The Concept of Response Time Estimation Range for Optimizing Systems Scheduled with Fixed Priority

Yecheng Zhao and Haibo Zeng
Dept. Electrical and Computer Engineering, Virginia Tech, USA. **Email**: {zyecheng, hbzeng}@vt.edu

***Abstract*—Priority assignment is a critical issue in the design of real-time systems scheduled with fixed priority. In this paper, we consider the design optimization of a class of such real-time systems, where the schedulability of a task not only depends on the set of higher/lower priority tasks, but also on the response times of other tasks. This makes Audsley's algorithm inapplicable for finding a schedulable priority assignment, not to mention that the design optimization may also involve other metrics such as end-to-end latency in the constraints or objective. The current approaches are to either develop heuristics or use standard exhaustive search algorithms such as Branch-and-Bound (BnB).**

Instead, we propose a new approach that breaks through the mindset of the current approaches. Our main idea is to introduce the concept of response time estimation range, which is used in place of the actual response time of each task for evaluating the system schedulability. This allows to leverage Audsley's algorithm to generalize from an unschedulable solution to many similar ones. We develop an optimization framework that builds upon such a concept. We then apply the proposed approach to industrial designs of two use cases. One is the real-time wormhole communication in a Network-on-Chip (NoC), where a traffic flow suffers indirect interferences that depend on the response times of higher priority flows. The other is distributed systems with data-driven activation, where a task is triggered by the availability of data, hence by the completion of its immediate predecessor. Experimental results show the proposed technique typically runs several times faster than exhaustive search algorithms based on BnB or Mixed Integer Linear Programming (MILP).

I. Introduction

In real-time systems with fixed priority scheduling, appropriate priority assignment is essential for judicious utilization of hardware resources [10]. This problem has attracted a large body of research efforts. Classical results include rate-monotonic (RM) [15], deadline-monotonic (DM) [14], and Audsley's algorithm [1], all of which are tractable in that they only need to check a number of priority orders polynomial in the number of tasks. In particular, Audsley's algorithm is shown to be optimal for finding a schedulable priority assignment for a variety of scenarios, as long as they satisfy the conditions identified by Davis et al. [7]. The applicability of Audsley's algorithm subsumes those of RM and DM.

However, when Audsley's algorithm is inapplicable, there is no efficient algorithm to find the optimal priority assignment. This may rise for two reasons. One is that the system model and associated schedulability analysis violate the conditions of Audsley's algorithm. The other is that the problem may contain constraints or an objective related to other metrics (e.g., memory, power). The current mindset is to either invent problem specific heuristics that may be substantially suboptimal, or directly use standard search frameworks such as Branch-and-Bound (BnB) and Mixed Integer Linear Programming (MILP).

In this paper, we consider a class of real-time systems where the schedulability of a task depends not only on the set of higher/lower priority tasks but also on the response times of other tasks. We call them systems with Response Time (RT) dependency. RT dependency disrespects the applicable conditions of Audsley's algorithm. We provide two examples of such systems. The first is the real-time wormhole communication in a Network-on-Chip (NoC) [18]. A traffic flow may suffer indirect inferences from higher priority flows that do not directly share any communication link with it. The indirect interferences and consequently the response time of the flow depend on the response times of higher priority flows. The second is distributed systems with data-driven activation, which is broadly adopted in application domains such as automotive [26]. A task is activated by the availability of input data and consequently the completion of its immediate predecessor. Hence, the task has an activation jitter that equals the response time of the immediate predecessor.

Our approach breaks through the mindset of current methods. It is suitable for design optimization problems of systems with RT dependency that involve priority assignment as a design variable, e.g., to find a schedulable priority assignment. Specifically, we propose several concepts including the response time estimation range, which is used in place of the actual response time of each task for evaluating the system schedulability. These concepts allow to leverage Audsley's algorithm to generalize from an unschedulable solution to many similar ones. We then develop an optimization procedure that is guided by the proposed concepts. The procedure is shown to provide significantly better solutions than existing heuristics. Compared to other exact algorithms based on BnB or MILP, it typically runs many times faster.

The rest of the paper is organized as follows. Section II discusses the related work. Section III presents the system model. Section IV introduces the concepts and discusses their properties. Section V develops an optimization framework that leverages these concepts. Section VI presents the experimental results comparing our approach with existing methods, before concluding the paper in Section VII.

II. Related Work

The problem of priority assignment in real-time systems scheduled with fixed priority has been studied extensively. See an authoritative survey by Davis et al. [10]. In particular, for periodic tasks with implicit deadline (i.e., task deadline equals the period), RM is shown to be optimal for schedulability in that there is no system that can be scheduled by some priority assignment but not by RM. When tasks have constrained deadline (i.e., no larger than period), DM is optimal for schedulability [14]. Audsley's algorithm [1] is optimal for finding a schedulable priority assignment for a variety of task

models and scheduling policies, as summarized in Davis et al. [10]. The three necessary and sufficient conditions for its optimality are presented in [7]. Besides schedulability, Audsley's algorithm can be revised to optimize several other objectives, including the number of priority levels [1], lexicographical distance (the perturbation needed to make the system schedulable from an initial priority order) [4], [10], and robustness (ability to tolerate additional interferences) [6].

When Audsley's algorithm is not directly applicable, the current approaches can be classified into three categories. The *first* is based on meta heuristics such as simulated annealing (e.g., [19], [2]) and genetic algorithm (e.g., [12]). The *second* is to develop problem specific heuristics (e.g., [17], [22], [20]). These two categories do not have any guarantee on optimality. The *third* category is to search for the exact optimum, often applying standard optimization frameworks such as BnB (e.g., [21], [8]), or MILP (e.g., [27]). However, this approach typically suffers from scalability issues and may have difficulty to handle large industrial designs.

Such approaches are also followed for the particular class of systems we consider, i.e., systems with RT-dependency. For the real-time wormhole communication in NoC, Shi and Burns develop a BnB algorithm that is enhanced with some problem specific pruning techniques [18]. Later Nikolić and Pinho propose a unified MILP formulation for optimizing priority assignment and routing of traffic flows [16]. The mixed-criticality extension relies on heuristics including DM [3]. The current work on systems with data-driven activation either uses BnB [11] or formulates the problem in MILP [26].

Alternatively, we present efficient and exact optimization procedures that leverage Audsley's algorithm [23], [24]. However, these works [23], [24] are for problems where Audsley's algorithm is still optimal for finding a schedulable priority assignment. In this paper, we consider systems with RT dependency where the response time of a task also depends on those of other tasks. This makes Audsley's algorithm even inapplicable for schedulability. Thus, in this work we cannot use the concepts and algorithms from [23], [24].

Our approach differs from other exact algorithms, such as [18], [16] for real-time wormhole communication in NoC, in two significant ways. First, we transform the problem to a search in the space of response time estimation, instead of directly searching for an optimal priority assignment. Second, we use Audsley's algorithm to generalize from an unschedulable response time estimation to a maximal range. However, the approaches based on standard optimization frameworks [18], [16] lack this problem-specific "learning" capability.

III. System Model

We consider a real-time system Γ consisting of a set of tasks $\Gamma = \{\tau_1, \tau_2, ...\tau_n\}$ scheduled under fixed-priority. Each task τ_i is characterized by a tuple $\langle C_i, T_i, D_i \rangle$, where C_i is its Worst Case Execution Time (WCET), T_i is the period, and D_i is the deadline. Without loss of generality, we assume *these parameters are all positive integers*. τ_i is also associated with an activation jitter J_i^A and a unique priority level π_i. $\pi_i > \pi_j$ if τ_i has a higher priority than τ_j. $hp_i = \{\tau_j | \pi_j > \pi_i\}$ (resp. $lp_i = \{\tau_j | \pi_j < \pi_i\}$) denotes the set of tasks with priority higher (resp. lower) than τ_i.

In the optimization problem, we assume the task WCET, period, and deadline are given parameters. The decision variables include task priority assignment $\mathbf{P}$. Consequently, hp_i, lp_i, and the task worst case response time (or in short, *response time*) R_i of each task τ_i are also unknown variables. The activation jitter J_i^A, however, may be a known parameter as in the wormhole communication (Section III-B), or an unknown variable as for the case of data-driven activation (Section III-C). The *system schedulability*, i.e., each task shall meet its deadline, can be checked with an analysis that satisfies the property of *RT dependency* as defined in Section III-A. Besides, the optimization problem may also involve other constraints and objective.

Formally, the optimization problem $\mathbb{O}$ can be written as

$$\begin{aligned} \mathbb{O}: \quad & \min \quad g(\mathbf{X}) \\ & \text{s.t.} \quad \text{system schedulability} \\ & \quad\quad \mathbf{h}(\mathbf{X}) \leq 0 \end{aligned} \tag{1}$$

Here $\mathbf{X}$ is the set of decision variables that includes the vectors of task priority assignments $\mathbf{P}$ and response times $\mathbf{R} = [R_1, R_2, ...R_n]$. $\mathbf{h}(\cdot)$ is the set of constraints in addition to the system schedulability constraints, and $g(\cdot)$ is the objective function (the function to be optimized by assigning appropriate values to the variables that also satisfy all the constraints).

For example, one may be interested in maximizing the minimum laxity, the difference between the deadline and response time, among all tasks. This can be formulated as

$$\begin{aligned} \mathbb{O}: \quad & \max \quad L \\ & \text{s.t.} \quad \text{system schedulability} \\ & \quad\quad L \leq D_i - R_i, \forall i \end{aligned} \tag{2}$$

A. Response Time Dependency

We now formalize the property that the analysis for checking system schedulability shall satisfy, which we term as *response time dependency (RT dependency)*. Specifically, we assume the response time R_i of task τ_i can be computed as the least fixed point of the following equation, where the function $f_i(\cdot)$ takes hp_i and $\mathbf{R}$ as inputs

$$R_i = f_i\left(hp_i, \mathbf{R}\right), \quad \forall \tau_i \tag{3}$$

Note that if hp_i is given, lp_i is also fixed since $lp_i \bigcup hp_i = \Gamma \backslash \{\tau_i\}$. Thus, for simplicity we omit it in the definition of f_i.

The condition that the response time analysis can be written in the form of Eq. (3) implies the following two assumptions

- A1: the response time R_i of τ_i, *if the response times of other tasks are known*, depends on the set of higher priority tasks hp_i, but not on their relative order.
- A2: the response time R_i of τ_i, *if the response times of other tasks are known*, depends on the set of lower priority tasks lp_i, but not on their relative order.

We suppose (3) further satisfies two additional assumptions.

- A3: the response time R_i of τ_i is monotonically non-decreasing with the set of higher priority tasks hp_i. Specifically, given two sets of tasks hp_i and hp_i' such that $hp_i \subseteq hp_i'$, the response time R_i with hp_i' as the higher priority tasks is no smaller than that with hp_i.

- A4: the response time R_i of τ_i is monotonically non-decreasing with the increase of the response time R_j of any other task τ_j.

We now give the formal definition of RT dependency.

Definition 1. The response time analysis of a real-time system Γ is said to be response time dependent (**RT dependent**) if it satisfies the above four assumptions A1-A4.

Remark 1. Assumptions A1-A4 are desired properties for calculating R_i, i.e., the least fixed point solution to Eq. (3) where the input is the sets of higher/lower priority tasks and their response times, and the output is R_i.

As a more compact representation, let $\mathbf{hp}$ be the vector of higher priority task sets, and $\mathbf{f}(\mathbf{hp}, \mathbf{R})$ be a vector of functions over $\mathbf{hp}$ and $\mathbf{R}$. The response times of all tasks in the system are computed as the least fixed point of the following equation

$$\mathbf{R} = \mathbf{f}(\mathbf{hp}, \mathbf{R}) \tag{4}$$

We note that a response time analysis with RT dependency in general violates the applicability of Audsley's algorithm. Specifically, Audsley's algorithm requires that, in addition to A3 above, the following two conditions are satisfied [7]

- A1': the response time R_i of τ_i depends on the set of higher priority tasks hp_i, but not on their relative order.
- A2': the response time R_i of τ_i depends on the set of lower priority tasks lp_i, but not on their relative order.

To calculate R_i for τ_i by Eq. (3) in an RT dependent system, it requires the knowledge of R_j of some other task τ_j. But R_j is highly dependent on the relative priority of τ_j, which violates A1'-A2'. Despite this, we can still develop an efficient algorithm to optimize priority assignment in such systems.

We provide two examples that fit this model. One is the real-time wormhole communication in NoC [18] and its mixed-criticality extension [3] (detailed in Section III-B), the other is distributed systems with data-driven activation (Section III-C).

B. Real-Time Wormhole Communication in NoC

Wormhole switching with fixed priority preemptive scheduling is a viable solution for real-time communication in an NoC [18]. A wormhole-based NoC consists of multiple IP blocks and routers (i.e., the nodes in the network) connected through physical links to form a grid. Each traffic flow is transmitted through a specified path of links [18]. Here, the traffic flows are the scheduling entity, and the physical links are the hardware resource accessed by the traffic flows. Since a flow may access several physical links shared with other flows, priority-based arbitration at the routers is used to provide hard real-time service guarantees [18]. In the end, a flow suffers direct interferences, i.e., from those higher priority flows that directly share at least one link. In addition, it may also be delayed by indirect inferences, i.e., from those higher priority flows with no shared links. The indirect interferences are captured by the interference jitter introduced below.

