S-Bottleneck Scheduling with Safety-Performance Trade-offs in Stochastic Conditional DAG Models

Ashrarul H. Sifat*, Xuanliang Deng*, Shao-Yu Huang, Burhanuddin Bharmal, Sen Wang,
Ryan K. Williams, Haibo Zeng, Changhee Jung, Jia-bin Huang
Virginia Tech, Purdue University, University of Maryland

***Abstract*—In this paper, we propose a general solution to the problem of scheduling real-time applications on heterogeneous hardware platforms. To fully utilize the computing capacity of heterogeneous processing units, we model the real-time application as a heterogeneous Directed Acyclic Graph (DAG) which specifies the types of processors (CPU, GPU, etc.) where each task should run. In this well-known DAG context, we propose a novel extension aimed at safety-critical systems that operate in unpredictable environments: the coupling of conditional DAG nodes with *stochasticity*. Specifically, conditional DAG nodes enable the modeling of systems that execute computational pipelines based on *environmental context*, while stochasticity of DAG edges captures the uncertain nature of a system's environment or the reliability of its hardware. Furthermore, considering the pessimism of deterministic worst-case execution time (WCET) in scheduling processes, we model execution times of tasks (DAG nodes) as *probability distributions* which yields a novel *stochastic conditional DAG* model. Coupled with a novel S-bottleneck heuristic and safety-performance (SP) metric, our proposed framework allows for efficient online scheduling in complex computational pipelines, with more flexible representation of timing constraints, and ultimately, safety-performance trade offs.**

I. System model

A. Stochastic Heterogeneous Conditional DAGs

In the challenge problem model, there are precedence constraints among different computational tasks, i.e., tasks are connected with input/output ports following a specific execution order. To capture this nature, we propose a new DAG task model, the *Stochastic Heterogeneous Conditional Directed Acyclic Graph (StochHC-DAG)*, which incorporates both the timing and resource constraints for safety-critical autonomous systems. Our core concept is to model computational pipelines that execute conditionally under some uncertainty, recognizing that not all outcomes can be perfectly predicted in real-world applications (e.g., extreme events). In practice, the execution times of tasks are not perfectly known before run-time and may vary *online* due to environmental dynamism and conditions of the hardware platform. Thus, we propose a generalization of the challenge problem model by assuming the execution times of tasks follow probability distributions rather than WCET to reduce schedule pessimism. To fully utilize this probabilistic information and respect timing constraints, we additionally propose new DAG node structures: (1) stochastic conditional nodes for computational path selection; and (2) sensor/synchronization nodes for controlling the difference of timestamps among data streams. A real-time application is then represented by a *StochHC-DAG*, $G = (V, E, C, Type, Tag)$, described by [1] [2]:

- $\mathbf{V} = \{v_1, v_2, \cdots, v_n\}$ is the set of IDs for all computational tasks in the application.
- $\mathbf{E} \subseteq V \times V$ is the set of edges among tasks that indicates the data dependencies, with associated probabilities indicating the likelihood of edge traversal during execution.
- $\mathbf{C} = \{C_1, C_2, \cdots, C_n\}$ is the set of probability distributions of execution times for all tasks.
- **Type** $= \{type_1, type_2, \cdots, type_n\}$ is the set of types of all tasks. A node in *StochHC-DAG* has one of the following types {Computing, Conditional, Sensor, Sync}
- **Tag** $= \{tag_1, tag_2, \cdots, tag_n\}$ indicates the type of processing units that each task should run onto (e.g. CPU, GPU, DLA etc.).

An example of the proposed DAG framework for the challenge problem is given in Figure 4.

B. Stochastic Conditional Nodes in DAG Models

To capture the stochastic nature of real-time applications, a new stochastic conditional node structure is proposed in *StochHC-DAG*. This node allows for the selection of computational paths in a DAG based on events that occur under uncertainty. For example, consider the event that an autonomous vehicle is in a good environment for object detection/tracking vs. a bad environment. In any given window of time, the outcome of this event is uncertain as detecting/predicting environment conditions in practice is imperfect. Thus, if varying computational pipelines are necessary based on environment conditions, it is critical to model the *distribution* of possible execution times. Figure 1 illustrates the structure of our stochastic conditional node which allows for such a modeling. Nodes v_1 and v_2 represent two different computational paths selected according to environment type. The outgoing edges of the conditional node are associated with probabilities since the detection of environment type is uncertain, allowing us to define execution time *distributions* based on conditional uncertainty.

