500 and 4500 tasks. The results are consistent with the reported ones. The scheduler run-time for graphs of sizes 100 and 500 is tens of seconds per graph, using 1000 simulations in the robustness analysis. For larger graphs of size 4500, we use 100 simulations in robustness analysis and obtain run-times in the order of minutes.

9. DISCUSSION AND CONCLUSIONS

We presented an iterative robust scheduler that, starting from the schedule produced by a list scheduler using nominal execution times, iteratively improves the schedule robustness. The robustness analysis computes a newly proposed impact metric per task; the scheduler employs a new robustness heuristic called 'iterative highest robustness impact first'. The approach reduces the expected number of deadlines misses in a schedule thereby improving schedule robustness. The iterative mechanism is needed to extract sufficiently precise robustness information without excessive lookahead at each scheduling step. The scheduler has been validated on real world applications and has shown from 3% upto 79% robustness improvement. It has also been tested on 1000 synthetic graphs and has shown improvement in 76% of the schedules. Our future work aims to lift the fixed binding assumption towards dynamic binding which may further aid in the robustness improvement of schedules.

10. REFERENCES

[1] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *Real-Time Systems Symposium, 1998. Proceedings., The 19th IEEE*, pages 4–13, Dec 1998.

[2] S. Adyanthaya et al. Fast multiprocessor scheduling with fixed task binding of large scale industrial cyber physical systems. In *Digital System Design (DSD), 2013 Euromicro Conference on*, pages 979–988, 2013.

[3] S. Adyanthaya et al. Robustness analysis of multiprocessor schedules. In *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV), 2014 International Conference on*, pages 9–17, 2014.

[4] S. Ali et al. Measuring the robustness of a resource allocation. *Parallel and Distributed Systems, IEEE Transactions on*, 15(7):630–641, 2004.

[5] L. Boloni and D. C. Marinescu. Robust scheduling of metaprograms. *Journal of Scheduling*, 5(5):395–412, 2002.

[6] C. E. Clark. The greatest of a finite set of random variables. *Operations Research*, 9(2):145–162, 1961.

[7] R. L. Daniels and J. E. Carrillo. β-robust scheduling for single-machine systems with uncertain processing times. *IIE Transactions*, 29(11):977–985, 1997.

[8] A. Davenport, C. Gefflot, and C. Beck. Slack-based techniques for robust schedules. In *Sixth European Conference on Planning*, 2014.

[9] R. I. Davis and A. Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv.*, 43(4):35:1–35:44, Oct. 2011.

[10] D. England, J. B. Weissman, and J. Sadagopan. A new metric for robustness with application to job scheduling. In *HPDC*, pages 135–143. IEEE, 2005.

[11] F. Fauberteau et al. Robust partitioned scheduling for real-time multiprocessor systems. In *Distributed, Parallel and Biologically Inspired Systems*, pages 193–204. Springer, 2010.

[12] D. Hardy and I. Puaut. Static probabilistic worst case execution time estimation for architectures with faulty instruction caches. In *Proceedings of the 21st International Conference on Real-Time Networks and Systems*, RTNS '13, pages 35–44, New York, NY, USA, 2013. ACM.

[13] T. C. Hu. Parallel sequencing and assembly line problems. *Operations Research*, 9(6):841–848, 1961.

[14] K. Li et al. Scheduling precedence constrained stochastic tasks on heterogeneous cluster systems. *Computers, IEEE Transactions on*, 64(1):191–204, Jan 2015.

[15] M. Lombardi et al. Robust scheduling of task graphs under execution time uncertainty. *Computers, IEEE Transactions on*, 62(1):98–111, 2013.

[16] S. Nadarajah and S. Kotz. Exact distribution of the max/min of two gaussian random variables. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 16(2):210–212, 2008.

[17] R. Schiffelers, W. Alberts, and J. Voeten. Model based design, analysis and synthesis of servo controllers for lithoscanners. MPM, 2012.

[18] V. Shestak et al. Stochastic robustness metric and its use for static resource allocations. *J. Parallel Distrib. Comput.*, 68(8):1157–1173, 2008.

[19] S. Stuijk et al. SDF^3: SDF For Free. In *Application of Concurrency to System Design, 6th International Conference, ACSD, Proceedings*, pages 276–278, 2006. [Online]. Available: http://www.es.ele.tue.nl/sdf3.

[20] B. Theelen. A performance analysis tool for scenario-aware streaming applications. In *Quantitative Evaluation of Systems, 2007. QEST 2007. Fourth International Conference on the*, pages 269–270, Sept 2007.

[21] D. Wang et al. Estimating deadline-miss probabilities of tasks in large distributed systems. In *Advances in Grid and Pervasive Computing*, pages 254–263. 2012.