The response time of traffic flow τ_i with constrained deadline is calculated as [18]

$$R_i = C_i + \sum_{\forall \tau_j \in shp_i} \left\lceil \frac{R_i + J_j^A + J_j^I}{T_j} \right\rceil C_j \tag{5}$$

Here shp_i is the set of higher priority traffic flows that share at least one link with τ_i. J_j^A is the activation jitter inherited from its sending software task, which can be considered as a parameter as it is independent from the priority assignment of the traffic flows. J_j^I represents the interference jitter of τ_j. J_j^I occurs when flows with higher priority than τ_j share a link with τ_j and thus indirectly interfere with τ_i. It is computed as

$$J_j^I = R_j - C_j \tag{6}$$

Substitute (6) into (5), there is

$$R_i = C_i + \sum_{\forall \tau_j \in shp_i} \left\lceil \frac{R_i + J_j^A + R_j - C_j}{T_j} \right\rceil C_j \tag{7}$$

We now show that the response time analysis in Eq. (7) satisfies the four assumptions A1-A4. Note that the set shp_i is solely dependent on the set of higher priority flows hp_i, as it is the intersection of hp_i and the *fixed* set of flows sharing a link with τ_i. For A1 and A2, if the response time R_j of all other tasks τ_j are known, the fixed point solution of Eq. (7) depends only on the set shp_i but not on other design variables. For A3, since only the response times of tasks in shp_i contribute positively to the right hand side of (7), a larger hp_i (and consequently a larger shp_i) will only make R_i larger. Likewise, in the right hand side of (7), the response time R_j of any other task τ_j has an coefficient that is either zero (in case it is not in shp_i) or positive (in case it belongs to shp_i), A4 is also satisfied.

Hence, the analysis in Eq. (7) is RT dependent. However, Audsley's algorithm is not applicable even for finding a schedulable priority order [18]. Similarly, the NoC system with mixed-criticality traffic flows also relies on a response time analysis that violates the conditions for Audsley's algorithm but satisfies A1-A4 [3].

C. Data-Driven Activation

Data-driven activation is an activation model in distributed systems such as automotive applications [26]. In this model, task executions and message transmissions are triggered by the arrival of input data. Hence, each task/message τ_i is characterized by a period T_i, an activation jitter J_i^A, and a deadline that is relative to the arrival time (i.e., the expected activation time). In a communication chain, the activation jitter inherits the worst case response time of the immediate predecessor: if τ_j is triggered by the arrival of data sent by τ_k, then $J_j^A = R_k$. This dependency is often referred to as *jitter propagation*. Note that the direct communication happens only between a task and another task/message, but not between two messages. However, indirect communication, and consequently jitter propagation dependency, may still exist between two messages.

With respect to the scheduling policy, tasks and messages are both assigned with a fixed priority. However, tasks are preemptive while messages are non-preemptive (like in the

TABLE I: An Example Wormhole NoC System Γ_e (all flows share the same link)

τ_i	T_i	C_i	D_i	J_i^A
τ_1	10	4	10	0
τ_2	35	9	35	0
τ_3	120	5	120	0
τ_4	180	35	180	0

Controller Area Network bus). For a task/message τ_i, the *queuing delay* w_i is defined as the worst case delay from τ_i's activation to its completion. The response time R_i of τ_i with preemptive scheduling and constrained deadline is

$$\begin{aligned} w_i &= C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i + J_j^A}{T_j} \right\rceil C_j \\ R_i &= J_i^A + w_i \end{aligned} \tag{8}$$

When there is direct or indirect communication from τ_j to τ_i sharing the same resource (CPU or bus), J_i^A and consequently R_i rely on the exact value of R_j. The response time for non-preemptive scheduling or tasks with arbitrary deadlines requires to evaluate all instances in the busy period [26]. Still, they satisfy the properties of RT dependency.

The end-to-end latency L_p of a path p is the sum of the queuing delays of all tasks/messages on p

$$L_p = \sum_{\forall \tau_j \text{ on path } p} w_i \tag{9}$$

L_p shall be no larger than the end-to-end deadline D_p of p.

IV. Response Time Estimation Range

The response time analysis in Eq. (3) does not satisfies the conditions of Audsley's algorithm. However, it has a useful property as in the assumptions A1-A2: if the response times of other tasks are appropriately estimated, then R_i only requires the knowledge on the set of higher/lower priority tasks, but not on the relative order in them. This, combined with A3, allows to leverage Audsley's algorithm assuming we can appropriately estimate the response times.

Hence, the main idea of our optimization framework is to use a separate procedure to search for an appropriate response time estimation. Before presenting the details of the optimization framework in the next section, in this section we first define a few concepts and study their useful properties. We illustrate with an example NoC system in Table I, which has four tasks (traffic flows) sharing the same link. We also summarize the response time related concepts in Table II.

Definition 2. A **response time estimation (RTE)** is a collection of tuple elements $\langle \tau_i, r_i \rangle$ for all tasks, i.e., $\mathcal{E} = \{\langle \tau_1, r_1 \rangle, ... \langle \tau_n, r_n \rangle\}$, where each tuple $\langle \tau_i, r_i \rangle$ with $r_i \in [C_i, D_i]$ represents that the response time of τ_i is estimated to be r_i.

The following concept defines the way how a response time estimation is used during optimization. Essentially, given an RTE $\mathcal{E}$, we define the estimation-inferred response time of task τ_i as the least fixed point to Eq. (3) assuming the response times of other tasks follow the estimated value in $\mathcal{E}$.

TABLE II: Concepts Related to Response Time

Notation	Concept	Definition
R_i	Actual response time of τ_i	
$\mathbf{R}$	Vector of actual task response times	
$\mathcal{E}$	Response time estimation (RTE)	Defn. 2
R_i^E	Estimation-inferred response time of τ_i	Defn. 3
$\mathbf{R}^E$	Vector of estimation-inferred response times	Defn. 3
$\mathbf{E}_i$	Use of response time estimation for computing R_i^E	Eqn. (11)
$\mathcal{G}$	Response time estimation range	Defn. 5
R_i^G	Estimation range-inferred response time of τ_i	Defn. 8
$\mathbf{R}^G$	Vector of estimation range-inferred response times	Defn. 8
$\mathbf{G}_i$	Use of response time estimation range for computing R_i^G	Eqn. (19)
$\mathcal{U}$	Maximal unschedulable response time estimation range (MUTER)	Defn. 10

Definition 3. Given a priority assignment $\mathbf{P}$ and a response time estimation $\mathcal{E} = \{\langle \tau_1, r_1 \rangle, ... \langle \tau_n, r_n \rangle\}$, the **estimation-inferred response time** of task τ_i, denoted as R_i^E, is the least fixed point solution of the following equation

$$R_i^E = f_i(hp_i, \mathbf{E}_i) \tag{10}$$

Here $\mathbf{E}_i$ is a vector constructed as follows: the i-th entry of $\mathbf{E}_i$ is the variable R_i^E, and the j-th entry for any $j \neq i$ takes the corresponding given value r_j in $\mathcal{E}$, i.e.,

$$\mathbf{E}_i = [r_1, ..., R_i^E, ..., r_n] \tag{11}$$

The **vector of estimation-inferred response times** is denoted as $\mathbf{R}^E$.

Remark 2. With a given RTE $\mathcal{E}$, the calculation of estimation-inferred response time R_i^E for task τ_i only depends on the set of higher priority tasks, but not on their relative order. Also, given a priority assignment, the actual response time is generally different from the estimation-inferred response time. This is illustrated in the following example.

Example 1. Assume the priority order is $\pi_1 > \pi_2 > \pi_3 > \pi_4$ for the system in Table I. The actual task response times are the least fixed point solution for the following equations

$$\begin{cases} R_1 = & 4 \\ R_2 = & 9 + \left\lceil \frac{R_2 + R_1 - 4}{10} \right\rceil \cdot 4 \\ R_3 = & 5 + \left\lceil \frac{R_3 + R_1 - 4}{10} \right\rceil \cdot 4 + \left\lceil \frac{R_3 + R_2 - 9}{35} \right\rceil \cdot 9 \\ R_4 = & 35 + \left\lceil \frac{R_4 + R_1 - 4}{10} \right\rceil \cdot 4 + \left\lceil \frac{R_4 + R_2 - 9}{35} \right\rceil \cdot 9 \\ & + \left\lceil \frac{R_4 + R_3 - 5}{120} \right\rceil \cdot 5 \end{cases} \tag{12}$$

Hence, the vector of actual response times is $\mathbf{R} = [4, 17, 26, 150]$. However, the estimation-inferred response times, given an RTE $\mathcal{E} = \{\langle \tau_1, 4 \rangle, \langle \tau_2, 9 \rangle, \langle \tau_3, 5 \rangle, \langle \tau_4, 35 \rangle\}$, are the least fixed point of the equations below

$$\begin{cases} R_1^E = & 4 \\ R_2^E = & 9 + \left\lceil \frac{R_2^E + 4 - 4}{10} \right\rceil \cdot 4 \\ R_3^E = & 5 + \left\lceil \frac{R_3^E + 4 - 4}{10} \right\rceil \cdot 4 + \left\lceil \frac{R_3^E + 9 - 9}{35} \right\rceil \cdot 9 \\ R_4^E = & 35 + \left\lceil \frac{R_4^E + 4 - 4}{10} \right\rceil \cdot 4 + \left\lceil \frac{R_4^E + 9 - 9}{35} \right\rceil \cdot 9 \\ & + \left\lceil \frac{R_4^E + 5 - 5}{120} \right\rceil \cdot 5 \end{cases} \tag{13}$$

Hence, the vector of estimation-inferred response times is $\mathbf{R}^E = [4, 17, 26, 137]$. We note that $R_4^E \neq R_4$.

We now define a desired property of an RTE such that it allows a schedulable priority assignment (Definition 4). We prove that there must exist an RTE with this property if and only if the system is schedulable (Theorem 1).

Definition 4. A response time estimation $\mathcal{E} = \{\langle \tau_1, r_1 \rangle, ... \langle \tau_n, r_n \rangle\}\}$ is defined as **schedulable** if and only if there exists a priority assignment $\mathbf{P}$ such that the estimation-inferred response times are component-wise no larger than $\mathcal{E}$. That is, $\mathcal{E}$ is schedulable if and only if

$$\exists \mathbf{P} \ s.t. \ \ \forall i = 1..n, R_i^E = f_i(hp_i, \mathbf{E}_i) \leq r_i \qquad (14)$$

Theorem 1. A system Γ has a schedulable priority assignment if and only if there exists a schedulable RTE $\mathcal{E}$.

Proof. "Only If" part. Let $\mathbf{P}$ be a schedulable priority assignment of Γ and $\mathbf{R} = [R_1, ...R_n]$ be the actual response time under $\mathbf{P}$. Take $r_i = R_i$ for each $\langle \tau_i, r_i \rangle$ in $\mathcal{E}$. Then $\mathcal{E}$ and $\mathbf{P}$ satisfy Eq. (14), where the estimation-inferred response time for any τ_i is $R_i^E = r_i$. Hence, $\mathcal{E}$ is schedulable.

"If" part. Let $\mathcal{E}$ and $\mathbf{P}$ be the response time estimation and priority assignment that satisfy the condition (14). By (14), $\mathbf{R}^E$ is component-wise no larger than $\mathcal{E}$, or $\mathbf{R}^E \leq \mathcal{E}$. Hence, for any τ_i, $\mathbf{R}^E$ is component-wise no larger than $\mathbf{E}_i$, i.e., $\mathbf{R}^E \leq \mathbf{E}_i$. Combine the monotonicity of f_i with respect to the task response times (assumption A4) and Eq. (14), there is

$$\forall i, \quad f_i(hp_i, \mathbf{R}^E) \leq f_i(hp_i, \mathbf{E}_i) = R_i^E \leq r_i \qquad (15)$$

Now consider the vector function $\mathbf{f}(\mathbf{hp}, \mathbf{R})$, there is

$$\mathbf{f}(\mathbf{hp}, \mathbf{R}^E) \leq \mathbf{R}^E \leq \mathcal{E} \qquad (16)$$

This implies that the least fixed point of (4), i.e., the actual response times $\mathbf{R}$, must be component-wise no larger than $\mathbf{R}^E$ and consequently $\mathcal{E}$. Hence, Γ is schedulable with $\mathbf{P}$. □

Remark 3. This paper focuses on developing optimization algorithms while assuming a given schedulability analysis (that satisfies A1-A4). Hence, the notions of "schedulability" (e.g., Definition 4 and Theorem 1) and "optimality" (e.g., Figure 1 and Theorem 4) throughout the paper are defined w.r.t. the given schedulability analysis. When the given schedulability analysis is sufficient only, it is possible that our optimization algorithm deems the system to be unschedulable but there actually exists a truly schedulable priority assignment.

Theorem 1 suggests that instead of directly searching for a feasible priority assignment, an alternative is to search for a response time estimation $\mathcal{E}$ that satisfies condition (14). Checking $\mathcal{E}$ against (14) can be easily verified using Audsley's algorithm, as the estimation-inferred response time depends only on the set of higher priority tasks but not on their relative order. With this observation, a straightforward algorithm would be to examine all possible response time estimations to find a schedulable one. However, this is obviously impractical, as the total number of response time estimations is $O(\prod_i D_i)$.

We now introduce a search space reduction technique that drastically improves the algorithm efficiency. The intuition is a large number of unschedulable response time estimations share common reasons that cause the unschedulability. Thus, it is possible to generalize from one unschedulable response time estimation to a range of them, all of which can be removed from the search space together. We first discuss two motivating examples before giving the formal definition.

