Partitioned Scheduling with Safety-Performance Trade-Offs in Stochastic Conditional DAG Models

Xuanliang Deng*, Ashrarul H. Sifat*, Shaoyu Huang, Sen Wang
Jiabin Huang, Changhee Jung, Ryan Williams, Haibo Zeng

[a] *Virginia Tech*
[b] *University of Maryland*
[c] *Purdue University*

Abstract

This paper is motivated by robotic systems that solve difficult real-world problems such as search and rescue (SAR) or precision agriculture [1]. These applications require robots to operate in complex, uncertain environments while maintaining safe interactions with human teammates within a specified level of performance. In this paper, we study the scheduling of real-time applications on heterogeneous hardware platforms inspired by such contexts. To capture the *stochasticity* due to unpredictable environments, we propose the stochastic heterogeneous parallel conditional DAG (SHPC-DAG) model, which extends the most recent HPC-DAG model in two regards. First, it uses conditional DAG nodes to model the execution of computational pipelines based on *context*, while the stochasticity of DAG edges captures the uncertain nature of a system's environment or the reliability of its hardware. Second, considering the pessimism of deterministic worst-case execution time (WCET), it uses *probability distributions* to model the execution times of subtasks (DAG nodes). We propose a new partitioning algorithm *Least Latency Partitioned (LLP)*, which considers precedence constraints among nodes during the allocation process. Coupled with a scheduling algorithm that accounts for varying subtask criticality and constraints, the end-to-end latencies of safety-critical paths/nodes are then minimized. We use tasksets inspired by real robotics to demonstrate that our framework allows for efficient scheduling in complex computational pipelines, with more flexible repre-

[1]This work was supported by NSF under Grant CNS-1932074. Haibo Zeng is the corresponding author. Email: hbzeng@vt.edu

sentation of timing constraints, and ultimately, safety-performance tradeoffs.

Keywords: DAG scheduling, stochastic scheduling, safety-critical timing constraints

1. Introduction

Modeling and scheduling real-time applications on heterogeneous hardware platforms has received significant attention recently. Due to the increasingly complex functionalities of emerging real-time applications (such as human-robot search and rescue (SAR) or precision agriculture [1, 2]), fully utilizing the computing power of heterogeneous hardware becomes an important problem. Among the modeling approaches, Directed Acyclic Graphs (DAGs) have become a popular method of late. In DAGs, subtasks and data transmissions are represented as nodes and edges respectively, which can capture the dependency between tasks in a straightforward manner [3]. The Heterogeneous Parallel Conditional DAG (HPC-DAG) model, which is the most expressive DAG task model to date, further accounts for the fact that a subtask may only execute on certain types of hardware [4]. However, HPC-DAGs *do not consider the stochasticity that an application may encounter in real-world operations* [4, 5, 6, 7]. We propose the stochastic HPC-DAG model (SHPC-DAG), where nodes and edges are associated with probabilities to capture the uncertainties at run-time.

Another enhancement in the SHPC-DAG model is the use of probabilistic execution time for each subtask. The majority of scheduling algorithms utilize the worst-case execution times (WCETs) of tasks for analyzing schedulability, which can be overly pessimistic [8]. In addition, the execution times of tasks are often associated with some uncertainty before run-time and may vary due to environmental dynamism and varying sensor inputs [9, 10]. It is then natural to account for the uncertain nature of a system's environment and/or the reliability of its hardware, by representing the execution times of subtasks as random variables and optimizing the expectation of the objective function [9]. However, existing studies on DAG scheduling that model the execution times as random variables [9, 11, 12] all assume that the random execution times follow some well-known probability distributions such as Gaussian or Weibull distributions. Although this assumption allows one to exploit the mathematical properties of the particular distribution to simplify analysis, such distributions do not always hold in real-world applications. For

example, if robots are assigned distinct behaviors when they are operating indoors versus outdoors (as in our motivating SAR problem), the execution time of the same task may vary significantly due to environmental dynamism, thus making its probability distribution discrete rather than continuous and difficult to fit by Gaussian or Weibull functions. Another important example has been demonstrated in [13], where the probability distributions of natural image data are shown to be long-tailed, and thus non-Gaussian. Thus, we do not assume a specific type of probability distribution to characterize the random execution times of subtasks.

For the proposed SHPC-DAG system model, we develop a novel partitioned scheduling algorithm called Least Latency Partitioned (LLP) scheduling consisting of a node-to-processor allocation strategy and a scheduling algorithm. Compared with the heuristics proposed in [6, 14], our method utilizes the precedence constraints among subtasks to select the proper processor for allocation, aiming at maximal parallel execution and minimal end-to-end latency. Importantly, our algorithm has polynomial time complexity in the size of DAG. The Safety-Performance (SP) metric from the study [15] enables a holistic mission objective in safety-critical systems. The SP metric ensures safety first and then utilizes remaining timing slack to maximize performance, capturing the idea that additional timing slack may not necessarily yield a safer system. For robotic systems with complex mission objectives in dynamic environments, the system may not always need to prioritize safety concerns, especially when no human is involved. Instead, broader mission objectives that also take into account performance may better describe the system needs, which leads to the key idea of SP metric to capture such safety-performance trade-offs. Our proposed scheduling algorithm utilizes such metrics in both the allocation and scheduling processes.

This paper studies the stochastic version of the restricted assignment scheduling problem (RASP) [5]. A specific model with similar assumptions has also been posed by the 2021 edition of the Real-time Systems Symposium (RTSS) Industry challenge. To the best of our knowledge, this paper is the first to consider the stochastic version of the RASP problem (S-RASP) and makes the following contributions.

- We introduce a novel SHPC-DAG task model that captures the stochasticity of the uncertain environment.
- We propose a novel partitioned scheduling algorithm LLP to optimize the SP metric. It leverages the precedence constraints among subtasks

to maximize parallel execution. We show the algorithm has polynomial time complexity in the size of the DAG.

- The proposed method LLP outperforms the scheduling algorithm in HPC-DAG [6] under hard timing constraints, and performs even better when changing hard timing constraints to probabilistic constraints. Our simulation results show an improvement of up to 50% in the acceptance ratio on random synthetic DAG tasksets.

The paper is organized as follows. Section 2 describes the proposed SHPC-DAG model. Section 3 gives a brief introduction to the SP metric. Section 4 and 5 illustrate the proposed allocation strategy and scheduling algorithm. Section 6 evaluates the proposed algorithms. Section 7 reviews the related work. Finally, Section 8 concludes our paper with future research directions.

2. System Model

2.1. Motivating Applications

Many applications of real-time systems, such as autonomous robots, operate in changing environmental conditions (called *contexts*) [16, 17, 18]. To clarify our work, we consider the following definition:

Definition 2.1. (Context ($\mathcal{CTX}$)). A context of operation for a real-time system is a set of detectable environment features that are associated with a set of computational tasks and timing requirements. Each distinct context (including the unknown context) is associated with an integer index that takes values in a finite set,

$$ctx \in \mathcal{N}, \mathcal{N} = \{0, 1, 2, \dots\} \tag{1}$$

Contexts are required system behavior (computation and timing requirements) based on the specific nature of the operational environment. We argue that although operational environments can contain various uncertainties and unknowns (especially for robots) as well as continuous inputs to a system that are not easily quantizable, for many applications there still remain detectable environment features that dictate a system's desired behavior and timing requirements. For instance, when a mobile robot traverses an environment with humans nearby, the responsiveness of vision-based detection and robot motion controllers is critical as collisions with humans are

unacceptable [19]. On the other hand, a recent trend in robotics is that not all collisions are bad [20], and thus if a robot is operating with no human presence the requirements on vision and motion responsiveness can be relaxed. In this example, the vision-based detection of nearby humans (e.g., see [21]) indicates the nature of the operational environment, and from this (potentially imperfect) detection the required system behavior can be set. We aim to capture such relationships in the models and algorithms proposed in this work.

Safety standards for software development in safety critical systems vary in different domains [22]. Development of software for general safety critical systems usually follows IEC 61508 [22, 23], which performs risk analysis and specifies the appropriate safety integrity level (SIL) for each function. Another example is the RTCA DO-178B that is applied to airborne software development [24]. In this work, we define the notion of time sensitivity level (TSL), which refers to the safety levels of subtasks and reflects any priorities that may exist between functionalities and subsystems under a specified safety standard.

Definition 2.2. (Time sensitivity level (TSL)). TSL is the criticality of the subtask regarding its functionality, determined by software safety standard and application requirements. It takes values in the set of positive integers,

$$tsl \in \mathcal{Z}^+, \mathcal{Z}^+ = \{1, 2, 3, ...\} \tag{2}$$

According to application-dependent criticalities, TSLs of all subtasks help to form an *ordering* in the taskset. The larger the TSL, the earlier the subtask should be guaranteed to schedule for meeting application requirements. For example, subtasks along a computational path for pedestrian detection would certainly be scheduled as early as possible for autonomous applications [25], without violating precedence constraints. Note that if the criticalities of the subtasks (i.e., functionalities) are not specified in the application requirements and safety standards, we set all TSLs to the same values by default.

2.2. Assumptions

We state our assumptions first for better clarity. We study a single-rate DAG with the event-driven activation model, which assumes the same activation periods for all subtasks in the DAG [26]. Contrary to the multi-rate DAG in [27] where all subtasks activate periodically and independently, the event-driven activation model is a single-rate model where subtasks are

activated simultaneously and execute in the order constrained by data dependencies. Therefore, the intra-task interference could be eliminated as in single-rate DAG, each subtask is executed only once by following the precedence constraints. In [27], a method is proposed to convert a multi-rate DAG with timing constraints to a single-rate DAG. Specifically, each subtask executes a number of jobs within one hyper-period according to its period, and synchronization constructs are inserted in the DAG to guarantee that the original multi-rate DAG and the converted single-rate DAG have equivalent temporal requirements. Due to the existence of this conversion [27], our results for single-rate DAG could be applied to multi-rate DAGs without loss of generality. Finally, we assume DAG subtasks are executed *non-preemptively*.