C. Sensor and Synchronization Nodes in DAG Models

As our stochastic conditional node is based on the concept of environmental events, we propose two new DAG nodes for sensing, data synchronization, and event generation: (1) a

This work was supported by the National Science Foundation under grant CNS-1932074.

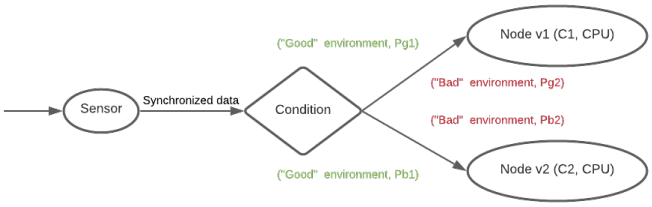

Fig. 1: Structure of stochastic conditional node.

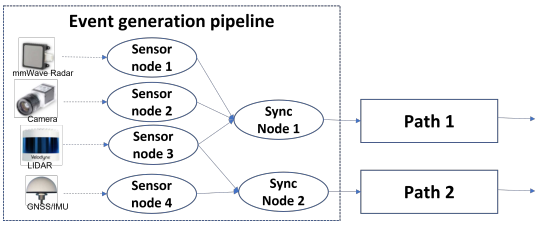

Fig. 2: Event generation pipeline depicting sensor and sync nodes.

sensor node for modeling sensor lag and data post-processing; and (2) a *synchronization node* for fusing sensor data with different frequencies. We then define an *event generation pipeline* comprising these nodes which acts as an event source for our DAG task model (Figure 2). The *sensor node* defines the distribution of lag between the occurrence of a physical event (e.g., nearby obstacle) and the availability of raw sensor data representing the event. Modeling this information in a DAG allows our scheduling algorithm to account for sensor characteristics and satisfy data-based timing constraints. The *synchronization node* then collects data streams from our sensor nodes and outputs a fused data stream with a given user-defined frequency, guaranteeing a bounded difference of data stream timestamps. To implement our synchronization node, we propose a smart ring buffer which utilizes data age and period as illustrated in Figure 3. At each timestep, any new data are written into the smart ring buffer head. The sensor timestamp difference as well as the individual period and age requirements are then verified and data not satisfying these criteria are dropped from the tail. The wide availability of DDS middleware make the information for the synchronicity check readily available for most autonomous systems [3].

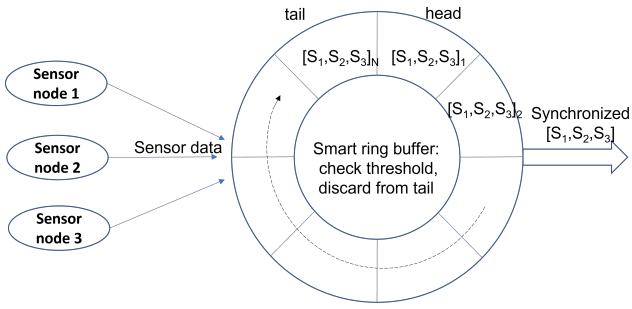

Fig. 3: Smart ring buffer for sensor synchronization in sync node.

II. Shifting Bottleneck Scheduling Algorithm

We propose a new scheduling algorithm which utilizes our proposed DAG model and shifting bottleneck heuristics [4] [5]. Baruah et al. has proposed an exact method to solve the DAG scheduling problem by solving it as an Integer Linear Programming (ILP) problem [6]. However, it only works with the simplest DAG model and he proves in his later work that it is unlikely to write ILP solver for conditional DAG in polynomial time [7]. Therefore, considering the safety-critical requirements and dynamism of autonomous systems, we need an efficient heuristic instead of an exact method.

A detailed explanation and preliminary implementation of our algorithm is provided on https://github.com/Xuanliang-Deng/RTSS2021_Industry_Submission. The process of the algorithm is briefly summarized below.