Example 2. For the system in Table I, consider an RTE $\mathcal{E} = \{\langle \tau_1, 4 \rangle, \langle \tau_2, 9 \rangle, \langle \tau_3, 30 \rangle, \langle \tau_4, 180 \rangle\}$. $\mathcal{E}$ is unschedulable, as τ_1 must have a higher priority than τ_2, and the response time of τ_2 must be no smaller than $C_1 + C_2 = 13$. In fact, any RTE $\mathcal{E} = \{\langle \tau_2, 3 \rangle, \langle \tau_2, r_2 \rangle, \langle \tau_3, 30 \rangle, \langle \tau_4, 180 \rangle\}$ where $r_2 \leq 12$ is unschedulable.

Example 3. Consider another RTE $\mathcal{E} = \{\langle \tau_1, 10 \rangle, \langle \tau_2, 17 \rangle, \langle \tau_3, 30 \rangle, \langle \tau_4, 180 \rangle\}$. $\mathcal{E}$ is unschedulable for the following reasons. τ_2 cannot be assigned a higher priority than τ_1. Hence, in the best case, it is assigned the second highest priority after τ_1. The estimation-inferred response time R_2^E w.r.t. to $\mathcal{E}$ is the least fixed point of

$$R_2^E = 9 + \left\lceil \frac{R_2^E + 10 - 4}{10} \right\rceil \cdot 4 \qquad (17)$$

This gives $R_2^E = 21$, which exceeds $r_2 = 17$ in $\mathcal{E}$. In fact, unless the response time estimation for τ_1 is no larger than 7, R_2^E will be at least 21. Thus, it can be inferred that any RTE $\mathcal{E} = \{\langle \tau_1, r_1 \rangle, \langle \tau_2, 17 \rangle, \langle \tau_3, 30 \rangle, \langle \tau_4, 180 \rangle\}$ where $r_1 \in [8, 10]$ should be unschedulable.

The above two examples illustrate the two conflicting requirements of an RTE. Consider any element $\langle \tau_i, r_i \rangle$ in an RTE $\mathcal{E}$. r_i is used in two distinct places in condition (14): **(a)** at the right hand side of the i-th inequality in (14), where it acts like a virtual deadline that the estimation-inferred response time R_i^E of τ_i shall not violate; **(b)** as a constant entry in vector $\mathbf{E}_j$ for any other $j \neq i$, where the estimation-inferred response time R_j^E is non-decreasing with a larger r_i. Obviously, it is desirable to have a larger r_i for **(a)** but a smaller r_i for **(b)**. These examples also suggest that an unschedulable RTE can be generalized to a range of response time estimations such that any of them is unschedulable too.

Hence, we consider splitting the estimation r_i into an optimistic estimation r_i^l and a pessimistic one r_i^u, where $r_i^l \leq r_i^u$. Instead of using r_i, we now use r_i^u for **(a)** and r_i^l for **(b)**. This results in a weaker condition than (14), as detailed in Eq. (21). Suppose that this weaker condition does not even allow a schedulable priority assignment, then it can be implied that any r_i in the range $[r_i^l, r_i^u]$ would be unschedulable for the original condition (14).

We now formalize the idea in the following definitions.

Definition 5. A **response time estimation range** $\mathcal{G}$ is a collection of tuple elements $\langle \tau_i, [r_i^l, r_i^u] \rangle$ for each task τ_i, i.e., $\mathcal{G} = \{\langle \tau_1, [r_1^l, r_1^u] \rangle, .. \langle \tau_n, [r_n^l, r_n^u] \rangle\}$, where $C_i \leq r_i^l \leq r_i^u \leq D_i$. It represents a range of possible estimation values for the actual response time R_i of each task τ_i.

Note that in the definition, we restrict $[r_i^l, r_i^u]$ of each task τ_i to be within $[C_i, D_i]$, as the response time R_i of τ_i for any schedulable system shall be in that range.

Definition 6. A response time estimation $\mathcal{E}$ is said to be **contained** in a response time estimation range $\mathcal{G}$, denoted as $\mathcal{E} \in \mathcal{G}$, if and only if for each $\langle \tau_i, r_i \rangle$ in $\mathcal{E}$, the corresponding range $\langle \tau_i, [r_i^l, r_i^u] \rangle$ in $\mathcal{G}$ satisfies $r_i \in [r_i^l, r_i^u]$.

Example 4. Consider the following response time estimations and response time estimation range.

$$\mathcal{E}_1 = \{\langle \tau_1, 4\rangle, \langle \tau_2, 13\rangle, \langle \tau_3, 30\rangle, \langle \tau_4, 180\rangle\}$$
$$\mathcal{E}_2 = \{\langle \tau_1, 4\rangle, \langle \tau_2, 17\rangle, \langle \tau_3, 30\rangle, \langle \tau_4, 180\rangle\}$$
$$\mathcal{G} = \{\langle \tau_1, [4, 9]\rangle, \langle \tau_2, [9, 16]\rangle, \langle \tau_3, [5, 120]\rangle, \langle \tau_4, [35, 180]\rangle\}$$

$\mathcal{E}_1 \in \mathcal{G}$, as each response time estimated in $\mathcal{E}_1$ is contained in the corresponding range in $\mathcal{G}$. $\mathcal{E}_2 \notin \mathcal{G}$, as the response time estimation of τ_2 is 17, outside the corresponding range in $\mathcal{G}$.

Definition 7. $\mathcal{G}_1$ is a **subset** of $\mathcal{G}_2$, or equivalently $\mathcal{G}_2$ is a **superset** of $\mathcal{G}_1$, denoted as $\mathcal{G}_1 \subseteq \mathcal{G}_2$, if and only if for each $\langle \tau_i, [r^l_{i1}, r^u_{i1}]\rangle$ in $\mathcal{G}_1$, the corresponding $\langle \tau_i, [r^l_{i2}, r^u_{i2}]\rangle$ in $\mathcal{G}_2$ satisfies $r^l_{i2} \leq r^l_{i1}$ and $r^u_{i2} \geq r^u_{i1}$. $\mathcal{G}_1$ is a **strict subset** of $\mathcal{G}_2$, denoted as $\mathcal{G}_1 \subset \mathcal{G}_2$, if and only if $\mathcal{G}_1 \subseteq \mathcal{G}_2$ and $\mathcal{G}_1 \neq \mathcal{G}_2$.

Example 5. For the response time estimation ranges below

$$\mathcal{G}_1 = \{\langle \tau_1, [4, 10]\rangle, \langle \tau_2, [13, 17]\rangle, \langle \tau_3, [5, 30]\rangle, \langle \tau_4, [35, 180]\rangle\}$$
$$\mathcal{G}_2 = \{\langle \tau_1, [4, 10]\rangle, \langle \tau_2, [9, 17]\rangle, \langle \tau_3, [5, 30]\rangle, \langle \tau_4, [35, 180]\rangle\}$$

obviously $\mathcal{G}_1 \subseteq \mathcal{G}_2$ since each element in $\mathcal{G}_2$ is a superset of the corresponding one in $\mathcal{G}_1$.

Definition 8. Given a response time estimation range $\mathcal{G} = \{\langle \tau_1, [r^l_1, r^u_1], ..[r^l_n, r^u_n]\rangle\}$, and a priority assignment $\mathbf{P}$, the **estimation range-inferred response time** of τ_i, denoted as R^G_i, is the least fixed point of the following equation

$$R^G_i = f_i(hp_i, \mathbf{G}_i) \tag{18}$$

where $\mathbf{G}_i$ is a vector constructed by taking the i-th entry as variable R^G_i and any other j-th entry as the value r^l_j from $\mathcal{G}$

$$\mathbf{G}_i = [r^l_1, ..., R^G_i, ..., r^l_n] \tag{19}$$

The **vector of estimation range-inferred response times** is denoted as $\mathbf{R}^G$.

Intuitively, due to property A4, the estimation range-inferred response time is essentially the smallest estimation inferred response time that can possibly be obtained for $\mathcal{E} \in \mathcal{G}$, as shown in the following equation.

$$\begin{aligned} &\forall \mathcal{E} \in \mathcal{G},\ \forall i,\ \forall j \neq i,\ r_j \geq r^l_j \\ &\Rightarrow \forall \mathcal{E} \in \mathcal{G},\ \forall i,\ R^E_i \geq R^G_i \qquad \text{(by property A4)} \end{aligned} \tag{20}$$

Also, given an estimation range, the analysis of estimation range-inferred response times depends only on the set of higher priority tasks but not on their relative order, hence it is compliant with Audsley's algorithm.

We now define the schedulability of a response time estimation range as follows.

Definition 9. A response time estimation range $\mathcal{G} = \{\langle \tau_1, [r^l_1, r^u_1]\rangle, ... \langle \tau_n, [r^l_n, r^u_n]\rangle\}$ is said to be **schedulable** if

$$\exists \mathbf{P}\ s.t.\ \ \forall i = 1,\ R^G_i = f_i(hp_i, \mathbf{G}_i) \leq r^u_i \tag{21}$$

Condition (21) is weaker than (14): it allows r^l_i to be smaller than r^u_i, hence easier to be satisfied than (14). Like (14), (21) can be verified efficiently using Audsley's algorithm.

The usefulness of the concept is shown in the following theorem, which demonstrates that an unschedulable response time estimation range implies all its contained response time estimations are unschedulable.

Theorem 2. Given an unschedulable response time estimation range $\mathcal{G} = \{\langle \tau_1, [r^l_1, r^u_1]\rangle, ... \langle \tau_n, [r^l_n, r^u_n]\rangle\}$, any response time estimation $\mathcal{E} \in \mathcal{G}$ is unschedulable.

Proof. Let $\mathcal{E} = \{\langle \tau_1, r_1\rangle, ... \langle \tau_n, r_n\rangle\}$ be any response time estimation contained in $\mathcal{G}$. If $\mathcal{G}$ is unschedulable, then for any priority assignment $\mathbf{P}$, there exists a task τ_i such that

$$\begin{aligned} R^G_i \geq r^u_i \Rightarrow &\quad R^E_i \geq r^u_i \quad \text{[by Eq. (20)]} \\ \Rightarrow &\quad R^E_i \geq r_i \quad \text{(since } \mathcal{E} \in \mathcal{G}\text{)} \end{aligned} \tag{22}$$

Hence, $\mathcal{E}$ is also unschedulable. □

Unlike the case of unschedulable response time estimation range, its schedulable version is less useful in the sense that the contained RTE may or may not be schedulable. We illustrate in the following example.

Example 6. Consider the following two response time estimation ranges

$$\mathcal{G}_1 = \{\langle \tau_1, [4, 10]\rangle, \langle \tau_2, [9, 16]\rangle, \langle \tau_3, [5, 120]\rangle, \langle \tau_4, [35, 180]\rangle\}$$
$$\mathcal{G}_2 = \{\langle \tau_1, [4, 10]\rangle, \langle \tau_2, [9, 35]\rangle, \langle \tau_3, [5, 120]\rangle, \langle \tau_4, [35, 180]\rangle\}$$

$\mathcal{G}_1$ is unschedulable by reasons similar to those in Example 3. Specifically, τ_1 has to have a higher priority than τ_2. Hence, the response time of τ_2 is at least 17, since it will suffer interferences from at least two instances of τ_1 even we estimate the response time of τ_1 with the lower bound 4. This deems that $R^G_2 \leq 16$ cannot be satisfied, and $\mathcal{G}_1$ is unschedulable.

On the other hand, $\mathcal{G}_2$ is schedulable as there exists a priority assignment $\pi_1 > \pi_2 > \pi_3 > \pi_4$ which makes Eq. (21) true. The corresponding estimation range-inferred response times are $\mathbf{R}^G = [4, 17, 26, 137]$. However $\mathcal{G}_2$ cannot infer if an RTE contained in it is schedulable or not. For example, the RTE $\mathcal{E}_2 = \{\langle \tau_1, 10\rangle, \langle \tau_2, 17\rangle, \langle \tau_3, 30\rangle, \langle \tau_4, 180\rangle\}$ is unschedulable (as discussed in Example 3). However, $\mathcal{E}'_2 = \{\langle \tau_1, 4\rangle, \langle \tau_2, 17\rangle, \langle \tau_3, 26\rangle, \langle \tau_4, 150\rangle\}$ is schedulable since it allows a schedulable priority assignment $\pi_1 > \pi_2 > \pi_3 > \pi_4$ (as inferred from Example 1). Note that both $\mathcal{E}_2$ and $\mathcal{E}'_2$ are contained in $\mathcal{G}_2$ which is schedulable.

We now define a class of unschedulable response time estimation ranges which are not a strict subset of any other unschedulable ones. This can maximize its contained unschedulable RTEs.

Definition 10. A response time estimation range $\mathcal{U}$ is a **Maximal Unschedulable response Time Estimation Range (MUTER)** if and only if it satisfies the following conditions

- $\mathcal{U}$ is unschedulable by Definition 9;
- For all $\mathcal{G}$ such that $\mathcal{U} \subset \mathcal{G}$, $\mathcal{G}$ is schedulable.

Remark 4. The term "maximal" here is consistent with the typical terminology in order theory[1]. Specifically, consider a subset $\mathbb{S}$ of some *partially* ordered set. A maximal element m of $\mathbb{S}$ is an element of $\mathbb{S}$ that is no smaller than any other element s in $\mathbb{S}$, i.e., $s \leq m, \forall s \in \mathbb{S}$. There are a couple of notable properties for this definition. First, if $m \in \mathbb{S}$ is a maximal element of $\mathbb{S}$, then for all $s \in \mathbb{S}$ such that $m \leq s$, it must be $m = s$. Second, it is possible that there are many maximal elements of $\mathbb{S}$, since $\mathbb{S}$ does not impose a total order.