2.3. Stochastic Heterogeneous Parallel Conditional DAGs

We propose a new DAG task model, the *Stochastic Heterogeneous Parallel Conditional Directed Acyclic Graph (SHPC-DAG)*, which incorporates both timing and resource constraints for safety-critical systems. Our observation is that computational pipelines may be executed conditionally due to uncertainty, recognizing that not all outcomes of computational subtasks can be perfectly predicted in real-world applications (e.g., extreme events). Also, the execution times of tasks may not be perfectly known before runtime and may vary significantly *online* due to environmental dynamism and conditions of the hardware platform. Scheduling according to the worst-case execution time (WCET) may be overly pessimistic. Hence, the SHPC-DAG model contains two significant extensions compared to the most recent HPC-DAG model. The first is that the execution time of a computational subtask is assumed to follow a probability distribution rather than the WCET. The second is that stochastic conditional nodes and probabilities associated with their outgoing edges are added to model computational path selection.

Definition 2.3. A real-time application can be modeled by a *SHPC-DAG*, denoted as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, f, Type, Tag, TSL)$, where:

- $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ is the set of nodes in the graph. Each of the n nodes $v_i \in \mathcal{V}$ is a subtask which represents a block of code that executes on a specific type of processor. Hence, there are n subtasks in the application.
- $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges among subtasks that indicates the data dependencies, with associated probabilities indicating the likelihood of

edge traversal during execution. This will be introduced in the next subsection along with the stochastic conditional node.

- $f = \{f_1, f_2, \ldots, f_n\}$ is the set of probability distribution functions that characterize the possible values of the subtasks' execution times. Each f_i can either be discrete (i.e., probability mass function (PMF)) or continuous (i.e., probability density function (PDF)). For both PMF and PDF, the inputs are the random execution times of subtasks. For PMF, the execution times take discrete values in the set of positive integer numbers $Z^+ = \{1, 2, 3, ...\}$. Each discrete execution time will be associated with a probability (i.e., the function output) and the sum of all probabilities equal to one. For PDFs, the random execution times take continuous values in a certain range with lower and upper bounds. The integration of probability density over this range equals to one.

- $Type = \{type_1, type_2, \ldots, type_n\}$ is the set of operation types of all nodes in $\mathcal{V}$. This may include stochastic *Conditional* nodes, for which the formal definition is given in section 2.4. For applications such as autonomous driving, other potential node types may include *Sensor* and *Sync*, which represent the sensor inputs and synchronization constructs of the application, respectively.

- $Tag = \{tag_1, tag_2, \ldots, tag_n\}$ indicates the type of processors that each task in $\mathcal{V}$ should run on, e.g., CPU, GPU, etc.

- $TSL = \{tsl_1, tsl_2, \ldots, tsl_n\}$ is the set of TSLs for all nodes in the graph.

Lastly, we adopt the formal definitions of safety-critical paths and nodes in [15] to equip SHPC-DAG with safety-critical features.

Definition 2.4. (Safety-critical path [15]). A safety-critical path consists of a chain of computational tasks that together achieve a critical functionality/feature regarding the safety aspect of the application (also known as a critical cause-effect chain).

Let $\mathcal{C}$ denote the set of DAG paths that are safety-critical where each path is indexed as ℓ_i, $\ell_i \in \mathcal{C}$. There may exist many paths in a SHPC-DAG but not all of them are critical to the safety aspect of the application. The determination of safety-critical paths is highly dependent on the application functionality and requirements.

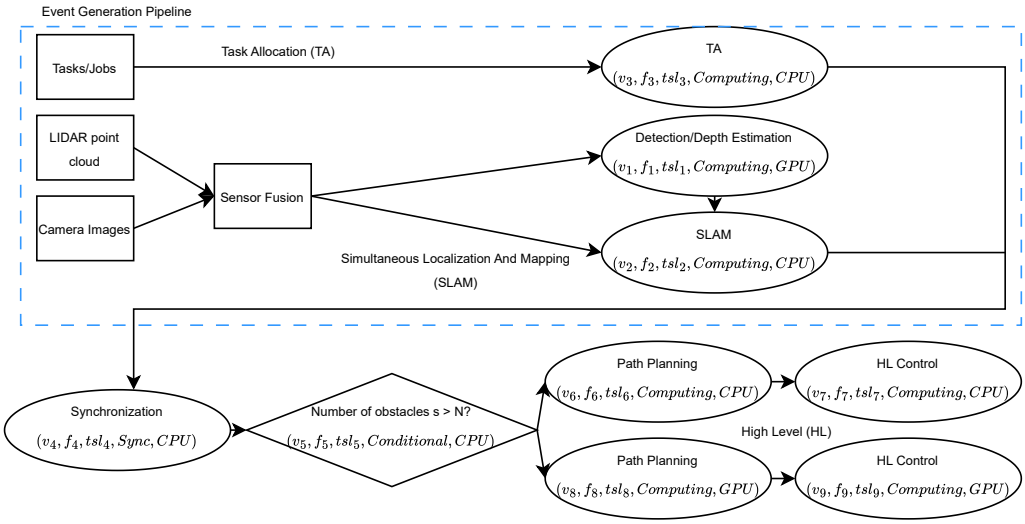


Figure 1: An illustrative example of our *SHPC-DAG* model with typical computational kernels and precedence constraints for subsystems seen in autonomous driving or robotic search and rescue.

Definition 2.5. (Safety-critical node [15]). Safety-critical nodes are subtasks along computational paths that achieve some safety-critical functionality of the application.

$$s = \begin{cases} True, \text{safety-critical node} \\ False, \text{non-safety-critical node} \end{cases}$$

Note that the deadline of a safety-critical node may be independent from the timing requirements of the paths it is associated with. For some autonomous applications, meeting the deadlines of the application or computational path alone cannot guarantee the safety in real operation. For example, in Simultaneous Localization and Mapping (SLAM), mapping should always be updated on time to ensure that subsequent tasks operate with a correct map of the environment, which is critical for safety [28]. Thus, the mapping subtask itself will have an individual deadline in addition to the deadline of the entire computational path. As in the above definition, an additional set of node attributes $S = \{s_1, s_2, \ldots, s_n\}$ with boolean values are used to indicate whether the nodes are safety-critical or not. The set of safety-critical nodes is denoted as $\mathcal{V}_c$, and we have $\mathcal{V}_c \subseteq \mathcal{V}$.

An illustrative system modeled by the proposed DAG framework is given in Figure 1. Such a computational setup may be seen as a subsystem in, for example, autonomous driving or robotic search and rescue. Robotic systems are equipped with sensors (e.g., cameras, lidars), which take measurements of the surrounding environment. The sensors' measurements could be asynchronous due to different sample rates. In the Robot Operating System (ROS) framework, it provides several configurable sensor fusion packages that can handle asynchronous multi-rate measurements [29], which is denoted as the sensor fusion node in Figure 1. The combined inputs are then passed to the computational subtasks (e.g., path planning, localization and mapping) for further processing. Since the whole subsystem about path planning is crucial for safety problems in autonomous driving, all the path planning subtasks in both branches are labeled as safety-critical nodes. Based on the environment that is of a cluttered, uncertain nature, the conditional node (denoted as a diamond box in the figure) selects which computational path is used for path planning. We explain that in the next subsection.

2.4. Stochastic Conditional Nodes in DAG Models

SHPC-DAG contains a novel stochastic conditional node structure to capture the stochastic nature of real-time applications during run-time. Such type of nodes enables the selection of computational paths in a DAG based on *contexts* that occur under uncertainty. Let m denote the number of possible contexts that can occur during execution (e.g, environmental conditions), and let $\mathcal{CTX} = \{ctx_1, ctx_2, \ldots, ctx_m\}$ denote the set of all contexts. The specific context that the application will have during execution is unknown before run-time. During execution, the context will be determined by real-time information (e.g., sensor inputs) and the embedded detector. In most existing works, they consider the context returned by the detector as a single deterministic context across $\mathcal{CTX}$. However, due to the imperfectness of the detector, it may misjudge the current context. To consider such case, we propose to use a probability distribution to represent the probability of occurrence of all possible contexts rather than a single deterministic result. The attributes (e.g., execution time, processor tag) of the nodes in the *SHPC-DAG* will depend on the corresponding context.

Definition 2.6. (Probability Distribution of Context Detection) The detected context of the real-time application will follow a probability distribution $P(ctx),\ ctx \in \mathcal{CTX}$ based on real-time information. The sum of proba-

bilities associated with all possible contexts will be 1, i.e., $\sum_{i=1}^{m} P(ctx_i) = 1$, where m is the number of distinct contexts.

Let k denote the number of all possible branches (i.e., outgoing edges) that a stochastic conditional node has and $\mathcal{B} = \{b_1, \ldots, b_k\}$ denote the set of all branches. Each branch is associated with a distinct context ctx. We assume $k \leq m$ to make sure that each branch represents a distinct context and execution path that will not duplicate. During execution the stochastic conditional node will select only one branch to execute with probability $P(b_i)$, thus $\sum_{i=1}^{k} P(b_i) = 1$.

The probability that a real-time application executes along a specific execution branch depends on the detected context during run-time. When receiving the input data from sensors, the embedded detector may determine the current context is ctx_i. However, due to the imperfectness of context detection, the actual context could be ctx_j $(j \neq i)$. This stochasticity needs to be considered when calculating the probability $P(b_i)$ of selecting each branch.

Definition 2.7. (Probability for Each Branch of Conditional Node). Suppose a stochastic conditional node has k outgoing branches under the assumption of m possible contexts in total and $k \leq m$. The branch b_i will be selected only if the detected context ctx_j satisfies the *condition* of branch b_i (determined by the conditional statement in conditional node). The probability $P(b_i)$ of selecting branch b_i is

$$P(b_i) = \sum_{j=1}^{m} P(b_i|ctx_j) \cdot P(ctx_j) \tag{3}$$

which should satisfy

$$\begin{cases} \sum_{i=1}^{k} P(b_i) = 1 \\ \sum_{j=1}^{m} P(ctx_j) = 1 \end{cases}$$

Note that the $P(b_i|ctx_j)$ term will be determined by the accuracy of the detector[2]. The classification of the context is conducted by the embedded

[2]In cases where perfect context knowledge exists, our definition for the stochastic conditional node collapses to a classical conditional node with fixed probabilities associated with each outgoing edge, as in [10].