- **Partition the DAG nodes**: We consider the heterogeneous platform which consists of different types of processing units (e.g., CPU, GPU, DSP etc.). In *StochHC-DAG*, each node is associated with a tag which indicates the processing unit where the node should run. Each node is statically mapped to a processing unit and the mapping is fixed a priori, thereby partitioning the DAG nodes according to their tags and allocating them to the corresponding processing unit.
- **Select Bottleneck Processor**: The starting makespan of the DAG is determined by the maximal finish time (FT) of all nodes on the set of processing units. To select the bottleneck processor, we first assume that there are no resource conflicts and each schedulable task originates at a source node and finishes in a sink node of the DAG. The potential starting time (ST), where a node v_i can start its execution, is the maximal finish time among all its predecessors,

$$ST_i = \max_{k \in pred(i)} FT_k \tag{1}$$

The execution time is denoted as ET_i, which follows a probability distribution, yielding the finish time of node v_i as:

$$FT_i = ST_i + ET_i(C_i) \tag{2}$$

The starting makespan MK_k of processing unit k is,

$$MK_k = \max_{node} \max_{v_i \in proc(k)} FT_i \tag{3}$$

Finally, the starting bottleneck processor is selected by,

$$\max_{k \in 1,2,\ldots,K} MK_k \tag{4}$$

- **Find optimal schedule with Branch and Bound (BnB)**: For selected bottleneck processor, we apply a single-processor analysis by searching with BnB to determine the optimal schedule. This search is different from typical BnB techniques as we utilize the precedence constraints and criticality of nodes in *StochHC-DAG* (see next Section) to greatly reduce the search. Specifically, any potential schedule which violates the precedence constraints