[1] http://en.wikipedia.org/wiki/Maximal_and_minimal_elements

Algorithm 1 Algorithm for Computing MUTER

1: **function** **MUTER**(unschedulable RTE $\mathcal{E} = \{\langle \tau_1, r_1 \rangle, ... \langle \tau_n, r_n \rangle\}$)
2: $\mathcal{G} = \{\langle \tau_1, [r_1, r_1] \rangle, ... \langle \tau_n, [r_n, r_n] \rangle\}$
3: **for** each $\langle \tau_i, [r_i^l, r_i^u] \rangle \in \mathcal{G}$ **do**
4: Use binary search to find out the smallest value that r_i^l can be decreased to while keeping $\mathcal{G}$ unschedulable.
5: Use binary search to find out the largest value that r_i^u can be increased to while keeping $\mathcal{G}$ unschedulable.
6: **end for**
7: return $\mathcal{G}$
8: **end function**

For the concept of response time estimation range, we shall treat the subset relationship in Definition 7 as a partial order, i.e., $\mathcal{G}_1$ is no larger than $\mathcal{G}_2$ if $\mathcal{G}_1 \subseteq \mathcal{G}_2$. Let $\mathbb{S}$ be the set of all *unschedulable* response time estimation ranges. A MUTER $\mathcal{U}$ by Definition 10 is essentially a "maximal" element of $\mathbb{S}$. Since the partial order in Definition 7 does not form a total order among response time estimation ranges, there may be multiple maximal elements of $\mathbb{S}$, i.e., multiple MUTERs.

Example 7. Consider the response time estimation ranges

$$\mathcal{G}_3 = \{\langle \tau_1, [4, 10] \rangle, \langle \tau_2, [9, 13] \rangle, \langle \tau_3, [5, 120] \rangle, \langle \tau_4, [35, 180] \rangle\}$$

and $\mathcal{G}_1$, the latter is defined in Example 6. Though both $\mathcal{G}_1$ and $\mathcal{G}_3$ are unschedulable, $\mathcal{G}_3$ is not a MUTER since $\mathcal{G}_3 \subset \mathcal{G}_1$. $\mathcal{G}_1$ is a MUTER since increasing r_2^u from 16 to 17 will make it schedulable, and the other bounds in $\mathcal{G}_1$ cannot be expanded either: the lower bound r_i^l is equal to the smallest value C_i, and the upper bound r_i^u is the same as the largest value D_i.

Intuitively, consider two unschedulable response time estimation ranges $\mathcal{G}_1$ and $\mathcal{G}_2$ such that $\mathcal{G}_1 \subseteq \mathcal{G}_2$. $\mathcal{G}_1$ is redundant in the presence of $\mathcal{G}_2$, since the latter contains all unschedulable RTEs contained in $\mathcal{G}_1$. In this sense, a MUTER $\mathcal{U}$ is more useful than any of its subset $\mathcal{G}$ (i.e., $\mathcal{G} \subseteq \mathcal{U}$), since it is more efficient than $\mathcal{G}$ in capturing unschedulable RTEs. In the optimization algorithm design, this allows to rule out the most unschedulable RTEs with the fewest number of unschedulable response time estimation ranges.

An unschedulable RTE $\mathcal{E} = \{\langle \tau_1, r_1 \rangle, ... \langle \tau_n, r_n \rangle\}$ can be generalized into a MUTER by Algorithm 1. We assume that the initial input $\mathcal{E}$ satisfies $r_i \in [C_i, D_i]$ for each task τ_i. We will later show in Section V how it can be guaranteed. The algorithm first converts the RTE to a response time estimation range $\mathcal{G} = \{\langle \tau_1, [r_1, r_1] \rangle, ... \langle \tau_n, [r_n, r_n] \rangle\}$ containing only $\mathcal{E}$ itself (Line 2). Then it leverages the property that condition (21) is monotonic w.r.t. each r_i^l and r_i^u: increasing r_i^u or decreasing r_i^l can only make (21) easier to satisfy. It uses binary search to find out the minimum value that r_i^l can be decreased to (or the maximum value r_i^u can be increased to) while maintaining the unschedulability of $\mathcal{G}$ (Lines 3–6).

Specifically, at Line 4, the algorithm preserves the values of r_i^u and all other response time estimation ranges $\langle \tau_j, [r_j^l, r_j^u] \rangle, i \neq j$, and uses binary search to decrease r_i^l as much as $\mathcal{G}$ is unschedulable. By Definition 3, r_i^l must be no smaller than C_i. Also, r_i^l has an initial value r_i which is known to be unschedulable. Thus, the initial lower and upper bounds for the binary search of r_i^l are C_i and r_i respectively. The binary search stops when the lower and upper bounds converge (i.e., their difference is less than 1, which is sufficient since all response time estimations are integers). Line 5 is similar except that (a) it increases r_i^u, while keeping r_i^l at the value determined by Line 4; (b) the initial lower and upper bounds for r_i^u are r_i and D_i respectively.

At Lines 4 and 5, whether or not the resulting $\mathcal{G}$ is schedulable is checked using Audsley's algorithm, i.e., to see if it permits a priority assignment that satisfies (21). Note that Algorithm 1 will always terminate since r_i^l is bounded below by C_i and r_i^u is bounded above by D_i.

We now formally prove that Algorithm 1 always returns a correct MUTER according to Definition 10.

Theorem 3. Given an unschedulable response time estimation $\mathcal{E} = \{\langle \tau_1, r_1 \rangle, ... \langle \tau_n, r_n \rangle\}$, Algorithm 1 correctly computes a MUTER $\mathcal{G}$.

Proof. We show that the returned response time range $\mathcal{G}$ satisfies the two conditions in Definition 10. First, $\mathcal{G}$ is initially unschedulable, and Algorithm 1 updates $\mathcal{G}$ only if $\mathcal{G}$ is maintained to be unschedulable. Thus, the returned $\mathcal{G}$ is guaranteed to be unschedulable.

We prove the second condition by contradiction. Let $\mathcal{G}'$ be an unschedulable response time estimation range such that $\mathcal{G}'$ is a strict superset of the returned $\mathcal{G}$, i.e., $\mathcal{G} \subset \mathcal{G}'$. Hence, there must exist a task τ_i such that the corresponding response time range element $\langle \tau_i, [r_i^l(\mathcal{G}'), r_i^u(\mathcal{G}')] \rangle$ in $\mathcal{G}'$ strictly contains the one $\langle \tau_i, [r_i^l(\mathcal{G}), r_i^u(\mathcal{G})] \rangle$ in $\mathcal{G}$. That is, at least one of the two conditions $r_i^l(\mathcal{G}') < r_i^l(\mathcal{G})$ and $r_i^u(\mathcal{G}) < r_i^u(\mathcal{G}')$ is satisfied. Now consider the resulting response time estimation range $\mathcal{G}_i$ after $\langle \tau_i, [r_i^l, r_i^u] \rangle$ is processed by Algorithm 1. Since the algorithm only expands $\mathcal{G}$ during each iteration, it must be $\mathcal{G}_i \subseteq \mathcal{G} \subset \mathcal{G}'$. We construct another response time estimation range $\mathcal{G}_i'$ such that (a) for each $j \neq i$, $r_j^l(\mathcal{G}_i') = r_j^l(\mathcal{G}_i)$ and $r_j^u(\mathcal{G}_i') = r_j^u(\mathcal{G}_i)$; (b) $r_i^l(\mathcal{G}_i') = r_i^l(\mathcal{G}')$ and $r_i^u(\mathcal{G}_i') = r_i^u(\mathcal{G}')$. Obviously $\mathcal{G}_i \subset \mathcal{G}_i' \subseteq \mathcal{G}'$. This contradicts with the fact that $\mathcal{G}_i$ is the result after $\langle \tau_i, [r_i^l, r_i^u] \rangle$ is maximally expanded. □

In Algorithm 1, Audsley's algorithm is called each time we try to check if $\mathcal{G}$ is schedulable. It is known that Audsley's algorithm only needs to explore $O(n^2)$ priority orders out of the $n!$ possible ones, where n is the number of tasks [10]. For each task τ_i, the binary searches at Lines 4–5 make $O(\log D_i)$ function calls to Audsley's algorithm, since there are at most D_i possible integer values for both r_i^l and r_i^u. Hence, Algorithm 1 checks a total of $O(n^2 \log D)$ priority orders to calculate a MUTER, where $D = \prod_i D_i$.

V. MUTER-Guided Optimization Algorithm

We now present an optimization algorithm that leverages the concepts of RTE and MUTER. We observe that in many optimization problems for systems with an RT-dependent response time analysis, finding a schedulable priority assignment is a major difficulty since there is no known tractable procedure like Audsley's algorithm. As in Theorem 1, finding a schedulable priority assignment is equivalent to finding a schedulable RTE. The latter has the promise to be more scalable for the following unique capability from Algorithm 1: given an unschedulable RTE it can efficiently generalize to a set of MUTERs, each of which contains a maximal range of unschedulable RTEs.

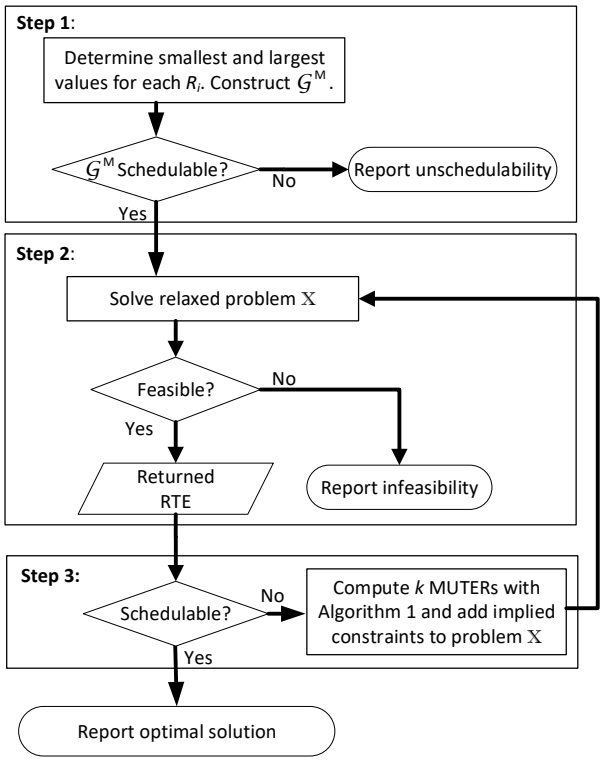

Fig. 1: Optimal Priority Assignment Algorithm Procedure

Hence, we design the optimization algorithm that leverages the power of Algorithm 1. Specifically, instead of solving the original problem $\mathbb{O}$ directly, we start with a relaxed problem $\mathbb{X}$ that leaves out all the system schedulability constraints. Solving this relaxed problem will return an RTE that is driven by other constraints and the objective. If the RTE is unschedulable, we use Algorithm 1 to generalize to a set of MUTERs. We then add the corresponding constraints to $\mathbb{X}$ to rule out similar unschedulable RTEs, and solve it again. The iterative procedure will end if the returned RTE is schedulable, which is guaranteed to be an optimal solution of the original problem, or the problem is deemed to be infeasible (see Theorem 4 below).

The procedure is illustrated in Figure 1. It contains an initial Step 1 that checks if the most relaxed response time estimation range is schedulable. If yes, it enters the iterations between Step 2 (solving the relaxed problem $\mathbb{X}$) and Step 3 (computing MUTERs). We explain each step in details below.

Step 1. Let R_i^L and R_i^U denote the smallest and largest values for the response time of each task τ_i in any schedulable priority assignment. In this paper we take $R_i^L = C_i$ and $R_i^U = D_i$. The first step simply evaluates whether the response time estimation range $\mathcal{G}^M = \{\langle \tau_1, [C_1, D_1] \rangle, ... \langle \tau_n, [C_n, D_n] \rangle\}$ is schedulable. Any schedulable RTE $\mathcal{E}$ must satisfy $\mathcal{E} \in \mathcal{G}^M$. If $\mathcal{G}^M$ is not schedulable, then the system must be unschedulable by any priority assignment, as proven in Theorem 2.

Step 2. The second step searches for a response time estimation that has not been deemed unschedulable by the currently computed MUTERs. This is done by solving a relaxed problem $\mathbb{X}$ consisting of no schedulability conditions but the implied constraints by the computed MUTERs. Specifically, for each MUTER $\mathcal{U} = \{\langle \tau_1, [r_1^l, r_1^u] \rangle, ... \langle \tau_n, [r_n^l, r_n^u] \rangle\}$, by Theorem 2 any schedulable RTE $\mathcal{E} = \{\langle \tau_1, r_1 \rangle, ... \langle \tau_n, r_n \rangle\}$ cannot be contained in $\mathcal{U}$. This implies the following constraint

$$\mathcal{E} \notin \mathcal{U} \Leftrightarrow \neg \begin{cases} r_1^l \le r_1 \le r_1^u \\ ... \\ r_n^l \le r_n \le r_n^u \end{cases} \Leftrightarrow \left\| \begin{array}{l} r_1 < r_1^l \| r_1 > r_1^u \\ ... \\ r_n < r_n^l \| r_n > r_n^u \end{array} \right. \quad (23)$$

where $\neg$, $\{$, and $\|$ represent the logic-NOT, logic-AND, and logic-OR operations, respectively. Thus, $\mathbb{X}$ is essentially a mathematical programming problem consisting of the objective and the design constraints $\mathbf{h}(\cdot)$ in $\mathbb{O}$, while replacing the system schedulability constraints with those of (23) implied by all currently computed MUTERs. $\mathbb{X}$ also includes the response time estimations of all tasks $[r_1, ...r_n]$ as the additional design variables, as well as their initial bounding constraints $C_i \le r_i \le D_i, \forall i$. Formally, problem $\mathbb{X}$ can be expressed as

$$\begin{array}{rll} \mathbb{X}: & \min & g(\mathbf{X}) \\ & \text{s.t.} & \text{Implied constraints (23)}, \ \forall \, \mathcal{U} \in \mathbb{U} \\ & & C_i \le r_i \le D_i, \forall i \\ & & \mathbf{h}(\mathbf{X}) \le 0 \end{array} \quad (24)$$

where $\mathbb{U}$ is the set of currently known MUTERs.