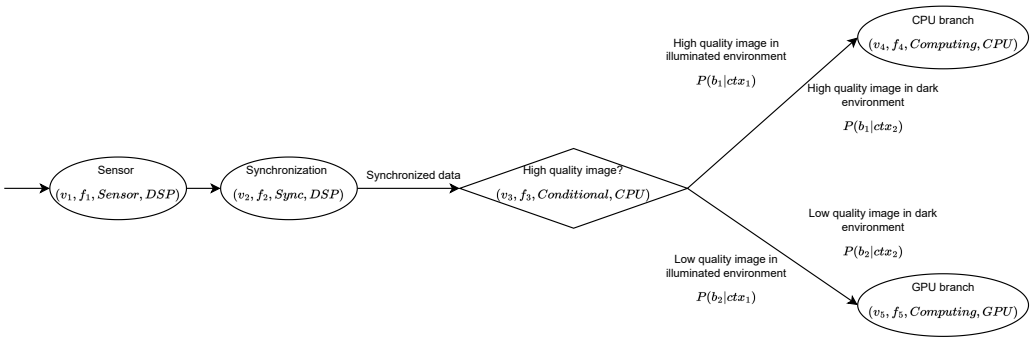


Figure 2: An illustrative example of the structure of our proposed stochastic conditional node.

detector in robots [21].

Figure 2 illustrates the idea of stochastic conditional node considering two possible contexts: illuminated environment (i.e., ctx_1) and dark environment (i.e., ctx_2). The conditional statement embedded in the conditional node determines whether the data collected is of high quality or low quality (e.g., the collected images are clear or not), and selects execution branch accordingly. We consider the stochasticity by associating edges of conditional nodes with conditional probabilites. For example, it is highly likely that robot could collect clear images of surroundings in a well-illuminated environment (i.e., $P(b_1|ctx_1)$). However, the context detector is imperfect. The result of detection could be a dark environment, but still with good quality images (i.e., $P(b_1|ctx_2)$). Either case will lead to the upper execution branch.

3. Safety-Performance Metric

Our proposed scheduling approach (Sections 4 and 5) requires an objective function to optimize when selecting appropriate schedules. While a typical function such as makespan can be used, we consider a metric that identifies safety-critical nodes and paths in our *SHPC-DAG*, allowing our scheduler to ensure safety wherever necessary and then exploit remaining timing headroom to maximize performance. Specifically, we exploit the Safety-Performance (SP) metric from the study [15], which defines a series of penalties/rewards based on violating/satisfying timing constraints in a DAG. In this work, the term *safety* refers to the concept of *nominal safety*, where the timing of software modules below predefined thresholds can contribute to

unsafe behaviors in autonomous systems. Its definition has been well cited in the autonomous vehicles (AV) area [30, 25].

Below we briefly summarize the definition of SP metric, starting with the calculation of the response time probability distributions for the nodes and paths in the graph model.

3.1. Response time probability distributions

When calculating the response time distribution of nodes and safety-critical paths, we need to consider the maximum and summation operations of independent random execution times characterized by probability distributions. For many data-driven projects in practice, the analytic form of the probability density function of execution time may not be available. Instead we have measurement data from experiment or simulation in different environments, i.e., a probability mass function (PMF) defined at discrete points. The subtask will be executed multiple times on the target hardware according to several distinct context of operations and measure execution times under each of the context [31]. Therefore, in this work we analyze the random variables characterized by PMFs.

Consider X and Y as two independent random variables with PMFs respectively. Let Z be the maximum of X and Y, i.e., $Z = \max(X, Y)$, S be the summation of X and Y. We define the operations as follows.

Maximum Operation. Consider a node in SHPC-DAG that has multiple predecessors (i.e., the nodes from which there exists a path to the current node) of which the response times are characterized as probability distributions. The earliest possible time instant of this node to start execution is determined by the maximum response times among all its predecessors, denoted as starting time (ST). From basic probability theory, the cumulative distribution function (CDF) $F_Z(z)$ of maximum of two independent random variables X and Y is given as,

$$\begin{aligned} F_Z(z) &= P(\max(X, Y) \leq z) = P(X \leq z, Y \leq z) \\ &= P(X \leq z)P(Y \leq z) = F_X(z)F_Y(z) \end{aligned} \tag{4}$$

Based on CDFs of X and Y, we could find the PMF of z by,

$$P(z) = F_X(z)F_Y(z) - F_X(z-1)F_Y(z-1) \tag{5}$$

where $z \geq 1$ and takes only integer values, this process terminates when $F_Z(z) = 1$ since $F_Z(z) = \sum_{z=1} P(z)$. For node with multiple predecessors (e.g., more than two), (4) and (5) can be extended in the same way.

Summation Operation. The density function of the summation $\mathcal{Z}$ of two independent random variables $\mathcal{X}$ and $\mathcal{Y}$ is the convolution of the density functions of the individual random variables [32, 33].

$$P(\mathcal{Z} = z) = \sum_{k=-\infty}^{\infty} P(\mathcal{X} = k)P(\mathcal{Y} = z - k) \tag{6}$$

For discrete convolution in the time domain, we have each probability mass represented as a data point in an array of length N. To speed up the calculation, the Fast Fourier transform (FFT) is applied to efficiently compute the above multiplication when N is a power of 2, yielding improved time complexity $O(Nlog(N))$. It is known that the convolution of two functions is the inverse Fast Fourier transform (IFFT) of the product of their individual Fourier transforms, therefore we could obtain the PMF of $\mathcal{Z}$ by taking the IFFT of the product.

Note that the response time distribution of safety-critical paths is not simply the linear summation of the nodes' execution times along the path. Instead, the precedence constraints in DAG and resource constraints should both be taken into account when analyzing the response time distributions, which is different from the work [32] that also considers tasks with probabilistic parameters.

Definition 3.1. (Starting time). The probability distribution of node v_i's starting time (ST) is determined by the maximum response time (RT) distributions, defined by (4) and (5), among all its predecessors and the nodes scheduled ahead of it on the same processor,

$$ST_i = \max_{v_j \in pred(v_i) \cup ahead(v_i)} RT_j \tag{7}$$

Definition 3.2. (Response time). The probability distribution of node v_i's response time (RT) is the convolution of its starting time (ST) and execution time (ET), both characterized by probability distributions, where * denotes the convolution operation

$$RT_i = ST_i * ET_i \tag{8}$$

The response time of a safety-critical path is equal to the response time of its ending subtask.

3.2. Safety-Performance Metric

To begin with, we give the formal definition of penalty in the SP metric when safety-critical paths and/or critical nodes violate their timing constraints [15].

Definition 3.3. For a given schedule $\mathcal{S}$, the penalty for safety-critical paths that violate their timing constraints is given by:

$$f_{\text{cp}}^{\text{p}}(\mathcal{S}) = \sum_{\ell_i \in \mathcal{C}_{\text{us}}} p_{\text{cp}}(P(R_{\ell_i} > \tau_{\ell_i}) - \lambda_{\ell_i}) \tag{9}$$

Here $\mathcal{C}_{\text{us}}$ denotes the set of safety-critical paths that violate a *probabilistic* timing constraint, i.e., $P(R_{\ell_i} > \tau_{\ell_i}) > \lambda_{\ell_i}$. The random variable R_{ℓ_i} denotes the response time of a critical path ℓ_i. τ_{ℓ_i} is the *nominally safe* response time (i.e., deadline) for path ℓ_i, and λ_{ℓ_i} is the probabilistic timing constraint for ℓ_i. $f_{\text{cp}}^{\text{p}}(\mathcal{S})$ represents a penalty term (p) for critical path violations (cp) based on schedule $\mathcal{S}$ with a generic penalty function $p_{\text{cp}}(\cdot)$.

Equation (9) can be interpreted as penalizing every violating critical path based on the *deviation* from its probabilistic timing constraint. This equation can also deal with systems that have hard timing constraint by setting $\lambda_{\ell_i} = 0$, which requires that the response time will always meet the nominally safe response time, i.e., $R_{\ell_i} \leq \tau_{\ell_i}$. If none of the safety-critical paths violate their timing constraints with schedule $\mathcal{S}$, the penalty $f_{\text{cp}}^{\text{p}}(\mathcal{S}) = 0$.

Definition 3.4. For a schedule $\mathcal{S}$, the penalty for the safety-critical nodes that violate their timing constraints is given by:

$$f_{\text{cn}}^{\text{p}}(\mathcal{S}) = \sum_{v_i \in \mathcal{V}_{\text{us}}} p_{\text{cn}}(P(R_{v_i} > \tau_{v_i}) - \lambda_{v_i}) \tag{10}$$

Here $\mathcal{V}_{\text{us}}$ denotes the set of safety-critical nodes that violate a *probabilistic* timing constraint, i.e., $P(R_{v_i} > \tau_{v_i}) > \lambda_{v_i}$. Similar to the terms in the definition of penalty for critical paths, R_{v_i}, τ_{v_i} and λ_{v_i} denote the response time, nominally safe response time, and probabilistic timing constraint of v_i, respectively. If there are no safety-critical nodes that violate their timing constraints based on schedule $\mathcal{S}$, then $\mathcal{V}_{\text{us}} = \emptyset$ and $f_{\text{cn}}^{\text{p}}(\mathcal{S}) = 0$, yielding no penalty.

Next, we introduce rewards gained when timing constraints for safety-critical paths are all satisfied. The reward term can be non-zero only when there exists no critical path or node that violates its timing constraints.

Definition 3.5 (*Path Reward*)**.** The reward for all paths, assuming safety-critical timing constraints are satisfied, is given by:

$$f^{\mathrm{r}}_{\mathrm{path}}(\mathcal{S}) = \Big[\sum_{\ell_i \in \mathcal{C}} r^{\mathrm{s},\ell_i}_{\mathrm{path}}(\lambda_{\ell_i} - P(R_{\ell_i} > \tau_{\ell_i})) + r^{\mathrm{p},\ell_i}_{\mathrm{path}}(R_{\ell_i}) \Big] + \sum_{\ell_i \notin \mathcal{C}} r^{\mathrm{p},\ell_i}_{\mathrm{path}}(R_{\ell_i}) \quad (11)$$

Definition 3.6 (*Node Reward*)**.** The reward for all nodes, assuming safety-critical timing constraints are satisfied, is given by:

$$f^{\mathrm{r}}_{\mathrm{node}}(\mathcal{S}) = \Big[\sum_{v_i \in \mathcal{V}_c} r^{\mathrm{s},v_i}_{\mathrm{node}}(\lambda_{v_i} - P(R_{v_i} > \tau_{v_i})) + r^{\mathrm{p},v_i}_{\mathrm{node}}(R_{v_i}) \Big] + \sum_{v_i \notin \mathcal{V}_c} r^{\mathrm{p},v_i}_{\mathrm{node}}(R_{v_i}) \quad (12)$$

Here $f^{\mathrm{r}}_{\mathrm{path}}(\mathcal{S})$ and $f^{\mathrm{r}}_{\mathrm{node}}(\mathcal{S})$ represent the reward term (r) for *all* paths and nodes respectively, regardless of whether they are safety-critical or not. The generic reward functions $r^{\mathrm{s},\ell_i}_{\mathrm{path}}(\cdot), r^{\mathrm{p},\ell_i}_{\mathrm{path}}(\cdot)$, $r^{\mathrm{s},v_i}_{\mathrm{node}}(\cdot), r^{\mathrm{p},v_i}_{\mathrm{node}}(\cdot)$ separately reward safety margins (s) and system performance (p) based on timing for path ℓ_i and node v_i.