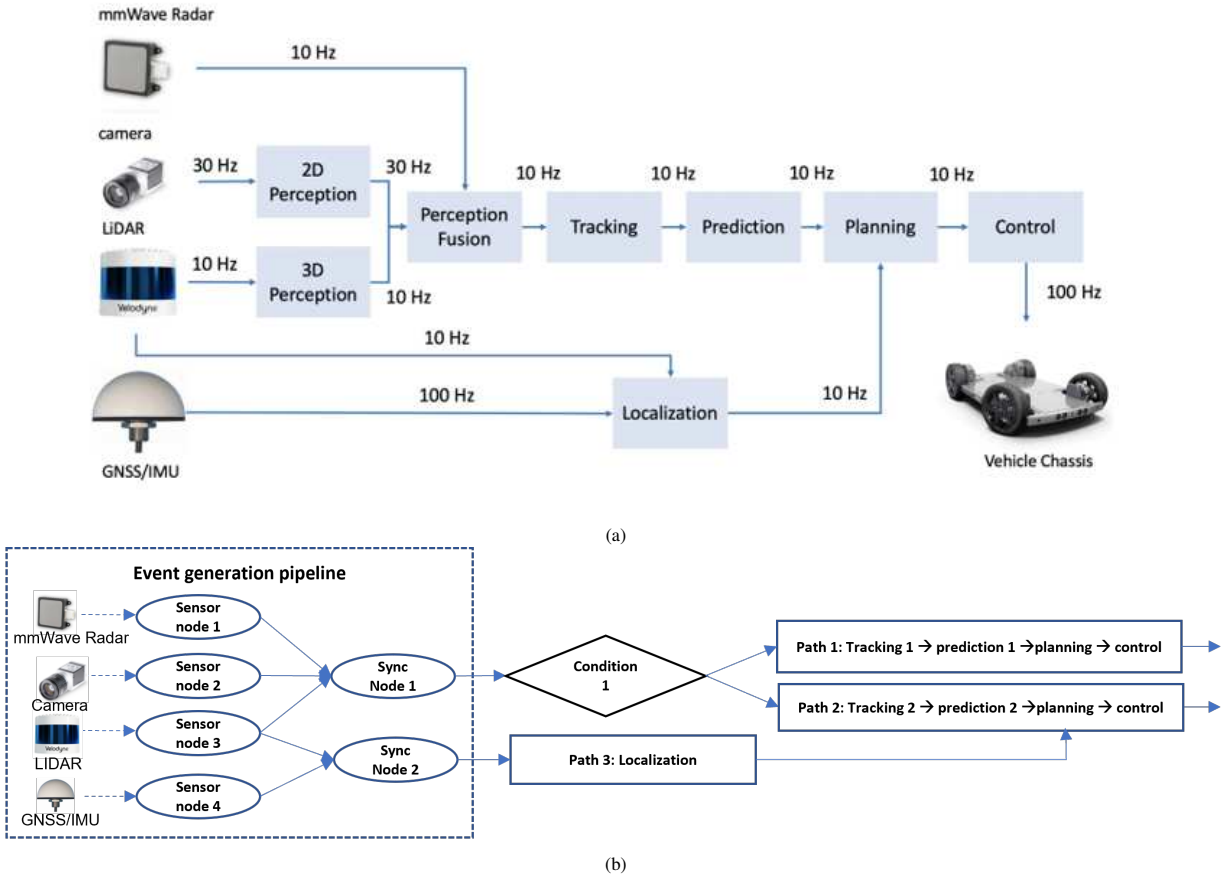

Fig. 4: (a) Challenge model of an autonomous vehicle computational system; (b) Proposed Stochastic HPC-DAG framework for the problem statement.

in *StochHC-DAG* will be infeasible. This branch will be cut directly in the search. In addition, nodes with higher criticality are expected to be scheduled ahead of normal DAG nodes on the same processor. The remaining feasible schedules with greatest objective function value (see next Section) will be selected as the optimal one.

- **Shift Bottleneck Processor** Once the optimal schedule of bottleneck processor is determined, we shift the bottleneck to the next processing unit which has maximal value of MK_k in the remaining processing units. The whole process is terminated when all the processors are traversed.

III. A Safety-Performance Metric

Our proposed scheduling approach requires an objective function to optimize when selecting appropriate schedules. While a typical function can be used, such as makespan, we propose a metric that recognizes that safety can live on a spectrum and, when appropriate, safety can be traded off with system performance. Importantly, we do not suggest that safety requirements are ignored, instead we propose to identify safety-critical nodes and paths in our *StochHC-DAG*, allowing our scheduler to ensure safety where necessary and then exploit remaining timing "headroom" to maximize performance. Specifically, we propose a novel *safety-performance metric* by defining a series of penalties/rewards based on violating/satisfying timing constraints in a *StochHC-DAG*. We start with the concept that our metric should penalize when safety-critical paths and/or critical nodes violate timing constraints based on a particular schedule. This implies that a system designer must label all paths and nodes in our *StochHC-DAG* as either safety-critical or non-safety-critical based on the application (e.g., a computational path for pedestrian detection would certainly have a safety-critical label). With this in mind, we define the first term of our metric which penalizes unsafe critical paths:

$$f_{\text{cp}}^{\text{p}}(\mathcal{S}) = \sum_{\ell_i \in \mathcal{C}_{\text{us}}} p_{\text{cp}}(P(R_{\ell_i} > \tau_{\ell_i}) - \lambda_{\ell_i}) \tag{5}$$

In the above term, we define $\mathcal{C}_{\text{us}}$ as the set of safety-critical DAG paths that violate a *probabilistic* timing constraint, that is, $P(R_{\ell_i} > \tau_{\ell_i}) > \lambda_{\ell_i}$ where R_{ℓ_i} is the random variable describing the uncertain response time of critical path ℓ_i, τ_{ℓ_i} is the *minimally safe* response time for path ℓ_i, and λ_{ℓ_i} is the probabilistic timing constraint for ℓ_i. With these definitions, and noting that $f_{\text{cp}}^{\text{p}}(\mathcal{S})$ 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 (5) can be interpreted as penalizing based on the *deviation* of every violating critical path from its probabilistic timing constraint. Thus, if there are no safety-

critical paths that violate their timing constraints based on schedule $\mathcal{S}$ then $\mathcal{C}_{\text{us}} = \emptyset$ and $f^{\text{p}}_{\text{cp}}(\mathcal{S}) = 0$ yielding no penalty. Otherwise, the severity of constraint violation dictates the penalty, driving our schedule optimization to improve critical path timing. It is important to note for the above term and all terms defined below, that if a hard timing constraint is desired, one can simply set $\lambda_{\ell_i} = 0$ which enforces sureness of satisfying $R_{\ell_i} > \tau_{\ell_i}$.