In this paper, we assume $g(\cdot)$ and $\mathbf{h}(\cdot)$ are both linear functions of $\mathbf{X}$, hence $\mathbb{X}$ can be solved using MILP solvers such as CPLEX. Note that the logical disjunction constraint in (23) can be formulated in MILP using "big-M" method. For example,

$$\left\| \begin{array}{l} r_1 < r_1^l \\ r_1 > r_1^u \end{array} \right. \Leftrightarrow \begin{cases} r_1 < r_1^l + M \cdot b_1 \\ r_1 > r_1^u - M \cdot (1 - b_1) \end{cases} \quad (25)$$

Here b_1 is an auxiliary *binary* variable, defined as 0 if $r_1 < r_1^l$, 1 if $r_1 > r_1^u$. M is a sufficiently large constant such as D_1.

If $\mathbb{X}$ is infeasible (i.e., no solution satisfies all constraints in $\mathbb{X}$), then the system cannot be schedulable by any RTE (see Theorem 4 below). Otherwise, solving $\mathbb{X}$ will return an RTE (composed of the solution values assigned to the decision variables $[r_1, ...r_n]$) that respects all the implied constraints by the known MUTERs.

Step 3. This step evaluates the schedulability of the RTE $\mathcal{E}$ returned in Step 2, i.e., whether it satisfies Eq. (21). If yes, then $\mathcal{E}$ is optimal, and the associated priority assignment $\mathbf{P}$ is an optimal priority assignment (proven in Theorem 4). Here $\mathbf{P}$ can be obtained as a byproduct of applying Audsley's algorithm in checking $\mathcal{E}$ against the condition (21).

If $\mathcal{E}$ is unschedulable, it is generalized into at most k MUTERs using Algorithm 1, where k is a predefined parameter. The implied constraints from these newly discovered MUTERs are then added to the problem $\mathbb{X}$, and we enter the next iteration. In our experiments, we find that $k = 5$ is a good setting in most cases. Since problem $\mathbb{X}$ contains the constraints $C_i \le r_i \le D_i, \forall i$, the RTE $\mathcal{E} = \{\langle \tau_1, r_1 \rangle, ... \langle \tau_n, r_n \rangle\}$ from Step 2, which is the input to Algorithm 1 for computing MUTERs, must satisfy the prerequisite $r_i \in [C_i, D_i], \forall i$.

The following theorem formally proves the correctness of the proposed algorithm in Figure 1.

Theorem 4. The algorithm in Figure 1 guarantees to **terminate**. Upon termination, it reports **infeasibility/unschedulability** if the original problem $\mathbb{O}$ is infeasible, otherwise it returns an **optimal** priority assignment.

Proof. Termination. The number of MUTERs is bounded by $O(\prod_i D_i^2)$. Also, since at each iteration the algorithm in Figure 1 computes an RTE that respects the constraints imposed by all the previously found MUTERs, the newly computed MUTERs must be different. Hence, the algorithm shall always terminate as the number of iterations is finite.

We now prove that each of the three possible terminating conditions for the algorithm is correct.

Unschedulability. If at Step 1 it is deemed that $\mathcal{G}^M = \{\langle \tau_1, [C_1, D_1]\rangle, ... \langle \tau_n, [C_n, D_n]\rangle\}$ is unschedulable, there does not exist any schedulable priority assignment. This is because any RTE in $\mathcal{G}^M$ is unschedulable (by Theorem 2), there cannot exist any schedulable RTE. By Theorem 1 the system is unschedulable with any priority assignment.

Infeasibility. If at Step 2 it is deemed that problem $\mathbb{X}$ is infeasible, the original problem $\mathbb{O}$ must be infeasible too. This is because we can replace the system schedulability constraints $\mathbb{O}$ with the implied constraints of all MUTERs (Theorem 1), and $\mathbb{X}$ only contains those of a subset of all MUTERs. Hence $\mathbb{X}$ must be a relaxation of $\mathbb{O}$ (i.e., $\mathbb{X}$'s feasibility region is a superset of that of $\mathbb{O}$). In other words, an empty feasibility region for $\mathbb{X}$ means that $\mathbb{O}$ is also infeasible.

Optimality. If at Step 3 the RTE is deemed schedulable, then by Theorem 1, the corresponding returned priority assignment is guaranteed to be schedulable. As $\mathbb{X}$ is a relaxation of $\mathbb{O}$, the optimal solution of $\mathbb{X}$ that is schedulable must be optimal for $\mathbb{O}$ with a smaller feasibility region than $\mathbb{X}$. □

Although the algorithm in Figure 1 is guaranteed to terminate, in the worst case it may require to compute all MUTERs. Consequently it needs $O(\prod_i D_i^2)$ number of iterations between Steps 2 and 3, as each iteration computes a constant number of distinct MUTERs. Also, in each iteration it needs to solve an MILP problem $\mathbb{X}$. In the end, the algorithm still has an exponential worst case complexity, like those of the other exact algorithms based on BnB and a single MILP formulation.

We now demonstrate the algorithm by applying it to the problem of finding a schedulable priority assignment for the example system in Table I. The parameter k is set to 3.

Example 8. As the first step, we construct $\mathcal{G}^M$ as

$$\mathcal{G}^M = \{\langle \tau_1, [4, 10]\rangle, \langle \tau_2, [9, 35]\rangle, \langle \tau_3, [5, 120]\rangle, \langle \tau_4, [35, 180]\rangle\}$$

$\mathcal{G}^M$ is schedulable as it satisfies Eq. (21). The algorithm then enters an iterative procedure between Step 2 and Step 3.

Iteration 1. The relaxed problem $\mathbb{X}$ is constructed by leaving out all schedulability constraints. Hence, $\mathbb{X}$ only contains the constraints that the response time estimation is in the range defined by $\mathcal{G}^M$

$$\begin{array}{rlll} \mathbb{X}: & \min & 0 & \\ & \text{s.t.} & 4 \le r_1 \le 10 & \\ & & 9 \le r_2 \le 35 & \\ & & 5 \le r_3 \le 120 & \\ & & 35 \le r_4 \le 180 & \end{array} \tag{26}$$

Solving $\mathbb{X}$ returns the following response time estimation

$$\mathcal{E} = \{\langle \tau_1, 4\rangle, \langle \tau_2, 9\rangle, \langle \tau_3, 5\rangle, \langle \tau_4, 35\rangle\}$$

which is obviously unschedulable since the response time is estimated to be the WCET for each task. The following 3 MUTERs are computed

$$\mathcal{U}_1 = \{\langle \tau_1, [4, 10]\rangle, \langle \tau_2, [9, 35]\rangle, \langle \tau_3, [5, 120]\rangle, \langle \tau_4, [35, 136]\rangle\}$$
$$\mathcal{U}_2 = \{\langle \tau_1, [4, 10]\rangle, \langle \tau_2, [9, 25]\rangle, \langle \tau_3, [5, 25]\rangle, \langle \tau_4, [35, 180]\rangle\}$$
$$\mathcal{U}_3 = \{\langle \tau_1, [4, 8]\rangle, \langle \tau_2, [9, 35]\rangle, \langle \tau_3, [5, 8]\rangle, \langle \tau_4, [35, 180]\rangle\}$$

By (23), $\mathcal{U}_1$ implies the following constraints

$$\left\| \begin{array}{l} r_1 < 4 \parallel r_1 > 10 \\ r_2 < 9 \parallel r_2 > 35 \\ r_3 < 5 \parallel r_3 > 120 \\ r_4 < 35 \parallel r_4 > 136 \end{array} \right.$$

which can be simplified as follows considering that (26) shall be satisfied as well

$$r_4 \ge 137$$

Similarly, we can find the implied constraints for all three MUTERs. Below we give their simplified version, which is then added to $\mathbb{X}$

$$\begin{cases} \mathcal{U}_1: & r_4 \ge 137 \\ \mathcal{U}_2: & r_2 \ge 26 \parallel r_3 \ge 26 \\ \mathcal{U}_3: & r_1 \ge 8 \parallel r_3 \ge 8 \end{cases}$$

Iteration 2. Solving the updated problem $\mathbb{X}$ returns the solution below

$$\mathcal{E} = \{\langle \tau_1, 4\rangle, \langle \tau_2, 9\rangle, \langle \tau_3, 26\rangle, \langle \tau_4, 137\rangle\}$$

$\mathcal{E}$ is unschedulable, and the MUTER below is computed

$$\mathcal{U}_4 = \{\langle \tau_1, [4, 10]\rangle, \langle \tau_2, [9, 16]\rangle, \langle \tau_3, [5, 120]\rangle, \langle \tau_4, [35, 180]\rangle\}$$

The following implied constraint is added to problem $\mathbb{X}$

$$r_2 \ge 17$$

Iteration 3. Solving $\mathbb{X}$ returns the following solution

$$\mathcal{E} = \{\langle \tau_1, 4\rangle, \langle \tau_2, 26\rangle, \langle \tau_3, 26\rangle, \langle \tau_4, 137\rangle\}$$

$\mathcal{E}$ is still unschedulable. We compute the following MUTER

$$\mathcal{U}_5 = \{\langle \tau_1, [4, 10]\rangle, \langle \tau_2, [13, 35]\rangle, \langle \tau_3, [5, 120]\rangle, \langle \tau_4, [35, 149]\rangle\}$$

which implies the constraint below to be added to $\mathbb{X}$

$$r_2 \le 12 \parallel r_4 \ge 150$$

Iteration 4. Solving $\mathbb{X}$ now returns the following solution

$$\mathcal{E} = \{\langle \tau_1, 4\rangle, \langle \tau_2, 26\rangle, \langle \tau_3, 26\rangle, \langle \tau_4, 150\rangle\}$$

At this point, $\mathcal{E}$ becomes schedulable w.r.t. condition (21), and the associated schedulable priority assignment is

$$\pi_1 > \pi_3 > \pi_2 > \pi_4$$

VI. Experimental Evaluation

In this section, we present the experimental results using industrial case studies and synthetic systems. All MILP problems are solved using CPLEX 12.6.1.

A. *Mixed-criticality NoC*

Our first experiment considers NoC system with mixed-criticality traffic flows. We refer to [3] for the details of the system model and schedulability analysis. We compare three methods, all of which are optimal.

- **Enhanced-BnB**: An enhanced branch-and-bound procedure proposed in [18]. The branching order is optimized such that the likely schedulable priority assignments are explored first.
- **MILP**: A unified MILP formulation solved by CPLEX, as detailed in Appendix A.
- **MUTER-guided**: The proposed method in Section V.

We first evaluate them on a case study of autonomous vehicle applications [3]. The case study consists of 38 flows deployed on a 4×4 NoC, 6 of which are of HI-criticality. Each flow is characterized by the data size, the source and destination processors, and the criticality factor $(C_i(HI)/C_i(LO))$ if it is of HI-criticality. Since the utilization and WCET of the flows are not given, we generate a set of systems where we set $U(LO)$, the total system utilization at LO-criticality mode, to be a particular value. The utilization of each flow is then assigned proportional to the product of its data size and route length. Besides system schedulability, we consider the objective of maximizing the minimum laxity among all flows, as defined in Eq. (2). The time limit is set to 24 hours.

The results are summarized in Table III. As in the table, when the system utilization is relatively low such that the system is easily schedulable, **MUTER-guided** is over 1000× faster than both **Enhanced-BnB** and **MILP**. However, when the system utilization is around the borderline of being schedulable (as in the last two rows of the table), **MUTER-guided** requires many iterations to refine the schedulability region. It becomes about 10× slower than **MILP** (but still faster than **Enhanced-BnB**).

This motivates us to do a more systematic evaluation using randomly generated synthetic systems with different utilization levels and number of flows. We first try to identify the typical **borderline utilization** around which the problem is difficult to solve, using the following settings. In this experiment, we are only concerned to find a schedulable priority assignment. The system uses a 4×4 NoC platform. The periods of flows are selected from the interval [1, 1000] following the log-uniform distribution. The criticality factor of HI-criticality flows are set to be 2. We generate 1000 random systems for each value on the system utilization in LO-criticality mode $U(LO)$. The LO-criticality utilizations of individual flows are generated using the UUnifast algorithm. Each system contains 40 flows, 20 of which are of HI-criticality. The source, destination, and routing of each flow are generated following the stress test scheme in [13]. The time limit is set to 10 minutes to avoid excessive waiting. As in Table III, the methods typically require a much longer time if they cannot finish in 10 minutes.