Finally, with all terms defined, we design our scheduler to optimize the overall safety-performance metric defined as a weighted sum of terms (9)-(12). This yields a schedule that trade off safety and system performance relative to probabilistic timing constraints. An example for the penalty functions is a linear operator, e.g., $p_{\mathrm{cp}}(x) = \gamma x$, yielding a weighted penalty for timing constraint violation. For the safety reward functions, an example is a *sigmoid function*, e.g., $r^{\mathrm{s}}_{\mathrm{path}}(x) = 1/(1 + \exp^{-x})$, which models diminishing returns of rewards for increasing safety margins. For the non-critical, performance-based reward functions one could consider the right-hand tail of the response time distribution, e.g., $r^{\mathrm{s}}_{\mathrm{path}}(R_{v_i}) = P(R_{v_i} > x)$, with x defined as a response time threshold that yields higher quality computational output. One clear case of such a timing-performance relationship can be seen in the *traveling salesperson problem (TSP)*, where increasing computational time can allow for significantly improved TSP tours as branch-and-bound techniques can be used instead of greedy solutions [34]. Finally, it is important to note that instead of transforming the conditional DAG into an ordinary DAG (e.g., [35]), we directly analyze the conditional DAG on a *path-basis* for calculating the SP metric (details in Section 5). This avoids the cost of additional DAG transformation and helps focus on the conditional branching of safety-critical paths.

4. The LLP Scheduler: Subtask Allocation

We consider partitioned scheduling for the *SHPC-DAG* task model on heterogeneous hardware platforms. Partitioned scheduling has low run-time overhead due to the absence of thread migration compared to global scheduling [6, 36], and the proper partitioning (i.e., allocating subtasks onto the processors) is typically determined before scheduling [14]. In this section, we propose a novel partitioned scheduling algorithm called Least Latency Partitioned (LLP). It utilizes the proposed *SHPC-DAG* task model and SP metric to maximize the probabilistic reward of all safety-critical paths and nodes, while meeting their timing and performance requirements.

The existing work of HPC-DAG proposed two different allocation methods that allocate the subtasks on a minimal number of processors and schedule with Earliest Deadline First (EDF) algorithm on each processor [6]: Sequential Allocation (HPC-s) and Parallel Allocation (HPC-p). In HPC-s, subtasks with the same processor tags could be allocated on a single processor if the schedulability test is passed. If HPC-s fails, HPC-p will remove one subtask at a time and reallocate it on another available processor until the remaining subtasks could be schedulable on a single processor.

Our proposed LLP allocation method takes additional factors into account to maximize parallel execution of subtasks and to reduce end-to-end path response times. First, LLP avoids situations where processors remain idle during run-time when subtasks are ready to execute. Since we consider a non-preemptive model, the computing power of all available processors should be utilized to maximize parallel execution. Second, LLP considers the precedence constraints among subtasks during allocation, while HPC-DAG uses simple heuristics (i.e., best-fit (BF) or worst-fit (WF)) to determine the allocation. Third, in addition to precedence constraints, LLP further sorts the subtasks according to the their TSLs, while in HPC-DAG the orderings of subtasks that do not belong to the critical path (i.e., the path with the longest response time) are not specified.

The main idea of LLP is to utilize precedence constraints among subtasks and TSLs in both allocation and scheduling processes to reduce end-to-end latency and maximize the SP metric of safety-critical paths and nodes. The LLP allocation process can be divided into the following steps.

Assign time sensitivity levels (TSLs) to subtasks. The TSLs of subtasks are highly dependent on the application requirements and the applied

safety standards. Based on the functionality regarding the safety aspect of the application, each path will be assigned a TSL according to the specific safety standard (e.g., IEC 61508) that is applied to the system. For safety-critical nodes along the path, they will be assigned a higher TSL. Note that for subtasks that are predecessors of the ones along the safety-critical paths, we use the *backward propagation* to update their TSLs by inheriting the maximal TSL among all their successors. If the TSLs of the subtasks are not specified in the application requirements, we sort all the safety-critical paths in the *SHPC-DAG* by expected value of lengths in *decreasing* order. Subtasks along the safety-critical path with largest expected value of length will be assigned highest TSL. The subtasks along shorter safety-critical paths will be assigned incrementally smaller TSLs until all subtasks are traversed.

Partition SHPC-DAG based on tags. Based on the processor tags (e.g., CPU, GPU) associated with subtasks, we divide the SHPC-DAG into different subgraphs. Each processor tag will be associated with a set of subtasks and processors, respectively, that have the same tags. Subtasks will only be allocated to their corresponding set of processors.

Sort subtasks according to dependencies and TSLs. To respect the precedence constraints in *SHPC-DAG*, we use topological sort to obtain complete linear ordering(s) of the subtasks. If two subtasks are independent with each other, the relative order is determined by the TSLs of the subtasks. The larger the TSL, the earlier the subtask appear in the ordering. This is to guarantee that the more time-sensitive nodes are selected to allocate ahead when computing resources are less occupied.

Allocate subtasks to processors by LLP. The procedure of allocating subtasks to computing resources is summarized in Algorithm 1. The subtasks are traversed from the beginning of each ordering (Line 1-3). Only the processors with the same processor tag as the current node will be considered at each iteration.

If the current subtask v_i has no predecessor with the same tag, we traverse all the processors with the same tag as v_i and use the worst-fit (WF) strategy to select the target processor for allocation (Line 4-5). The reason is that if v_i has no predecessor with the same tag, then it is ready to execute once all its predecessors (with different tags if any) finish execution and we want to find the least occupied processor for it to start as soon as possible.

If v_i has exactly one predecessor $pred_i$ with the same tag, and $pred_i$ has no other successors with the same tag (i.e., *siblings* of v_i), we allocate v_i to the same processor p_i of $pred_i$ (Line 6-9). Since v_i can only start execution after $pred_i$ due to the precedence constraint, allocating v_i to a different processor p_j will force it to remain idle until $pred_i$ finishes execution, while during this interval, other subtasks without dependencies between v_i could execute on p_j to maximize parallelism.

If v_i does have one predecessor with the same tag and one of its siblings has been allocated to the same processor of the predecessor, then we traverse all processors with the same tag as v_i and find the target processor by WF to allow v_i to start execution as soon as possible (Line 10).

If v_i has multiple predecessors, due to the dependencies, v_i can only start execution once all of its predecessors finish execution. Among all the processors on which v_i's predecessors have been allocated, find the most occupied one by the Best-fit (BF) strategy (Line 11-14) among all processors with the same tag.

We provide a sketch of proof by cases to demonstrate that LLP will always find an allocation with smaller or equal end-to-end latency than HPC-s and HPC-p methods in [6].

Proof. If for a specific tag t, the platform only has one corresponding processor with the same tag, then both LLP and methods of HPC-DAG will try to allocate all nodes on the single processor. The results of allocation will be the same. Else if there are multiple processors with tag t, allocation based on HPC-s and HPC-p will try to fill processors sequentially and may leave remaining processors idle during the execution, while LLP tries to utilize the computing power of all available processors to reduce latency. Assuming the same scheduling algorithm for each single processor for both methods, in the cases where allocation by HPC-p occupies all processors as well, LLP additonally utilizes the precedence constraints and parallelism among subtasks to maximize parallel execution, thus generating an allocation with less end-to-end latency for safety-critical paths. □

5. The LLP Scheduler: Scheduling

5.1. Scheduling Algorithm

For a single-rate (explained in Section 2.2) and non-preemptive SHPC-DAG, we propose a scheduling algorithm coupled with the allocation algorithm. In the work about HPC-DAG [6], the authors proposed that Earliest

Algorithm 1 LLP Allocation Algorithm

Input:

T: Set of processor tags.
L: Set of sorted orderings for all types of processor tags.
n: Cardinality of the set $n = |T| = |L|$.

```
1: Initilaize index k ← 1.
2: while k ≤ n do t ← T_k, l ← L_k
3:     for (v_i : l) do
4:         if v_i has no predecessor in l then
5:             Allocate v_i to the processor with tag t by WF.
6:         else if v_i has one predecessor in l then
7:             if None of v_i's siblings has been allocated then
8:                 Allocate v_i to the same processor
9:                 of its predecessor.
10:            else Allocate v_i to the processor with tag t by WF.
11:        else if v_i has multiple predecessors in l then
12:            Among all the processors onto which v_i's
13:            predecessors have been allocated, allocate v_i to
14:            the processor with tag t by BF.
15:        else
16:                The allocation of v_i fails. Terminate and return failure.
```

Deadline First (EDF) and Deadline Monotonic (DM) could be a representative scheduler for each processor, given that the intermediate deadlines of subtasks have been assigned. Their methods respect the precedence constraints in DAG automatically but lack the considerations of TSL and stochasticity in our problem.

We illustrate the idea of LLP scheduling algorithm. A feasible schedule on each processor should satisfy two requirements: (1) the precedence constraints among subtasks should not be violated and (2) subtasks with higher TSLs should be guaranteed to execute ahead. As stated in Section 4, the SHPC-DAG is partitioned by processor tags. For each processor tag, the topological sorting is applied to obtain the linear ordering Θ of all subtasks per tag. For each processor, the set of subtasks allocated upon it is denoted as V. For these subtasks, the initial schedule S_i is obtained by sorting subtasks according to Θ with the same tag (Line 3-7). The schedule is then adjusted by re-sorting subtasks with higher TSLs to the front of the schedule without violating the precedence constraints (Line 8-13). Pseudo code of the algorithm is given in Algorithm 2.

5.2. Schedulability analysis

In this section, we assume that subtasks have already been assigned TSLs and safety-critical labels, and they have been allocated on the corresponding types of processors. We present the schedulability analysis to test if all tasks respect their deadlines when scheduled by the LLP algorithm and SP metric.