Next, we define a similar term for penalizing unsafe critical nodes in a DAG:

$$f^{\text{p}}_{\text{cn}}(\mathcal{S}) = \sum_{v_i \in \mathcal{V}_{\text{us}}} p_{\text{cn}}(P(R_{v_i} > \tau_{v_i}) - \lambda_{v_i}) \tag{6}$$

where $\mathcal{V}_{\text{us}}$ is the set of safety-critical DAG nodes that violate a *probabilistic* timing constraint, that is, $P(R_{v_i} > \tau_{v_i}) > \lambda_{v_i}$ where R_{v_i} is the random variable describing the uncertain response time of critical node v_i, τ_{v_i} is the minimally safe response time for node v_i, and λ_{v_i} is the probabilistic timing constraint for node v_i. Importantly, we model specific terms for critical nodes as there may be instances where a timing constraint for a critical path is satisfied but the system remains unsafe. For example, if a localization and mapping node is too slow, even if the computational path it lies on meets a timing constraint, the staleness of the map may endanger the system or bystanders.

With the penalties for our metric defined, we now describe rewards gained when timing constraints for safety-critical paths are satisfied. Critically, the following reward terms are non-zero *only* when there exists no critical path or node constraints that are violated. In this way, a system will focus purely on safety when required, only balancing safety and performance when all critical constraints are satisfied. The reward terms for our metric are now:

$$\begin{aligned} f^{\text{r}}_{\text{path}}(\mathcal{S}) = \sum_{\ell_i \in \mathcal{P}} & \alpha_{\ell_i} r^{\text{s}}_{\text{path}}(\lambda_{\ell_i} - P(R_{\ell_i} > \tau_{\ell_i})) \\ & + (1 - \alpha_{\ell_i}) r^{\text{p}}_{\text{path}}(P(R_{\ell_i})) \end{aligned} \tag{7}$$

and

$$\begin{aligned} f^{\text{r}}_{\text{node}}(\mathcal{S}) = \sum_{v_i \in \mathcal{V}} & \alpha_{v_i} r^{\text{s}}_{\text{node}}(\lambda_{v_i} - P(R_{v_i} > \tau_{v_i})) \\ & + (1 - \alpha_{v_i}) r^{\text{p}}_{\text{node}}(P(R_{v_i})) \end{aligned} \tag{8}$$

In the above, $f^{\text{r}}_{\text{path}}(\mathcal{S})$ and $f^{\text{r}}_{\text{node}}(\mathcal{S})$ represent a reward term (r) for *every* path and node based on schedule $\mathcal{S}$, respectively, with generic reward functions $r^{\text{s}}_{\text{path}}(\cdot), r^{\text{p}}_{\text{path}}(\cdot), r^{\text{s}}_{\text{node}}(\cdot), r^{\text{p}}_{\text{node}}(\cdot)$ that separately reward safety margins (s) and system performance based on timing (p). Then, with safety-performance balancing parameters $\alpha_{\ell_i}, \alpha_{v_i}$, equations (7) and (8) can be interpreted as rewarding for each DAG path and node, a balance of exceeding timing constraints (safety margin) and system performance related to improved response time ($P(R_{\ell_i})$ and $P(R_{v_i})$). Finally, with all terms defined our scheduler can optimize our safety-performance metric defined as a weighted sum of terms (5)-(8), yielding efficiently computable schedules that trade off safety and system performance relative to probabilistic timing constraints.

REFERENCES

[1] Zahaf Houssam-Eddine, Nicola Capodieci, Roberto Cavicchioli, Giuseppe Lipari, and Marko Bertogna. The hpc-dag task model for heterogeneous real-time systems. *IEEE Transactions on Computers*, 2020.

[2] Donald W Gillies and Jane W-S Liu. Scheduling tasks with and/or precedence constraints. *SIAM Journal on Computing*, 24(4):797–810, 1995.

[3] Real-Time Innovations (RTI). DDS in Autonomous Car Design.

[4] Subhash C Sarin, Balaji Nagarajan, and Lingrui Liao. *Stochastic scheduling: expectation-variance analysis of a schedule*. Cambridge university press, 2010.

[5] Thomas Morton and David W Pentico. *Heuristic scheduling systems: with applications to production systems and project management*, volume 3. John Wiley & Sons, 1993.

[6] Sanjoy Baruah. Scheduling dags when processor assignments are specified. In *Proceedings of the 28th International Conference on Real-Time Networks and Systems*, RTNS 2020, page 111–116, New York, NY, USA, 2020. Association for Computing Machinery.

[7] Sanjoy Baruah and Alberto Marchetti-Spaccamela. Feasibility Analysis of Conditional DAG Tasks. In Björn B. Brandenburg, editor, *33rd Euromicro Conference on Real-Time Systems (ECRTS 2021)*, volume 196 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:17, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.