Figure 2 shows the portion of systems that incur a timeout for each method, for systems with utilization within 50%-130%. For the purpose of finding the borderline utilization, this range is sufficient, since the systems with utilization at 50% are always easily schedulable, and those at 130% are mostly easily unschedulable. As indicated from the figure, the timeout ratio is different at different utilization, but the utilization around 80%–95% has the highest timeout ratio for

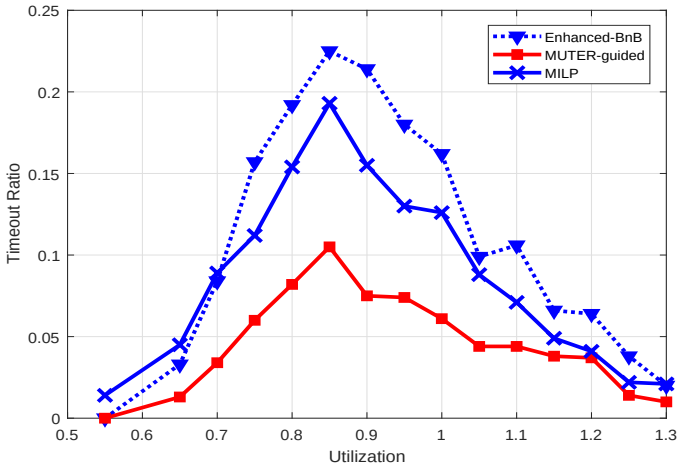

Fig. 2: Timeout ratio vs. LO-criticality Utilization (Time limit = 10 minutes).

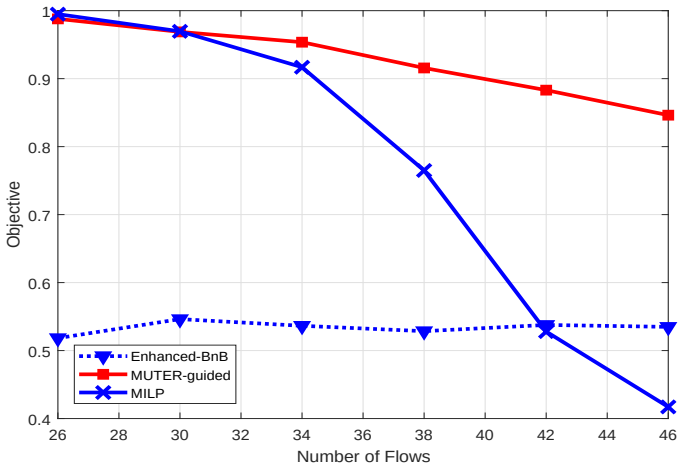

Fig. 3: Normalized objective vs. Number of flows (Time limit = 10 minutes).

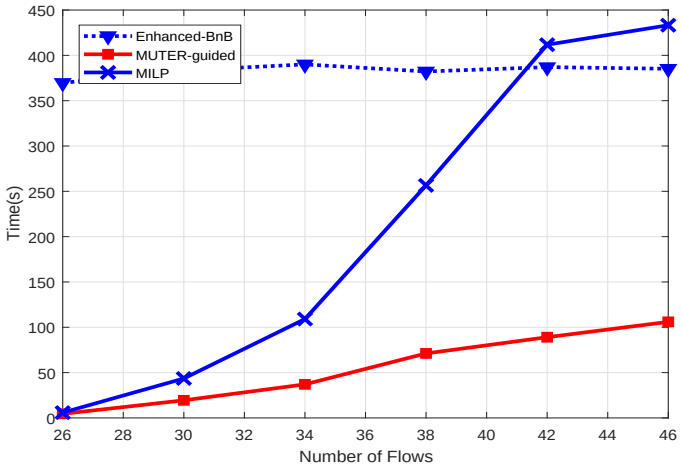

Fig. 4: Average runtime vs. Number of flows (Time limit = 10 minutes).

TABLE III: Results on autonomous vehicle applications ("N/A" denotes no solution found; Time limit = 24 hours)

$U(LO)$	**Enhanced-BnB**			**MUTER-guided**			**MILP**		
	Objective	Time	Status	Objective	Time	Status	Objective	Time	Status
1.357	40713	$\geq 24h$	Timeout	51309	1.7s	Terminate	51309	51743.02s	Terminate
1.404	40835	$\geq 24h$	Timeout	50318	2.21s	Terminate	50318	21789.89s	Terminate
1.451	5765	$\geq 24h$	Timeout	49328	35.54s	Terminate	49328	$\geq 24h$	Timeout
1.497	N/A	$\geq 24h$	Timeout	47204	49569.68s	Terminate	47204	6835.83s	Terminate
1.544	N/A	$\geq 24h$	Timeout	Infeasible	141.85s	Terminate	Infeasible	28.12s	Terminate

all three methods. For systems with other numbers of flows the "borderline utilization" is similarly around 80%–95%. Note that in this case study, there are totally 32 physical links shared by 38 traffic flows, and each flow may require multiple physical links to transmit. This means that unlike the case of software tasks running on a uniprocessor, the system may still be schedulable with a utilization higher than 100%. The difficulty of this borderline utilization can be intuitively explained as follows. The priority assignment problem is usually easy when the system utilization is either relatively low (e.g., around 50% in Figure 2) or relatively high (e.g., about 130% in Figure 2). In the former case, the system can easily be schedulable by a large number of priority assignments. In the latter, there are usually tasks that are obviously unschedulable however the priorities are assigned. In contrast, when the system utilization is around the borderline (i.e., away from the two extreme cases), the problem becomes relatively more difficult.

In the following, we perform another experiment where the system utilization in LO-criticality mode is chosen randomly from the interval [80%, 95%]. The number of flows now varies between 26–46. We add the objective back to maximize the minimum laxity, and the other settings follow that of Figure 2. We normalize the objective by dividing it by $L_i^u = \min_i(D_i - C_i(HI))$. Intuitively, L_i^u is an upper bound on the optimal solution. There are two scenarios that need special treatment. If the method times out without finding any feasible solution, we consider the corresponding normalized objective to be 0 as a penalty. Similarly, if the method is capable of proving infeasibility within the time limit, we set the normalized objective to 1. The time limit is 10 minutes.

Figure 3 and Figure 4 illustrate the average normalized objective value and average runtime of each method, respectively. **MUTER-guided** demonstrates much better optimization quality than both **Enhanced-BnB** and **MILP**, especially for large systems. In general, this is because **Enhanced-BnB** and **MILP** have higher timeout ratios and are often only able to find suboptimal solutions within the time limit. Hence, on average **MUTER-guided** has the promise to be more scalable than **Enhanced-BnB** and **MILP**.

B. Data Driven Activation in Distributed Systems

In this experiment, we evaluate the proposed method on an industrial experimental vehicle system with advanced active safety features [5]. The system consists of 29 Electronic Control Units (ECUs) connected through 4 CAN buses, 92 tasks, and 192 CAN messages. End-to-end deadlines are imposed on 12 pairs of source-sink tasks, which contain a total of 222 unique paths. The allocation of tasks and messages onto corresponding execution resources are given.

We compare the following three methods

TABLE IV: Maximizing min laxity for the vehicle system

Method	**MILP-Approx**	**MUTER-Approx**	**MUTER-Accurate**
Objective	90466	90466	90466
Runtime (s)	596.72	4.68	9.00

- **MILP-Approx**: Formulating the problem as a single MILP and solving it with CPLEX, where the response time analysis is approximate [26].
- **MUTER-Accurate**: The proposed technique using the accurate response time analysis.
- **MUTER-Approx**: Same as **MUTER-Accurate** but using the approximation analysis.

The accurate analysis is too complex to be formulated in MILP, since it needs to check all the instances in the busy period, the length of which is unknown a priori. Hence, for tasks with preemptive scheduling, an approximate analysis is proposed [26] which enforces the deadline to be no larger than the period. This makes it safe to only ensure the first instance in the busy period is schedulable. For messages with non-preemptive scheduling, we enforces constrained deadline to allow to adopt a similar analysis from [9].

We first consider the objective of maximizing the minimum laxity among all paths. The laxity of an end-to-end path is computed as the difference between the end-to-end latency deadline and the actual end-to-end latency. Formally, the objective is $\max \min_p(D_p - L_p)$, where p represents a path and L_p is defined in (9).

The results of the experiment are summarized in Table IV. As in the table, all three methods are capable of finding the optimal solution (which happens to be the same with either the accurate analysis or the approximate analysis). However, the proposed MUTER based technique runs much faster than a single MILP formulation.

Next we study the problem of finding the breakdown utilization of each resource (ECU or bus), which is defined as the maximum value the utilization of the resource can be scaled to such that the system is still feasible (in terms of both schedulability of individual tasks/messages and end-to-end path deadlines) assuming the utilization of other resources remains the same. For **MUTER-Approx** and **MUTER-Accurate**, the breakdown utilization is computed by a binary search on top of them. The binary search stops when the upper-bound and lower-bound are within 3%. A time limit of 2 hours is set for each ECU/Bus to prevent excessive waiting.

We summarize the runtime and breakdown utilization in Table V. **MUTER-Accurate** and **MUTER-Approx** both can finish within the time limit for each ECU/Bus. For **MILP-Approx**, it occasionally fails to finish in two hours. As

TABLE V: Breakdown utilization for the vehicle system ("N/A" denotes no solution found; Time limit = 2 hours)

ECU /Bus	Breakdown utilization			Runtime (seconds)		
	MILP -Approx	MUTER -Approx	MUTER -Accurate	MILP -Approx	MUTER -Approx	MUTER -Accurate
Bus1	**0.032**	**0.922±0.015**	**0.999±0.015**	Timeout	260	55
Bus2	**0.598**	**0.938±0.015**	**1.000±0.015**	Timeout	24	55
Bus3	0.024	0.035±0.015	0.035±0.015	521	5.1	11
Bus4	N/A	0.074±0.015	0.074±0.015	Timeout	9.2	21
ECU1	1.000	1.000	1.000	372	4.6	9.0
ECU2	**0.055**	**0.058±0.015**	**0.656±0.015**	679	0.1	13
ECU3	1.000	1.000±0.015	1.000±0.015	360	4.6	9.0
ECU4	0.094	0.108±0.015	0.108±0.015	343	0.003	0.003
ECU5	1.000	1.000±0.015	1.000±0.015	415	4.6	9.0
ECU6	N/A	0.780±0.015	0.780±0.015	Timeout	9.4	18
ECU7	**0.769**	**0.771±0.015**	**0.786±0.015**	403	12	28
ECU8	1.000	1.000±0.015	1.000±0.015	375	4.6	9.0
ECU9	**0.289**	**0.287±0.015**	**0.901±0.015**	2047	0.6	2745
ECU10	1.000	1.000±0.015	1.000±0.015	378	4.6	9.0
ECU11	1.000	1.000±0.015	1.000±0.015	645	6.8	9.2
ECU12	1.000	1.000±0.015	1.000±0.015	485	4.6	9.0
ECU13	**0.044**	**0.049±0.015**	**0.789±0.015**	385	0.3	5089
ECU14	**N/A**	**1.000±0.015**	**1.000±0.015**	Timeout	28	9.0
ECU15	1.000	1.000±0.015	1.000±0.015	480	4.6	9.0
ECU16	**N/A**	**0.524±0.015**	**1.000±0.015**	Timeout	4.8	51
ECU17	1.000	1.000±0.015	1.000±0.015	484	4.6	9.0
ECU18	N/A	0.798	0.798	Timeout	25	36
ECU19	1.000	1.000±0.015	1.000±0.015	480	4.6	9.0
ECU20	0.425	0.427±0.015	0.427±0.015	622	5.4	10
ECU21	**0.910**	**0.924±0.015**	**0.939±0.015**	715	19	45
ECU22	1.000	1.000±0.015	1.000±0.015	377	4.6	9.0
ECU23	1.000	1.000±0.015	1.000±0.015	368	4.6	9.0
ECU24	1.000	1.000±0.015	1.000±0.015	375	4.6	9.0
ECU25	0.128	0.139±0.015	0.139±0.015	402	0.2	0.1
ECU26	1.000	1.000±0.015	1.000±0.015	374	4.6	9.0
ECU27	1.000	1.000±0.015	1.000±0.015	382	4.7	9.0
ECU28	1.000	1.000±0.015	1.000±0.015	377	4.6	9.0
ECU29	1.000	1.000±0.015	1.000±0.015	373	4.6	9.0

highlighted in red bold fonts in the table, **MUTER-Accurate** is capable of tolerating noticeably higher utilization than the other two techniques for a number of resources, due to the accurate response time analysis. This demonstrates that the approximate analysis may lead to substantially suboptimal solutions. The MUTER-guided framework is mostly about two magnitudes faster than MILP. Furthermore, unlike MILP, it is more flexible in terms of the adopted schedulability analysis, as it uses a generic procedure to check the system schedulability.

VII. Conclusion

In this paper, we study the optimization of real-time systems where the response time of a task not only depends on the set of higher/lower priority tasks but also on the response times of other tasks. We introduce a set of concepts, including the response time estimation range that replaces the actual response time of each task for checking the system schedulability. We develop an optimization framework which builds upon such concepts and leverages Audsley's algorithm to generalize from an unschedulable solution to many similar ones. Experimental results on two example systems show that the proposed technique is potentially much faster than exhaustive search algorithms based on BnB or MILP.

References

[1] N. Audsley. On priority assignment in fixed priority scheduling. *Information Processing Letters*, 79(1):39 – 44, 2001.

[2] I. Bate and P. Emberson. Incorporating scenarios and heuristics to improve flexibility in real-time embedded systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, 2006.

[3] A. Burns, J. Harbin, and L. Indrusiak. A wormhole noc protocol for mixed criticality systems. In *IEEE Real-Time Systems Symposium*, 2014.

[4] Y. Chu and A. Burns. Flexible hard real-time scheduling for deliberative ai systems. *Real-Time Systems*, 40(3):241–263, 2008.

[5] A. Davare, Q. Zhu, M. Di Natale, C. Pinello, S. Kanajan, and A. S. Vincentelli. Period optimization for hard real-time distributed automotive systems. In *Design Automation Conference*, 2007.