Definition 5.1. Let $\mathcal{G}$ be a *SHPC-DAG* that is allocated to the hardware platform and the obtained schedule is S under LLP algorithm. Let $\mathcal{V}$ be the set of all safety-critical nodes and $\mathcal{L}$ be the set of all safety-critical paths. Denote τ_{v_i} and τ_{l_i} as the deadlines of the safety-critical nodes and paths respectively. The λ_{v_i} and λ_{l_i} are the probabilistic timing constraints used in SP metric.Task $\mathcal{G}$ is schedulable by LLP if and only if,

$$\begin{cases} P(R_{v_i} > \tau_{v_i}) - \lambda_{v_i} < 0, \forall v_i \in \mathcal{V} \\ P(R_{l_i} > \tau_{l_i}) - \lambda_{l_i} < 0, \forall l_i \in \mathcal{C} \\ (f^r_{node}(S) + f^r_{path}(S)) - (f^p_{cn}(S) + f^p_{cp}(S)) > 0 \end{cases}$$

Based on the definitions in Section 3.2, the SP metric is defined as the sum of all penalties and rewards. If the timing constraints of safety-critical nodes/paths are violated, the whole taskset is deemed as unschedulable.

Algorithm 2 LLP Scheduling Algorithm

Input:

Θ: Linear ordering of subtasks with the same tag as current processor.
V: Set of subtasks allocated to the current processor.
N: Cardinality of the ordering Θ, $N = |\Theta|$.
n: Cardinality of the set V, $n = |V|$.

Output:

S: Final schedule of the current processor determined by LLP.

```
1: Initialize indices i, j, k ← 1.
2: Initial schedule S_i = ∅, final schedule S = ∅.
3: for i = 1:N do
4:     Search each subtask v_i in Θ from the beginning.
5:     if v_i ∈ V then
6:         S_i.push_back(v_i).
7:         remove v_i from Θ.
8: Adjust initial schedule S_i by LLP.
9: for j = 1:n do
10:    for k = 1:j do
11:        if S_i[k].TSL ≤ S_i[k+1].TSL
12:            &&(!is_predecessor(S_i[k],S_i[k+1])) then
13:            swap(S_i[k], S_i[k + 1]).
14: S = S_i.
15: return S.
```

The constraints of these safety-critical nodes/paths can be independent from those of the whole taskset and must be met to ensure safety. Based on this, if the sum of rewards (i.e., $f^r_{node}(S)$ for nodes and $f^r_{path}(S)$ for paths) is greater than penalties (i.e., $f^p_{cn}(S) + f^p_{cp}(S)$), then task $\mathcal{G}$ is deemed as schedulable under LLP algorithm and SP metric.

5.3. Time complexity analysis

We give the time complexity analysis of the proposed partitioned scheduling algorithm which includes both the allocation and scheduling parts. Denote N as the number of subtasks in SHPC-DAG, and n_t as the number of subtasks for one specific tag (e.g., CPU). Similarly, denote K as the number of all processors and k_t as the number of processor with a specific tag. In general, we have $N \geq n_t$ and $K \geq k_t$ since a SHPC-DAG task might contain computing subtasks that should be executed upon different types of processors.

For the LLP allocation algorithm, the first step is to update the TSLs of each subtask and the subtasks will inherit the TSLs of their successors, if any, in the SHPC-DAG. Since the safety-critical paths and nodes are predefined by users, for each subtask v_i, we need to check the dependency between every subtask and v_i. If the precedence constrains exist, the TSL will be updated accordingly. For each tag this process will have complexity $\mathcal{O}(N * (N-1)/2) = \mathcal{O}(N^2)$. The time complexity of topological sort is $\mathcal{O}(N + E)$, where N is the number of subtasks in graph and E is the number of edges connecting subtasks. It is upper bounded by $\mathcal{O}(N^2)$. During step 4 the worst case will require the subtask to traverse all the processors with the same tag to find the most suitable one, which requires $k_t * \mathcal{O}(n)$ at most. The overall time complexity of allocation is $\mathcal{O}(N^2) + \mathcal{O}(N^2) + k_t * \mathcal{O}(n_t) = \mathcal{O}(N^2)$. For the scheduling part on each processor, the overall time complexity will be upper bounded by $K * \mathcal{O}((n_t)^2)$. In a real application, the number of processor on platform K is relatively small compared to the number of subtasks N in application. Therefore the overall time complexity of the proposed partitioned scheduling algorithm is $\mathcal{O}(N^2)$.

6. Evaluation

In this section, we evaluate the performance of our proposed allocation and scheduling algorithm against the state-of-the-art. The test platform simulates NVIDIA AGX, a widely used hardware in robotic applications.

Table 1: Exemplary computational kernels for realistic robotic systems (time in seconds)

Name	Deadline (s)	Tag	WCET	Avg.ET
LOAM SLAM	15	CPU	26.5	12.7
RRT Path Planning	50	CPU	20	15
Task allocation	50	CPU	45	30
MPC HL Control	0.01	CPU	0.16	0.01
Midas Depth Estimation	0.5	GPU	0.35	0.32
PWC-net Optical Flow	0.5	GPU	0.57	0.5

It contains 8 CPU processors and 2 GPU processors. Each processor is considered as a single computing resource [6], i.e., only one subtask at a time is executed on each processor.

We compared the proposed LLP algorithm with recent works that are closely related to multiprocessor DAG scheduling.

- **HPC-s** and **HPC-p**. These are two partitioned scheduling methods originally developed for HPC-DAG models on heterogeneous platforms [6, 37].
- **AG-partition**. A partitioning heuristic for DAG scheduling on multicore platforms with identical cores [38]. We partition the DAG task graph into subgraphs by the processor tags of subtasks first, then apply AG-partition to each subgraph.
- **UGC**. A hierarchical scheduling algorithm based on reservation systems for DAG scheduling on homogeneous multi-processor platforms in study [10]. For fair comparison, the number of reservation servers is set to the number of processors per processor tag. The execution time budget is set to the deadline of DAG for the spinning reservation system.
- A **simulated annealing (SA)** method.

6.1. Taskset Generation

To validate our task model and scheduler, we conduct experiments on both synthetic tasksets and a real-world robotic application.

6.1.1. Synthetic Tasksets Generation

Since a robotic system typically contains a base set of computational subtasks that are widespread in robotic applications, we consider a list of

such subtasks in the design of experiments. A comprehensive data analysis of execution times for such subtasks has been carried out in the study [39]. We take the data from that study to define the execution time probability distributions needed for generating synthetic tasksets, lending practical relevance to our experimental design. Table 1 summarizes the execution time statistics of those representative subtasks. It also gives the other temporal characteristics including subtask deadline and tag.

We select the autonomous robot navigation, one of the most important problems in robotics, as the test scenario. We define two *contexts* that are used to construct the stochastic conditional node. As the robot could operate in either a crowded environment with the interference from obstacles and pedestrian, or a spatial space with only few obstacles, the contexts can be defined as *crowded* or *spatial*. The generation process of taskset is based on well-known methods in the real-time systems literature, as detailed below.

- **Type.** The type of each subtask is initialized as *Computing*. The type can be changed to *Conditional* in following step if the subtask is selected as stochastic conditional node.

- **Tag.** The processor tag assigned to each subtask could either be CPU or GPU, each with an equal probability of 0.5.

- **Probabilities.** The probability distributions that characterize the execution times of nodes are generated as discrete probability mass functions (PMFs). The robotic system is assumed to operate in two contexts with equal probabilities. Therefore, $P(ctx_1) = P(ctx_2) = 0.5$. The four conditional probabilities (refer to Fig. 2), $P(b_1|ctx_1)$, $P(b_1|ctx_2)$, $P(b_2|ctx_1)$, $P(b_2|ctx_2)$ associated with outgoing edges of conditional node are randomly generated such that they satisfy equation (3), i.e., $P(b_i) = \sum_{j=1}^{2} P(b_i|ctx_j) \cdot P(ctx_j)$ and $\sum_{i=1}^{2} P(b_i) = 1$.

- **Execution times.** Based on two defined contexts *crowded* (ctx_1) and *spatial* (ctx_2), the two possible execution times (ET_i) of each subtask are set to its WCET for ctx_1 and $0.5 \cdot$ WCET for ctx_2.

- **Task Utilization.** The utilization ($U_i = C_i/T_i$) of each subtask is defined as the ratio of expected execution time over period, where the expected execution time is calculated as $\sum_{i=1}^{2} P(ctx_i) \cdot ET_i$. For each

subtask, it is randomly generated as one of the six computational kernels provided in Table 1, with corresponding parameters. For each input utilization, the tasksets are generated using the *UUnifast-Discard* algorithm [40].

- **Dependencies.** We follow the method in He et al.'s experiments [41] to generate edges among nodes. A random number is uniformly generated in interval [0,1] and compared with threshold (i.e., parallelism factor in [41]) to decide whether an edge should be added between two nodes. For experiments, the default setting of the threshold is 0.4.

- **Safety-critical nodes/paths assignments**: Depending on real application requirements, the LOAM SLAM kernel and RRT path planning kernel are crucial for safety during navigation [28]. Therefore, these two kernels are assigned highest TSLs and are selected as safety-critical nodes. For synthetic tasksets, each node is generated either as safety-critical node (s = True and TSL = 2) or non-safety-critical node (s = False and TSL = 1). Any paths that contain safety-critical nodes are labeled as safety-critical paths. The threshold τ_{l_i} of a safety-critical path takes value from a range in $(200, 300)$.

- **Stochastic Conditional node assignment** :As the robot could operate in two distinct environments (crowded or spatial space), we first find the set of subtasks in the DAG with exactly two successors, each corresponds to a distinct execution path. Within this set, random subtasks are selected and assigned as stochastic conditional nodes, and their *types* are switched from *computing* to *conditional*. Based on our testing scenario, probabilities $P(b_1|ctx_1)$, $P(b_1|ctx_2)$, $P(b_2|ctx_1)$, $P(b_2|ctx_2)$ are generated such that constraints in the previous steps are satisified.

For each input utilization, the experiments are conducted 5 times with 1000 DAG tasksets each time. The sum of every per-tag utilization is upper bounded by the number of processors.