[6] R. Davis and A. Burns. Robust priority assignment for fixed priority real-time systems. In *IEEE Real-Time Systems Symposium*, 2007.

[7] R. Davis and A. Burns. Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. In *IEEE Real-Time Systems Symposium*, 2009.

[8] R. Davis and A. Burns. On Optimal Priority Assignment for Response Time Analysis of Global Fixed Priority Pre-emptive Scheduling in Multiprocessor Hard Real-Time Systems. *Technical report YCS-2010-451, Department of Computer Science, University of York*, 2010.

[9] R. Davis, A. Burns, R. Bril, and J. Lukkien. Controller area network (can) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*, 35(3):239–272, 2007.

[10] R. Davis, L. Cucu-Grosjean, M. Bertogna, and A. Burns. A review of priority assignment in real-time systems. *J. Syst. Archit.*, 65(C):64–82, Apr. 2016.

[11] M. Di Natale, W. Zheng, C. Pinello, P. Giusto, and A. Sangiovanni-Vincentelli. Optimizing end-to-end latencies by adaptation of the activation events in distributed automotive systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, 2007.

[12] A. Hamann, M. Jersak, K. Richter, and R. Ernst. Design space exploration and system optimization with symta/s - symbolic timing analysis for systems. In *IEEE Real-Time Systems Symposium*, 2004.

[13] L. Indrusiak, J. Harbin, and A. Burns. Average and worst-case latency improvements in mixed-criticality wormhole networks-on-chip. In *Euromicro Conference on Real-Time Systems*, 2015.

[14] J. Y.-T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance evaluation*, 2(4):237–250, 1982.

[15] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, Jan. 1973.

[16] B. Nikolić and L. Pinho. Optimal minimal routing and priority assignment for priority-preemptive real-time NoCs. *Real-Time Systems*, 53(4):578–612, March 2017.

[17] M. Saksena and Y. Wang. Scalable real-time system design using preemption thresholds. In *IEEE Real-Time Systems Symposium*, 2000.

[18] Z. Shi and A. Burns. Priority assignment for real-time wormhole communication in on-chip networks. In *IEEE Real-Time Systems Symposium*, 2008.

[19] K. Tindell, A. Burns, and A. Wellings. Allocating hard real-time tasks: An np-hard problem made easy. *Real-Time Syst.*, 4(2):145–165, 1992.

[20] C. Wang, Z. Gu, and H. Zeng. Global fixed priority scheduling with preemption threshold: Schedulability analysis and stack size minimization. *IEEE Trans. Parallel and Distributed Systems*, 27(11):3242–3255, Nov. 2016.

[21] Y. Wang and M. Saksena. Scheduling fixed-priority tasks with preemption threshold. In *International Conference on Real-Time Computing Systems and Applications*, 1999.

[22] H. Zeng, M. Di Natale, and Q. Zhu. Minimizing stack and communication memory usage in real-time embedded applications. *ACM Trans. Embed. Comput. Syst.*, 13(5s):1–25, July 2014.

[23] Y. Zhao and H. Zeng. The concept of unschedulability core for optimizing priority assignment in real-time systems. In *Conference on Design, Automation and Test in Europe*, 2017.

[24] Y. Zhao and H. Zeng. The virtual deadline based optimization algorithm for priority assignment in fixed-priority scheduling. In *IEEE Real-Time Systems Symposium*, 2017.

[25] Y. Zhao and H. Zeng. The Concept of Response Time Estimation Range for Optimizing Systems Scheduled with Fixed Priority. *Technical report, Department of Electrical and Computer Engineering, Virginia Tech*, 2018. https://github.com/zyecheng/MUTER/blob/master/ZhaoReport2018.pdf

[26] W. Zheng, M. Di Natale, C. Pinello, P. Giusto, and A. S. Vincentelli. Synthesis of task and message activation models in real-time distributed automotive systems. *Conf. Design, Automation & Test in Europe*, 2007.

[27] Q. Zhu, H. Zeng, W. Zheng, M. D. Natale, and A. Sangiovanni-Vincentelli. Optimization of task allocation and priority assignment in hard real-time distributed systems. *ACM Trans. Embed. Comput. Syst.*, 11(4):1–30, Jan. 2013.

APPENDIX

A. Response Time Analysis and MILP formulation for Mixed-Criticality NoC

We give a brief overview of the schedulability analysis for wormhole communication in NoC with mixed-criticality network flows [3] as well as the unified MILP formulation used in our experiments of Section VI-A. For a HI-criticality flow τ_i, the WCET now is characterized with a pair of values $C_i(LO)$ and $C_i(HI)$. Likewise, its period is described with two values $T_i(LO)$ and $T_i(HI)$. The other attributes, including J_i^A and D_i, are not criticality-dependent.

For a LO-criticality flow, its response time in LO-criticality mode follows that of (7), where each attribute uses the value in LO-criticality mode

$$R_i(LO) = C_i(LO) + \sum_{\forall \tau_j \in shp_i} \left\lceil \frac{R_i(LO) + J_j^A + R_j(LO) - C_j(LO)}{T_j(LO)} \right\rceil C_j(LO) \tag{27}$$

Consider formulating (27) as an MILP constraint. The key is to properly express shp_i and the nonlinear ceiling term mathematically. We introduce a set of real variables $R_i(LO)$ for representing the LO-criticality response time of flow τ_i, a set of integer variables $I_{i,j}$ for representing the value of $\left\lceil \frac{R_i(LO)+J_j^A+R_j(LO)-C_j(LO)}{T_j(LO)} \right\rceil$, and a set of binary variables $p_{i,j}$ for representing the relative priority order between τ_i and τ_j. $p_{i,j}$ is defined as

$$\forall \tau_i \neq \tau_j, \;\; p_{i,j} = \begin{cases} 1, \; \tau_j \text{ has higher priority than } \tau_i \\ 0, \text{ otherwise} \end{cases} \tag{28}$$

The priority order variables should satisfy the antisymmetry and transitivity properties

$$\begin{aligned} \text{antisymmetry:} \quad & \forall \tau_i \neq \tau_j, && p_{i,j} + p_{j,i} = 1 \\ \text{transitivity:} \quad & \forall \tau_i \neq \tau_j \neq \tau_k, && p_{i,j} + p_{j,k} \leq p_{i,k} + 1 \end{aligned} \tag{29}$$

Specifically, the first constraint in (29) says that if τ_j has a higher priority than τ_i ($p_{i,j} = 1$), then τ_i must have a lower priority than τ_j ($p_{j,i} = 0$). The second constraint enforces that if τ_i has a lower priority than τ_j ($p_{i,j} = 1$), and τ_j has a lower priority than τ_k ($p_{j,k} = 1$), then τ_i must have a lower priority than τ_k ($p_{i,k} = 1$).

The following constraints guarantee that the value of $I_{i,j}$ is safely estimated (i.e., no smaller than the ceiling of $\left\lceil \frac{R_i(LO)+J_j^A+R_j(LO)-C_j(LO)}{T_j(LO)} \right\rceil$)

$$\forall \tau_i \neq \tau_j, \;\; I_{i,j} \geq \frac{R_i(LO) + J_i^A + R_j(LO) - C_j(LO)}{T_j} \tag{30}$$

Since $I_{i,j}$ is an integer variable, the above constraint naturally compute $I_{i,j}$ to be at least the ceiling of the right-hand side.

With variable $I_{i,j}$, the response time can be computed as

$$\forall \tau_i, \;\; R_i = C_i(LO) + \sum_{\forall \tau_j \in s_i} p_{i,j} I_{i,j} C_j(LO) \tag{31}$$

where s_i represents the set of flows that share a link with τ_j. Note that the above formulation involves $p_{i,j}I_{i,j}$, the product of a binary variable $p_{i,j}$ and another variable $I_{i,j}$. This can be linearized using the "big-M" method. Specifically, a new variable $\Pi_{i,j}$ is introduced to represent $p_{i,j}I_{i,j}$, which should satisfy the following constraints

$$\begin{aligned} &\forall \tau_i \neq \tau_j, \; \Pi_{i,j} \geq I_{i,j} - (1 - p_{i,j})M \\ &\forall \tau_i \neq \tau_j, \; \Pi_{i,j} \leq I_{i,j} \end{aligned} \tag{32}$$

where M is a sufficiently large constant such as D_i, and the first constraint is only effective when $p_{i,j} = 1$ (otherwise it is trivially true). Now (31) can be rewritten as

$$\forall \tau_i, \;\; R_i(LO) = C_i(LO) + \sum_{\forall \tau_j \in s_i} \Pi_{i,j} C_j(LO) \tag{33}$$

For a HI-criticality flow τ_i, additionally we need to analyze its HI-criticality response time, which involves the following three different cases. The first considers all HI-criticality flows to undergo simultaneous mode change. Each HI-criticality flow τ_i thus suffers interference from all higher priority HI-criticality flows with a shared link. The response time of τ_i in this case is calculated as

$$R_i^a(HI) = C_i(HI) + \sum_{\tau_j \in shpH_i} \left\lceil \frac{R_i^a(HI) + J_j^A + R_j(HI) - C_j(HI)}{T_j(HI)} \right\rceil C_j(HI) \tag{34}$$

where $shpH_i$ denotes the set of higher priority HI-criticality flows that share any link with τ_i.

Similar to the case of LO-criticality, the above response time can be formulated as the following MILP constraints. First, additional variables are introduced to represent the response time $R_i^a(HI)$, the number of interferences $I_{i,j}^a$, and the product $\Pi_{i,j}^a$ of $I_{i,j}^a$ and $p_{i,j}$.

The $I_{i,j}^a$ variables are subject to the following constraints

$$\begin{aligned} &\forall \tau_i, \tau_j \in HI, \tau_i \neq \tau_j, \\ &I_{i,j}^a \geq \frac{R_i^a(HI) + J_j^A + R_j(HI) - C_j(HI)}{T_j(HI)} \end{aligned} \tag{35}$$

The $\Pi_{i,j}^a$ and R_i^a variables are subject to the following constraints

$$\begin{aligned} &\forall \tau_i, \tau_j \in HI, \tau_i \neq \tau_j, \; \Pi_{i,j}^a \geq I_{i,j}^a - (1 - p_{i,j}) \cdot M \\ &\forall \tau_i, \tau_j \in HI, \tau_i \neq \tau_j, \; \Pi_{i,j}^a \leq I_{i,j}^a \\ &\forall \tau_i \in HI, \; R_i^a(HI) = C_i(HI) + \sum_{\tau_j \in sH_i} \Pi_{i,j}^a C_j(HI) \end{aligned} \tag{36}$$

where sH_i is the set of HI-criticality flows that share a link with τ_i.

The second case considers a LO-criticality flow that suffers increased interference due to increased indirect interference from other HI-criticality flows. The response time of τ_i in this case is calculated as

$$R_i^b(HI) = C_i(LO) + \sum_{\tau_j \in shp_i} \left\lceil \frac{R_i^b(HI) + J_j^A + R_j(HI) - C_j(HI)}{T_j(LO)} \right\rceil C_j(LO) \tag{37}$$

The MILP formulation is largely similar to that of $R_i^a(HI)$. The $I_{i,j}^b$ variables are used to model the ceiling terms, and are subject to the following constraints

$$\begin{aligned} &\forall \tau_i, \tau_j \in HI, \tau_i \neq \tau_j, \\ &I_{i,j}^b \geq \frac{R_i^b(HI) + J_j^A + R_j(HI) - C_j(HI)}{T_j(LO)} \end{aligned} \tag{38}$$

The $\Pi_{i,j}^b = p_{i,j} I_{i,j}^b$ and R_i^b variables are subject to the following constraints

$$\begin{aligned} &\forall \tau_i, \tau_j \in HI, \tau_i \neq \tau_j,\ \Pi_{i,j}^b \geq I_{i,j}^b - (1 - p_{i,j}) \cdot M \\ &\forall \tau_i, \tau_j \in HI, \tau_i \neq \tau_j,\ \Pi_{i,j}^b \leq I_{i,j}^b \\ &\forall \tau_i \in HI,\ R_i^b(HI) = C_i(HI) + \sum_{\tau_j \in s_i} \Pi_{i,j}^b C_j(HI) \end{aligned} \tag{39}$$

The third case is that a LO-criticality flow suffers increased direct interference from HI-criticality flows as well as interference from LO-criticality flows. The response time is

$$\begin{aligned} R_i^c(HI) = C_i(LO) + \\ \sum_{\tau_j \in shpH_i} \left\lceil \frac{R_i^c(HI) + J_j^A + R_j(HI) - C_j(HI)}{T_j(HI)} \right\rceil C_j(HI) + \\ \sum_{\tau_j \in shpUL_i} \left\lceil \frac{R_i^c(HI) + J_j^A + R_j(LO) - C_j(LO)}{T_j(LO)} \right\rceil C_j(LO) + \\ \sum_{\tau_j \in shpDL_i} \left\lceil \frac{R_i^b(HI) + J_j^A + R_j(LO) - C_j(LO)}{T_j(LO)} \right\rceil C_j(LO) \end{aligned} \tag{40}$$

where $shpUL_i$ is the set of higher priority LO-criticality flows that share a link with τ_i and may be upstream of the flow that caused the mode change, and $shpDL_i$ is the set of higher priority LO-criticality flows that share any link with τ_i and are downstream of the flow that caused the mode change.

The MILP formulation of the third case is more complicated as the equation contains 3 different ceiling terms.

$I_{i,j}^c$, $I_{i,j}^{c-UL}$ and $I_{i,j}^{c-DL}$, which model the three ceiling terms respectively, are subject to the following constraints

$$\forall \tau_i \neq \tau_j, \begin{cases} I_{i,j}^c \geq \dfrac{R_i^c(HI) + J_j^A + R_j(HI) - C_j(HI)}{T_j(HI)} \\ I_{i,j}^{c-UL} \geq \dfrac{R_i^c(HI) + J_j^A + R_j(LO) - C_j(LO)}{T_j(LO)} \\ I_{i,j}^{c-DL} \geq \dfrac{R_i^c(HI) + J_j^A + R_j(LO) - C_j(LO)}{T_j(LO)} \end{cases} \tag{41}$$

$\Pi_{i,j}^c$, $\Pi_{i,j}^{c-UL}$ and $\Pi_{i,j}^{c-DL}$ are used to model the product of $p_{i,j}$ and each of the above three variables respectively, and are subject to the following constraints

$$\forall \tau_i \neq \tau_j, \begin{cases} \Pi_{i,j}^c(HI) \geq I_{i,j}^c(HI) - (1 - p_{i,j}) \cdot M \\ \Pi_{i,j}^c(HI) \leq I_{i,j}^c(HI) \\ \Pi_{i,j}^{c-UL}(HI) \geq I_{i,j}^{c-UL}(HI) - (1 - p_{i,j}) \cdot M \\ \Pi_{i,j}^{c-UL}(HI) \leq I_{i,j}^{c-UL}(HI) \\ \Pi_{i,j}^{c-DL}(HI) \geq I_{i,j}^{c-DL}(HI) - (1 - p_{i,j}) \cdot M \\ \Pi_{i,j}^{c-DL}(HI) \leq I_{i,j}^{c-DL}(HI) \end{cases} \tag{42}$$

Finally the response time R_i^c is subject to the following constraint

$$\begin{aligned} \forall \tau_i,\ R_i^c(HI) = C_i(LO) + \\ \sum_{\tau_j \in sH_i} \Pi_{i,j}^c(HI) C_j(HI) \\ \sum_{\tau_j \in sUL_i} \Pi_{i,j}^{c-UL}(HI) C_j(LO) \\ \sum_{\tau_j \in sDL_i} \Pi_{i,j}^{c-DL}(HI) C_j(LO) \end{aligned} \tag{43}$$

where sUL_i is the set of LO-criticality flows that share a link with τ_i and may be upstream of the flow that caused the mode change, and sDL_i is the set of flows that share any link with τ_i and are downstream of the flow that caused the mode change.

Since $R_i^c(HI)$ is always no smaller than $R_i^b(HI)$ [3], the worst case HI-criticality response time of τ_i is

$$R_i(HI) = \max\{R_i^a(HI), R_i^c(HI)\} \tag{44}$$

Finally, the worst case response time of τ_i is

$$R_i = \max\{R_i(HI), R_i(LO)\} \tag{45}$$

The above two equations can be formulated as the following MILP constraints

$$\begin{aligned} &R_i \geq R_i^a(HI) \\ &R_i \geq R_i^c(HI) \\ &R_i \geq R_i(LO) \end{aligned} \tag{46}$$

The unified MILP formulation, taking into account the optimization objective of maximizing the minimum laxity L, is as follows

$$\begin{aligned} &\text{maximize } L \\ &\text{s.t. } \forall \tau_i,\ t \leq D_i - R_i \\ &(29), \\ &(30), (32), \\ &(35), (36), (33), \\ &(41), (42), (43) \\ &(46) \end{aligned} \tag{47}$$

The design variables include (a) the set of binary variables $p_{i,j}, \forall \tau_i \neq \tau_j$, (b) the sets of integer variables $I_{i,j}, I_{i,j}^a, I_{i,j}^c, I_{i,j}^{c-UL}, I_{i,j}^{c-DL}, \forall \tau_i \neq \tau_j$, (c) the sets of integer variables $\Pi_{i,j}, \Pi_{i,j}^a, \Pi_{i,j}^c, \Pi_{i,j}^{c-UL}, \Pi_{i,j}^{c-DL}, \forall \tau_i \neq \tau_j$, (d) the sets of real variables $R_i, R_i^a(HI), R_i^c(HI), R_i(LO), \forall \tau_i$, and (e) the real variable L.

ARTIFACT EVALUATION

A. Introduction

The submitted artifact contains the software implementation of the proposed technique and code and scripts for performing the experiments presented in the paper. All materials are packed in a VirtualBox virtual machine image as required. The virtual machine runs a 64-bit Ubuntu-14.04.1-server operating system and is currently configured to run with 8 processors and 4096MB RAM (adjustable). The experiment results presented in the paper however, were produced directly on a host machine with 8 cores and 8192MB of RAM. Thus there may exist some difference in run time. The software is written in C++ and uses third party library CPLEX for solving Mixed Integer Linear Programming problems. All required modules are provided in the virtual machine with no need to install any additional library and tool.

The artifact can be used to reproduce the results presented in table III, table IV, table V, figure 2 and figure 3, 4 respectively. The expected run time for obtaining each table/figure (assuming a single computer with 8 cores) is summarized in the following table.

TABLE VI: Expcted Run Time of Experiments

Experiment	Run Time	Experiment	Run Time
table III	≈5 days	figure 2	≈1 week
table IV	≈20 minutes	figure 3, 4	≈2 weeks
table V	≈1 day		

Due to the long total run time, we have designed the artifact in a way that allows to split the experiments into smaller parts and perform them independently. This would be useful for 1) performing several experiments simultaneously on multiple computers (nodes); 2) performing experiments selectively.

B. Virtual Machine Set Up

The virtual machine image can be downloaded from the following link https://www.dropbox.com/s/kfzbxjnvp574ijn/MUTER.ova?dl=0. After booting into the system, use the following account and password for login.

- account: zyecheng
- password: rtasmuter

The main directory **Artifact** contains the following four folders: **CPLEX**, **ExpSpace**, **Src**, **Paper**.

CPLEX folder contains the necessary CPLEX libraries and header files for compiling the executable. **ExpSpace** folder contains case studies files for performing experiments. **Src** folder contains all the implementation code in C++. **Paper** folder contains the submitted version of the paper.

C. Compilation

A pre-compiled executable has been provided in the **SRC** and **ExpSpace** folder. If re-compilation is necessary, first enter the **SRC** folder, which contains all the source files and makefile. The executable can be compiled using "make all" command. This will produce an executable named "MUTER.exe" in the same folder. We also recommend using the script "build.sh"(also under **Src** folder). The script compiles the executable and moves it to **ExpSpace** folder for performing experiments.

D. Experiment

In the following, we show how to reproduce the experiment results using the artifact. Note that the amount of time needed for performing different experiments may vary greatly (from several seconds to weeks). To make the process more efficient, we first introduce experiments that take smaller amount of time to complete. We assume that the current working directory is **ExpSpace** and contains the executable "MUTER.exe".

- **Table IV**

In this experiment, we apply three algorithms: **MILP-Approx**, **MUTER-Approx** and **MUTER-Accurate** for optimizing an industrial vehicle system case study. The algorithms are compared in terms of optimization objective and total run time. The experiment can be performed using the following command.

./MUTER.exe VehicleSystem_All

The command will run **MUTER-Approx**, **MUTER-Accurate** and **MILP-Approx** in turns on the case study and prints the results of objective and run time by each algorithm directly to the console. The process should take approximately 10 to 20 minutes.

- **Table V**

This experiment applies the same three algorithms as in the previous one for finding the breakdown utilization of each ECU/Bus in the vehicle system case study. The algorithms are compared in terms of the actual breakdown utilization found and total run time for each ECU/Bus. We first provide the following command for reproducing the results for a single ECU/Bus of a given algorithm.

./MUTER.exe VechicleSystem_BreakDownUtil
Algorithm ECU/Bus

Argument "Algorithm" specifies the algorithm to be used and can be one of **MILP-Approx**, **MUTER-Approx** and **MUTER-Accurate**. The argument "ECU/Bus" specifies the name of the ECU/Bus and can be any entry of the first column of Table V. For example, to find out the breakdown utilization of ECU1 using method **MUTER-Accurate**, the following command should be used.

./MUTER.exe VechicleSystem_BreakDownUtil
MUTER-Accurate ECU1

The command will print the actual breakdown utilization and runtime directly to the console.

We also provide a script name "BreakDownUtil.sh" in **ExpSpace** folder which automates the process of applying the above command for computing breakdown utilization for all ECU/Bus by all algorithms. The entire process requires approximately one day to complete.

- **Table III**

This experiment evaluates the performance of **Enhanced-BnB**, **MUTER-guided** and **MILP** on optimizing an autonomous vehicle application with 5 different total LO mode utilization configurations (specified in the $U(LO)$ column).

We first provide the following command that reproduces the results by **MUTER-guided** algorithm (column 5 to 7) for a given $U(LO)$ configuration.

./MUTER.exe NoCCaseStudy_MUTER-guided row

The last argument "row" specifies for which utilization configuration to perform the optimization and can be any integer from 1 to 5. For example, to reproduce the result corresponding to the first $U(LO)$ configuration (1.357), the following command should be used.

./MUTER.exe NoCCaseStudy_MUTER-guided 1 (48)

The objective, run time and status will be printed directly to the console upon completion. We also provide a script named "NoCCaseStudy-MUTER-guided.sh", which automates the process of running the command for all $U(LO)$ configurations. Based on the results we presented in the table, the total amount of time should be around 15 to 20 hours .

The following command reproduces the results by **MILP** algorithm (column 8 to 10) for a given $U(LO)$. It works in the same way as that for **MUTER-guided**.

./MUTER.exe NoCCaseStudy_MILP row

The script "NoCCaseStudy-MILP.sh" automates the process of applying the above command for all configurations of $U(LO)$. Based on the result we previous have in the table, the total amount of time should be around 2 to 3 days.

The following command reproduces the results by **Enhanced-BnB** algorithm (column 2 to 4).

./MUTER.exe NoCCaseStudy_Enhanced-BnB

Different from the previous two commands, the above command does not take a third argument specifying $U(LO)$ configuration. Instead, it makes use of the fact that **Enhanced-BnB** is a sequential algorithm and creates 5 worker threads that run all $U(LO)$ configurations in parallel. The run-time status of each method (currently found best objective, time elapsed) is periodically (every 300ms) collected by another dedicated thread and updated to a text file named "MCNoCAutonomousCaseStudy_EnhancedBnB_Result.txt", which will be created under the same folder once the program starts.

Since the experiment will take very long to complete (up to the pre-set time limit of 24 hours), it is recommended that the command be executed in the background. This also makes it easier to check the status text file once in a while for validation. One way of executing the command in background is to use the "nohup" command:

nohup ./MUTER.exe NoCCaseStudy_Enhanced-BnB &

- **Figure 2**

This experiment evaluates the timeout ratio (w.r.t a time limit of 600 seconds) versus different system utilizations by algorithms **Enhanced-BnB**, **MUTER-guided** and **MILP** respectively. For each utilization level, we randomly generate 1000 synthetic systems and apply the three algorithms for finding schedulable priority assignments. The timeout ratio of an algorithm is estimated as the percentage of timeout cases out of 1000. The random systems for producing the results presented in figure 2 are provided in the **USweep** folder and will be used as default. We first provide the following command for reproducing the result of a single data point in the figure given an algorithm and system utilization.

./MUTER.exe NoCUSweep Algorithm Utilization

Argument "Algorithm" specifies the algorithm and can be one of **Enhanced-BnB**, **MUTER-guided** and **MILP**. Argument "Utilization" specifies the utilization level and can be any real positive number. For example, the following command perform the experiment for measuring the timeout ratio for **MUTER-guided** algorithm at utilization level 0.80.

./MUTER.exe NoCUSweep MUTER-guided 0.80

The above command runs **MUTER-guided** algorithm on each of the 1000 random systems. Each time a system is completed, the program updates the total number of timeout cases and timeout ratio, which are then printed directly to the console as well as stored to a text file named NoCSyntheticUSweep_0.80_MUTER-guided.txt" in the **USweep** folder. The naming format is similar for other algorithms and utilization levels. The total amount of time for reproducing a single point varies from several minutes to days. System utilizations with higher timeout ratio generally require more time to finish.

We also provide a script named "NoCUSweep.sh", which automates the process of reproducing all the data points shown in figure 2 using the above command. The entire process takes at least a week on a single computer.

- **Figure 3 and 4**

This experiment evaluates the objective and run time of algorithms **Enhanced-BnB**, **MUTER-guided** and **MILP** versus different system sizes. For each system size, we generate 1000 random systems and apply the three algorithms to perform optimization. For each algorithm, the objective and run time are computed as the average of the optimization results on the 1000 random systems. The random systems for producing the results in figure 3 and 4 are provided in the **NSweep** folder and will be used as default. The following command reproduces the results given an algorithm and system size.

./MUTER.exe NoCNSweep Algorithm N

Argument "Algorithm" specifies the algorithm and can be one of **Enhanced-BnB**, **MUTER-guided** and **MILP**. Argument "N" specifies the system size. For example, to perform experiment that measures the average objective and run time of algorithm **MUTER-guided** for system size of 30, the following command should be used.

./MUTER.exe NoCNSweep MUTER-guided 30

The above command runs **MUTER-guided** algorithm on each of the 1000 random systems of size 30. Each time a system is completed, the command updates the average objective and run time, which are then printed directly to the console as well as stored to a text file under folder **NSweep** named "NoCSyntheticNSweep_N30_MUTER-guided.txt". The naming format is similar for other algorithms and systems sizes. The time for reproducing each data point varies between several hours and days. Generally, larger system size requires more time. A script named "NoCNSweep.sh" is provided that automates the process for reproducing all data points using the above command.