6.1.2. Real Taskset

In Figure 1, an exemplary robotic system for autonomous driving has been presented. The parameters of these computational kernels are given in Table 1. The path planning kernels in both branches are labeled as safety-critical nodes with highest TSLs since they are crucial for safety problems in

autonomous driving. The conditional node controls the computational paths used for planning based on the cluttered, uncertain nature of the environment.

6.2. Results Analysis

We first evaluate the effectiveness of our proposed LLP algorithm over synthetic tasksets. The comparison consists of three parts: acceptance ratio, average value of safety-performance metric and run-time. We then evaluate the method on a real robotic subsystem.

6.2.1. Experiments on Synthetic Tasksets

In real-world operations, occasional deadline misses of subtasks are acceptable within a certain range. In addition, the execution times of subtasks are usually unknown beforehand and vary during run-time. Our experiments take this stochastic nature of subtasks into account by relaxing the probabilistic timing constraint λ in the SP metric (See (9) to (12) in Section 3.2).

Remark. *When $\lambda = 0$, the system is equivalent to a hard real-time system.*

As discussed in Section 3.2, when $\lambda = 0$, it means that for a task to be deemed schedulable, both the response times of safety-critical path/node have to be smaller than the thresholds and there is no probability tolerance for the timing constraints.

When λ is non-zero, the timing constraints of the real-time system are relaxed and becomes stochastic. For example, if $\lambda = 0.01$, it means that the task will be deemed schedulable if it has a $1 - 0.01 = 99\%$ cumulative probability that the response times of its safety-critical path/node are smaller than the predefined thresholds based on their probability distributions.

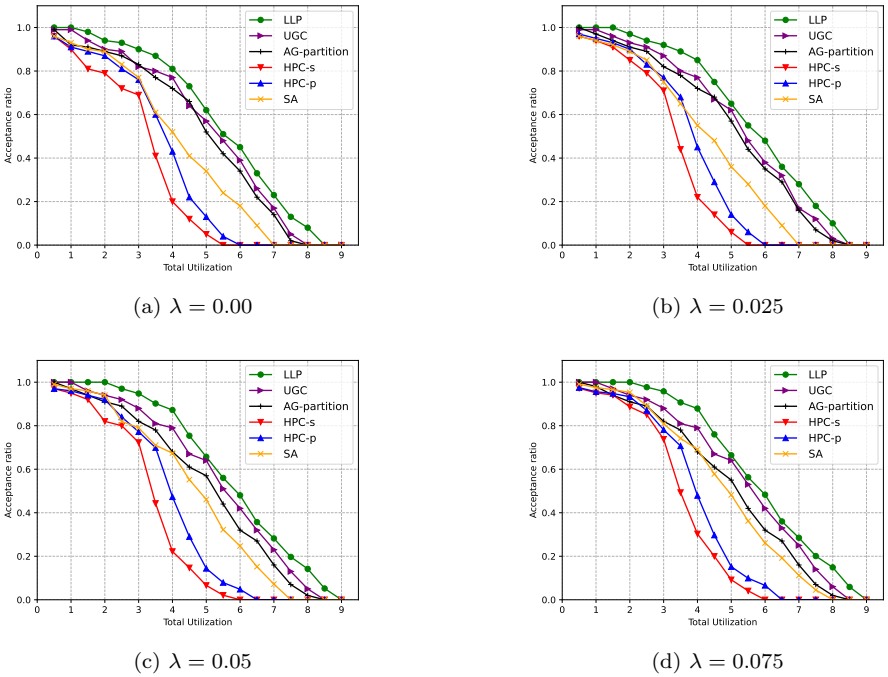
Remark. *The acceptable range of the probability tolerance coefficient λ is set to [0, 0.1].*

Relaxing the probability tolerance λ in a small range improves the schedulability. Probability tolerances are selected in a small range to simulate *weakly-hard* systems where only small deviations from timing requirements are permissible.

Comparison of Acceptance Ratio. In the first subplot of Figure 3, when the probability tolerance λ is set to 0, the timing constraints represent a *hard* real-time system. In addition, the SP metric is equivalent to a conventional metric *makespan* in this case, since there is no tolerance for deadline miss and the response times have to be smaller than deadlines for schedulable tasks. The LLP method outperforms other methods because it aims to maximize SP metric, which is equivalent to makespan for the *hard* real-time systems in this case, in the allocation and scheduling process. In addition, it utilizes TSLs and precedence constraints to further optimize the schedules.

When timing constraints are relaxed by probability tolerance λ, the system becomes *weakly-hard* and LLP still keeps a much higher schedulability than other methods when the utilization of subtasks increases. When the probability tolerance λ increases, the schedulability improves as more tasks will be deemed schedulable if the probability of occasional deadline misses remain within a small range characterized by λ. For the combination of HPC-s allocation and EDF scheduling, when the utilization of task increases, allocating all subtasks on a single processor per tag will more likely results in an infeasible schedule. Therefore, the acceptance ratio of HPC-s significantly drops as utilization increases. HPC-p is based on HPC-s by removing and reallocating one subtask at a time if a processor is fully occupied. It ignores the relative ordering of subtasks' TSLs, which could cause the safety-critical nodes/paths to miss their own independent deadlines. When the maximum iteration of SA is limited to 2000 (due to timing constraints in real operation), it cannot approach the global optimum fast enough which causes the performance gap between LLP and SA. AG-partition method traverses the available cores and allocate the nodes to the core with minimum cost. Although it also considers precedence constraints to reduce latency, it does not guarantee the safety-critical nodes to be scheduled first when precedence constraints are not violated. UGC needs to verify the feasibility of schedule on parallel servers beforehand, if the service cannot be guaranteed, the process is terminated and returns failure. It assumes all servers are identical and allow nodes to execute on any of the parallel reservation servers that are idle (i.e., property 2 in study [10]). However, the precedence constraints among nodes are not utilized when assigning nodes to available servers, which results in a difference between UGC and LLP with respect to acceptance ratio.

Comparison of Safety-Performance Metric. In Figure 4, we evaluate the average safety-performance (SP) metric obtained by different scheduling algo-



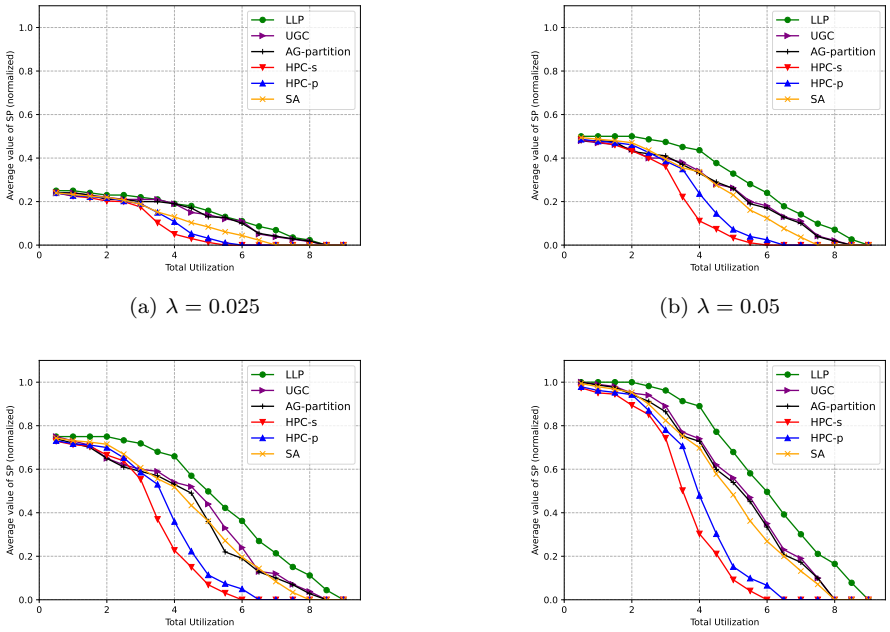
(a) $\lambda = 0.00$ (b) $\lambda = 0.025$

(c) $\lambda = 0.05$ (d) $\lambda = 0.075$

Figure 3: Acceptance ratio comparison with varied probability tolerance λ.

rithm. SP metric can be interpreted as rewarding for each safety-critical path and node, a balance of exceeding timing constraints (safety margin) and system performance related to improved response time. The larger the SP value, the more that schedule could trade exceeding safety margin for better performance or focus on improving safety by sacrificing performance without violating either safety or performance requirement.

In Figure 4, the more relaxed the timing constraints, the larger the difference between LLP and other methods. SA outperforms HPC-s and HPC-p since it uses SP as part of the objective function to optimize. LLP outpeforms AG-partition because (1) it tries to minimize the end-to-end latency in allocation process and prioritize the subtasks with higher TSLs in both allocation and scheduling, which results in more schedulable tasks; (2) LLP runs faster than AG-partition (shown in Table 2), it is less unlikely for a task to be deemed as unschedulable due to timeout in LLP. The reasons are the same for the SA method. LLP outperforms UGC because the execution times are represented as distributions instead of worst-case execution times



(a) $\lambda = 0.025$ (b) $\lambda = 0.05$

(c) $\lambda = 0.075$ (d) $\lambda = 0.1$

Figure 4: Average SP value comparison with varied probability tolerance λ.

in our experiments and UGC are not originally designed for this case. In addition, TSLs of nodes are not specified and utilized in UGC method.

For better clarity, we omit the hard timing constraint case where $\lambda = 0$ since the calculated SP will be zero. We pick $\alpha_{l_i} = \alpha_{n_i} = 0.5$ and show the results in Fig 4. The balancing parameters α_{l_i}, α_{n_i} could be tuned within a certain range (i.e., without violating the minimal requirement for safety-related timing constraints) based on the application. The trends are similar for various balancing parameters, we only provide one as representative due to the space constraint.

All the SP values in figures are normalized with 1000 for better readability.

Comparison of Time Complexity. We compare the efficiency of different methods by setting fixed probability tolerance λ and safety-performance balancing parameters α_{l_i}, α_{n_i}. Since probability tolerance λ will affect the acceptance ratio and balancing parameters will affect the SP value only, we pick $\lambda = 0.01$ and $\alpha_{l_i} = \alpha_{n_i} = 0.9$ to present a representative result in Table 2.

Table 2: Average Execution Time for Different Methods

(time in seconds)

Name/Number of nodes	5	10	15	20	25	30	35	40	45	50
LLP	2.24E-3	7.45E-3	1.41E-2	2.16E-2	3.07E-2	3.99E-2	5.67E-2	7.54E-2	1.14E-1	1.69E-1
UGC	2.07E-3	7.82E-3	1.25E-2	1.69E-2	2.22E-2	2.79E-2	8.14E-3	1.27E-2	1.45E-2	1.67E-2
AG-partition	3.85E-3	7.66E-3	1.59E-2	2.61E-2	3.92E-2	5.74E-2	6.13E-2	9.72E-2	1.52E-1	4.23E-1
HPC-s + EDF	4.47E-3	1.11E-2	1.84E-2	1.33E-2	2.84E-3	5.02E-3	7.12E-3	1.13E-2	1.89E-2	3.27E-2
HPC-p + EDF	4.47E-3	1.11E-2	1.84E-2	2.59E-2	3.36E-2	4.43E-2	5.43E-2	7.04E-2	9.38E-2	1.05E-1
SA	1.57E-1	4.12E-1	6.70E-1	8.20E-1	1.25	1.62	2.72	3.78	5.59	11.9

LLP runs around two orders of magnitude faster than SA in Table 2 because SA is a meta-heuristic that approximates global optimum with a large number of iterations. Compared with the heuristics for HPC-DAG [6], LLP runs with the same level of speed but provides a much higher acceptance ratio. LLP runs around 30% faster than AG-partition on average. The reason is that for each subtask, AG-partition will first compute its cost when assigned to a processor. Since this partitioning part requires the calculation of cost function for all processors, it will be cost-expensive when the number of subtasks increases. LLP runs as fast as UGC when the size of DAG is relatively small (i.e., number of nodes is smaller than 25). When the size of DAG increases, more DAG tasks are deemed as unschedulable by UGC because the execution time budget of the reservation system is static [10], and the promised service must be verified. If a feasible schedule that guarantees the service cannot be found beforehand, the process is terminated and returns failure directly, which saves scheduling time. To conclude, LLP performs better than heuristic methods (e.g., HPC-s/p, SA etc.) for real-time systems that require efficient scheduling. At the cost of slower speed, LLP achieves better performance than UGC according to SP metric.

6.2.2. Experiments on Real Tasksets

We utilize the exemplary robotic system in Figure 1 as the real taskset for the experiments. The probability mass function (PMF) of execution time for each computational kernels is constructed based on the parameters in Table 1, where the PMF consists of the worst-case execution time (WCET), minimum execution time and average execution time with corresponding probabilities reported in the study [39]. Two path planning nodes are labeled as safety-critical nodes based on the application requirement.

Similar to the experiments of synthetic tasksets, we apply different scheduling algorithm to this specific DAG taskset and run the experiments for 1000 times with various deadlines and probability tolerance λ. The deadline of the

application takes value in the interval [75,120], which corresponds to various operating context and timing requirements.

Table 3: Average run-time of varied scheduling algorithms (time in seconds)

LLP	UGC	AG-partition	HPC-s	HPC-p	SA
5.01E-4	4.86E-4	6.12E-4	7.47E-4	7.42E-4	2.29E-3

Table 4: Average SP metric with varied probability tolerance λ

	LLP	UGC	AG-partition	HPC-s	HPC-p	SA
$\lambda = 0.025$	0.682	0.479	0.473	0.079	0.099	0.475
$\lambda = 0.050$	0.696	0.481	0.472	0.099	0.099	0.468
$\lambda = 0.075$	0.712	0.492	0.474	0.100	0.109	0.495

As can be seen from Table 3, LLP runs as fast as UGC method with a difference of 3%, which aligns with results in Table 2 because the size of real taskset is relatively small. LLP runs faster than other methods with a minimum of 20% run-time improvement. For varied probability tolerance λ in Table 4, the SP metric schedule under LLP outperforms other methods with a significant margin. Although LLP runs slightly slower than UGC method in Table 3, it achieves much higher SP value with an average improvement of 42%.

7. Related Work

Existing work on the DAG task model mostly focuses on introducing extensions to the DAG task to describe the complexity of real-time applications [4]. Conditional DAG (C-DAG) introduces a conditional node that represents if-then-else structures for conditional execution [35, 42, 43, 44, 45]. Heterogeneous DAG, is another extension of the DAG task model. It associates each subtask with a tag to specify the kind of processing unit that each subtask should run upon [5, 6, 46, 47, 48, 49]. However, the stochastic nature of unknown execution times during run-time is absent in these studies, and they all use the WCETs of computational subtasks for response-time analysis, which may be overly pessimistic [8].

While the stochastic DAG scheduling problem has been studied in the literature [9, 11, 12, 50], they all assumed that random execution times followed specific probability distributions (e.g., the Gaussian distribution). Ueter et. al. considered a probabilistic version of the C-DAG in [10]. In contrast with

our present work, the probabilistic aspects of [10] only focus on the outgoing edges of the conditional node. They still consider the WCETs of nodes in the scheduling process with the assumption of an identical multi-core system, which is different from our S-RASP problem. In our work, we do not make assumptions of specific probability distributions for random execution times of subtasks. In addition, we consider the probabilistic nature both in the computing and conditional nodes, and propose a general solution to the scheduling problem on heterogeneous platforms.

Searching for an optimal schedule for tasks modeled by DAGs is considered to be NP-hard, therefore most existing scheduling algorithms are heuristics [3]. Recently, Baruah [5] formulated the scheduling problem as an integer linear programming (ILP) problem, which is an exact test with time complexity $\mathcal{O}(n^3)$ that may serve as a baseline against which the performance of heuristics can be compared. However, the stochastic nature of execution times of computational kernels is absent in this recent work as well.

8. Conclusion and Future Work

This paper proposed a partitioned scheduling algorithm called LLP which consists of novel allocation and scheduling algorithm for stochastic heterogeneous conditional DAG scheduling, which takes the stochastic nature of execution time and conditional nodes within the DAG tasks into account. We use a multi-objective metric called the Safety-Performance (SP) metric to explore the trade-offs between timing constraints and system performance for safety-critical systems. Furthermore, unlike existing works for stochastic scheduling, we do not make any assumptions about the specific distributions of random execution times, instead we propose a convolution based method to evaluate the response times of safety-critical paths and nodes against the stochastic timing constraints. The proposed algorithm LLP outperforms the state-of-the-art DAG scheduling algorithms for multi-processor systems. In future work, we plan to extend our work with multiple DAGs and SP metric with a third objective function of software reliability.

References

[1] L. Heintzman, A. Hashimoto, N. Abaid, R. K. Williams, Anticipatory planning and dynamic lost person models for Human-Robot search and rescue, in: 2021 IEEE International Conference on Robotics and Automation (ICRA), 2021, pp. 8252–8258.

[2] M. Rangwala, J. Liu, K. S. Ahluwalia, S. Ghajar, H. S. Dhami, others, DeepPaSTL: Spatio-Temporal deep learning methods for predicting Long-Term pasture terrains using synthetic datasets, Agronomy (2021).

[3] J. Huang, R. Li, X. Jiao, Y. Jiang, W. Chang, Dynamic dag scheduling on multiprocessor systems: Reliability, energy, and makespan, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 39 (2020) 3336–3347. doi:10.1109/TCAD.2020.3013045.

[4] M. VERUCCHI, M. BERTOGNA, A comprehensive analysis of dag tasks: solutions for modern real-time embedded systems, PhD Thesis (2021).

[5] S. Baruah, An ilp representation of a dag scheduling problem, Real-Time Systems (2021) 1–18.

[6] Z. Houssam-Eddine, N. Capodieci, R. Cavicchioli, G. Lipari, M. Bertogna, The hpc-dag task model for heterogeneous real-time systems, IEEE Transactions on Computers 70 (2021) 1747–1761. doi:10.1109/TC.2020.3023169.

[7] H.-E. Zahaf, N. Capodieci, R. Cavicchioli, M. Bertogna, G. Lipari, A c-dag task model for scheduling complex real-time tasks on heterogeneous platforms: preemption matters, arXiv preprint arXiv:1901.02450 (2019).

[8] S. Edgar, A. Burns, Statistical analysis of wcet for scheduling, in: Proceedings 22nd IEEE Real-Time Systems Symposium (RTSS 2001) (Cat. No.01PR1420), 2001, pp. 215–224. doi:10.1109/REAL.2001.990614.

[9] K. Li, X. Tang, B. Veeravalli, K. Li, Scheduling precedence constrained stochastic tasks on heterogeneous cluster systems, IEEE Transactions on computers 64 (2013) 191–204.

[10] N. Ueter, M. Günzel, J.-J. Chen, Response-time analysis and optimization for probabilistic conditional parallel dag tasks, in: 2021 IEEE Real-Time Systems Symposium (RTSS), 2021, pp. 380–392. doi:10.1109/RTSS52674.2021.00042.

[11] L.-C. Canon, E. Jeannot, Evaluation and optimization of the robustness of dag schedules in heterogeneous environments, IEEE

Transactions on Parallel and Distributed Systems 21 (2010) 532–546. doi:10.1109/TPDS.2009.84.

[12] K. Cao, J. Zhou, P. Cong, L. Li, T. Wei, M. Chen, S. Hu, X. S. Hu, Affinity-driven modeling and scheduling for makespan optimization in heterogeneous multiprocessor systems, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 38 (2019) 1189–1202. doi:10.1109/TCAD.2018.2846650.

[13] V. Feldman, C. Zhang, What neural networks memorize and why: Discovering the long tail via influence estimation, arXiv preprint arXiv:2008.03703 (2020).

[14] D. Casini, A. Biondi, G. Nelissen, G. Buttazzo, Partitioned fixed-priority scheduling of parallel tasks without preemptions, in: 2018 IEEE Real-Time Systems Symposium (RTSS), IEEE, 2018, pp. 421–433.

[15] A. H. Sifat, X. Deng, B. Bharmal, S. Wang, S. Huang, J. Huang, C. Jung, H. Zeng, R. Williams, A safety-performance metric enabling computational awareness in autonomous robots, IEEE Robotics and Automation Letters 8 (2023) 5727–5734. doi:10.1109/LRA.2023.3300251.

[16] P. Pedro, A. Burns, Schedulability analysis for mode changes in flexible real-time systems, in: EUROMICRO Workshop on Real-Time Systems, 1998.

[17] A. Shafti, P. Orlov, A. A. Faisal, Gaze-based, context-aware robotic system for assisted reaching and grasping, in: 2019 International Conference on Robotics and Automation (ICRA), IEEE, 2019, pp. 863–869.

[18] M. Shen, Y. Gu, N. Liu, G.-Z. Yang, Context-aware depth and pose estimation for bronchoscopic navigation, IEEE Robotics and Automation Letters 4 (2019) 732–739.

[19] K. Li, Y. Xu, J. Wang, M. Q.-H. Meng, SARL*: Deep reinforcement learning based Human-Aware navigation for mobile robot in indoor environments, in: 2019 IEEE International Conference on Robotics and Biomimetics (ROBIO), 2019, pp. 688–694.

[20] P. Lin, The robot car of tomorrow may just be programmed to hit you, Machine Ethics and Robot Ethics (2020).

[21] L. Chen, S. Lin, X. Lu, D. Cao, H. Wu, C. Guo, C. Liu, F.-Y. Wang, Deep neural network based vehicle and pedestrian detection for autonomous driving: A survey, IEEE Trans. Intell. Transp. Syst. 22 (2021) 3234–3246.

[22] P. Graydon, I. Bate, Safety assurance driven problem formulation for mixed-criticality scheduling, Proc. WMC, RTSS (2013) 19–24.

[23] R. Bell, Introduction to iec 61508, in: Acm international conference proceeding series, volume 162, Citeseer, 2006, pp. 3–12.

[24] L. A. Johnson, et al., Do-178b, software considerations in airborne systems and equipment certification, Crosstalk, October 199 (1998) 11–20.

[25] H. Zhao, Y. Zhang, P. Meng, H. Shi, L. E. Li, T. Lou, J. Zhao, Safety score: A quantitative approach to guiding safety-aware autonomous vehicle computing system design, in: 2020 IEEE Intelligent Vehicles Symposium (IV), 2020, pp. 1479–1485. doi:10.1109/IV47402.2020.9304602.

[26] A. Vincentelli, P. Giusto, C. Pinello, W. Zheng, M. Natale, Optimizing end-to-end latencies by adaptation of the activation events in distributed automotive systems, in: 13th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS'07), IEEE, 2007, pp. 293–302.

[27] M. Verucchi, M. Theile, M. Caccamo, M. Bertogna, Latency-aware generation of single-rate dags from multi-rate task sets, in: 2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), IEEE, 2020, pp. 226–238.

[28] L. Schmid, V. Reijgwart, L. Ott, J. Nieto, R. Siegwart, C. Cadena, A unified approach for autonomous volumetric exploration of large scale environments under severe odometry drift, IEEE Robotics and Automation Letters 6 (2021) 4504–4511.

[29] D. Ratasich, B. Frömel, O. Höftberger, R. Grosu, Generic sensor fusion package for ros, in: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2015, pp. 286–291.

[30] S. Shalev-shwartz, S. Shammah, A. Shashua, On a Formal Model of Safe and Scalable Self-driving Cars (2017) 1–37. arXiv:arXiv:1708.06374v6.

[31] S. B. Amor, Scheduling of Dependent Tasks with Probabilistic Execution Times on Multi-core Processors, Ph.D. thesis, Sorbonne Université, 2020.

[32] D. Maxim, L. Cucu-Grosjean, Response time analysis for fixed-priority tasks with multiple probabilistic parameters, in: 2013 IEEE 34th Real-Time Systems Symposium, IEEE, 2013, pp. 224–235.

[33] A. M. Mood, Introduction to the theory of statistics. (1950).

[34] D. L. Applegate, R. E. Bixby, V. Chvatál, W. J. Cook, A Computational Study, Princeton University Press, 2006.

[35] J. Sun, N. Guan, J. Sun, Y. Chi, Calculating response-time bounds for openmp task systems with conditional branches, in: 2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), IEEE, 2019, pp. 169–181.

[36] W.-H. Huang, M. Yang, J.-J. Chen, Resource-oriented partitioned scheduling in multiprocessor systems: How to partition and how to share?, in: 2016 IEEE Real-Time Systems Symposium (RTSS), IEEE, 2016, pp. 111–122.

[37] S. Baruah, An ilp representation of a dag scheduling problem, Real-Time Systems (2021) 1–18.

[38] S. Ben-Amor, L. Cucu-Grosjean, Graph reductions and partitioning heuristics for multicore dag scheduling, Journal of Systems Architecture 124 (2022) 102359.

[39] A. H. Sifat, B. Bharmal, H. Zeng, J.-B. Huang, C. Jung, R. K. Williams, Towards computational awareness in autonomous robots: an empirical study of computational kernels, Complex & Intelligent Systems (2023) 1–27.

[40] R. I. Davis, A. Burns, Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems, in: IEEE Real-Time Systems Symposium, 2009. doi:10.1109/RTSS.2009.31.

[41] Q. He, M. Lv, N. Guan, Response time bounds for dag tasks with arbitrary intra-task priority assignment, in: 33rd Euromicro Conference on Real-Time Systems (ECRTS 2021), Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[42] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, The global edf scheduling of systems of conditional sporadic dag tasks, in: 2015 27th Euromicro Conference on Real-Time Systems, 2015, pp. 222–231. doi:10.1109/ECRTS.2015.27.

[43] J. C. Fonseca, V. Nélis, G. Raravi, L. M. Pinho, A multi-dag model for real-time parallel applications with conditional execution, in: Proceedings of the 30th Annual ACM Symposium on Applied Computing, 2015, pp. 1925–1932.

[44] A. Melani, M. Bertogna, V. Bonifaci, A. Marchetti-Spaccamela, G. C. Buttazzo, Response-time analysis of conditional dag tasks in multiprocessor systems, in: 2015 27th Euromicro Conference on Real-Time Systems, IEEE, 2015, pp. 211–221.

[45] R. Pathan, P. Voudouris, P. Stenström, Scheduling parallel real-time recurrent tasks on multicore platforms, IEEE Transactions on Parallel and Distributed Systems 29 (2017) 915–928.

[46] S. Chang, X. Zhao, Z. Liu, Q. Deng, Real-time scheduling and analysis of parallel tasks on heterogeneous multi-cores, Journal of Systems Architecture 105 (2020) 101704.

[47] M. Han, N. Guan, J. Sun, Q. He, Q. Deng, W. Liu, Response time bounds for typed dag parallel tasks on heterogeneous multi-cores, IEEE Transactions on Parallel and Distributed Systems 30 (2019) 2567–2581.

[48] M. A. Serrano, E. Quinones, Response-time analysis of dag tasks supporting heterogeneous computing, in: Proceedings of the 55th Annual Design Automation Conference, 2018, pp. 1–6.

[49] K. Yang, M. Yang, J. H. Anderson, Reducing response-time bounds for dag-based task systems on heterogeneous multicore platforms, in: Proceedings of the 24th International Conference on Real-Time Networks and Systems, 2016, pp. 349–358.

[50] B.-A. Slim, C.-G. Liliana, D. Maxim, Worst-case response time analysis for partitioned fixed-priority dag tasks on identical processors, in: 2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), IEEE, 2019, pp. 1423–1426.

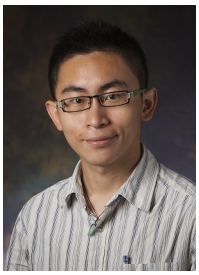
Author biographies

Xuanliang Deng is currently a Ph.D. student in the Electrical and Computer Engineering department at Virginia Tech. He received his master's degree in Electrical and Computer Engineering from Georgia Institute of Technology. His research interests include real-time system scheduling and optimization.

Ash is an Engineer and Researcher with a knack for solving real life problems and utilizing digital technologies for improving human life. He completed a bachelor's in Electrical and Electronic Engineering from Bangladesh University of Engineering and Technology. He then did his Master's and subsequent PhD in Electrical and Computer Engineering from Virginia Tech. Ash is driven for excellence in research and his PhD research was on Computational awareness for timely and resilient autonomous robots.

Shao-Yu Huang received the B.S. degree in computer science from National Yang Ming Chiao Tung University, Taiwan, in 2019. He is currently working toward the Ph.D. degree in computer science with Purdue University, USA. His research interests include compilers, computer architecture, soft error resilience, and real-time systems.

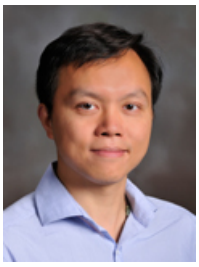
Sen Wang is a Ph.D. student at Virginia Tech. He received his M.S. from Georgia Tech, and B.S. from Northeastern University in China. His research interests lies in the intersection between optimization and real-time systems and robotics.



Jia-Bin Huang is a Capital One endowed Associate Professor in Computer Science at the University of Maryland College Park. He received his Ph.D. degree from the Department of Electrical and Computer Engineering at the University of Illinois, Urbana-Champaign. His research interests include computer vision, computer graphics, and machine learning. Huang is the recipient of the Thomas & Margaret Huang Award, NSF CRII award, faculty award from Samsung, Google, 3M, Qualcomm, and a Google Research Scholar Award.

Changhee Jung (Senior Member, IEEE) received the PhD degree in Computer Science from Georgia Institute of Technology, Atlanta, Georgia, in 2013. He is currently an Associate Professor of Computer Science at Purdue University, West Lafayette. His research interests span the field of compilers and computer architecture with an emphasis on performance, reliability, and security.

Ryan K. Williams (Member, IEEE) received the B.S. degree in computer engineering from Virginia Polytechnic Institute and State University, Blacksburg, VA, USA, in 2005, and the Ph.D. degree in electrical engineering from the University of Southern California, Los Angeles, CA, USA, in 2014. He is currently an Assistant Professor with the Electrical and Computer Engineering Department, Virginia Polytechnic Institute and State University, where he runs the Virginia Tech Laboratory for Coordination at Scale. His research interests include control, cooperation, and intelligence in distributed multiagent systems, topological methods in cooperative phenomena, and distributed algorithms for optimization, estimation, inference, and learning. Dr. Williams was the recipient of the Viterbi Fellowship, the NSF CRII and CAREER grants for young investigators. He was the finalist for the Best Multirobot Paper at the 2017 IEEE International Conference on Robotics and Automation, and has been featured by various news outlets, including the L.A. Times.



Haibo Zeng is with the Department of Electrical and Computer Engineering at Virginia Tech, USA. He received his Ph.D. in Electrical Engineering and Computer Sciences from the University of California at Berkeley. He was a senior researcher at General Motors R&D until October 2011, and an assistant professor at McGill University until August 2014. His research interests are embedded systems, cyber-physical systems, and real-time systems. He received five paper awards in the above fields.