Online Reconfiguration of Regularity-based Resource Partitions in Cyber-Physical Systems

Wei-Ju Chen[†], Peng Wu[‡], Pei-Chi Huang[§], Aloysius K. Mok[†], Song Han[‡]
[†]Department of Computer Science, The University of Texas at Austin
{*albertwj, mok*}*@cs.utexas.edu*
[‡]Department of Computer Science and Engineering, University of Connecticut
{*peng.wu, song.han*}*@uconn.edu*
[§]Department of Computer Science, The University of Nebraska at Omaha
{*phuang*}*@unomaha.edu*

***Abstract*—We consider the problem of resource provisioning for real-time cyber-physical applications in an open system environment where there does not exist a global resource scheduler that has complete knowledge of the real-time performance requirements of each individual application that shares the resources with the other applications. Regularity-based Resource Partition (RRP) model is an effective strategy to hierarchically partition and assign various resource slices among the applications. However, RRP model does not consider changes in resource requests from the applications at run time. To allow for the run time adaptation to change resource requirements, we consider in this paper the issues in online resource partition reconfiguration, including semantics issues that arise in configuration transitions that may cause application failures. Based on the reconfiguration semantics, we study the online resource reconfigurability problem under the RRP model where the availability factors of resource partitions may be reconfigured during run time. We formalize the *Dynamic Partition Reconfiguration (DPR)* problem and provide a solution to this problem. Extensive experiments have been conducted to evaluate the performance of the proposed approach in different scenarios. We also present a case study using the autonomous F1/10 model car; the controller of the F1/10 car requires resource adaptation to satisfy the computing needs of its PID controller and vision system under different operating conditions. Our implementation demonstrates the effectiveness and benefit of online resource partition reconfiguration using the DPR approach in a real system.**

I. Introduction

A cyber-physical system (CPS) may consist of multiple applications that share resources from the same resource pool. In an open system environment [1], [2], there is no global scheduler that has full knowledge of the real-time performance requirements of each individual application. Each application tenders a request and is allocated a fraction of the shared resource to meet its own need. It is up to the application-level scheduler to schedule the tasks in each application to meet the task-level timing constraints. In the literature [3]–[7], the problem of resource allocation in the open system environment assumes the application resource requirements do not change during application execution. In many real applications, this assumption may not hold as the application may respond to changes in the operating environment in real time. In this paper, we introduce the *Dynamic Partition Reconfiguration (DPR)* problem that addresses the issues of dynamic resource reconfiguration under the Regularity-based Resource Partition (RRP) model [3]. Other works have investigated the dynamic reconfiguration problem but as far as we know there is no previous work that addresses the precise semantics of resource reconfiguration that may cause system instability issues in the open system environment. To illustrate this problem, consider an autonomous car control system which operates in two operational contexts: "Straight Ahead" and "Turn Corner". In the Straight Ahead context, the car runs straight along a corridor toward a corner while keeping itself in the middle of the corridor. In the Turn Corner context, the car makes a turn around the corner it has detected. The application computation requirement of the CPU is different in the two contexts[1]. If the resource allocation changes abruptly from one context to the next, then instability may occur that results in the car crashing into the side of the corridor. In this paper, we shall address the resource provisioning semantics during context changes. Explicit and precise semantics of system behavior during a context change will prevent unexpected results at run time. Based on the semantics, we shall present a three-stage algorithm to ensure the context changes will not incur system instability and discuss the associated scheduling results.

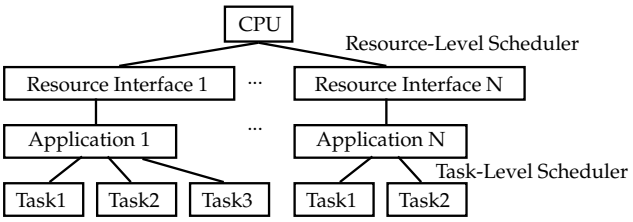

Fig. 1: Overview of the hierarchical scheduling model.

The RRP model is an abstraction of a component-based hierarchical scheduling system where each component is an application providing the functionality that is required by a CPS with real-time performance constraints [3], [8], [9]. For example, an autonomous car may have an application task for keeping the car in a traffic lane and another application task for detecting obstacles ahead. A component/application may

[1]The application demo can be found in the following link: http://www.youtube.com/watch?v=8b-MMP3-cug

2576-3172/19/$31.00 ©2019 IEEE
DOI 10.1109/RTSS46320.2019.00050

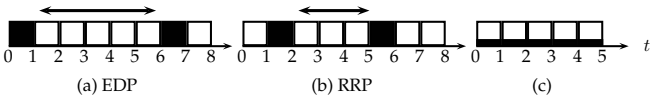

Fig. 2: (a) and (b): two possible schedules for a resource interface with a bandwidth of $\frac{1}{4}$. (c): resource supply from the application's point of view.

consist of several sub-components (sub-tasks). A parent component distributes its share of resource to its sub-components each of which in turn distributes it to its sub-components in a hierarchical fashion. Fig. 1 gives an overview of the hierarchical resource scheduling model by taking the CPU resource as an example. In this example, the CPU resource is distributed to N resource interfaces and each resource interface is utilized by an application. Given a resource interface, each application distributes its resource share to its task group according to self-defined policies. In past work on hierarchical scheduling systems, there is another popular approach to characterize the resource usage interface of each component besides our RRP model: the Periodic Resource Model (PRM) [5] (or its variant the Explicit Deadline Periodic (EDP) model [7]). The main difference between these two resource models is illustrated in Fig. 2 which shows the possible schedules of a resource allocation with a bandwidth assignment of $1/4$ of the resource. In the EDP model, there is an interval of length 5 from time 1 to 6 where the resource supply is zero. In contrast, the length of such a zero-supply interval can be limited by an interface which explicitly specifies the allowable resource supply jitter in the RRP model. Ideally, from the application's point of view, the resource should be supplied uniformly over any time interval as if it is dedicated to the application, but at a slower rate ($\frac{1}{4}$) as depicted in Fig. 2 (c). Taking the resource supply jitter into consideration, the resource supply specified by the resource interface under the RRP model better approximates the ideal supply which is uniform over any time interval. Hence, changes to the task group can be more easily accommodated by the application's own task scheduler by rescheduling tasks within its allocated resource partition. This is possible as long as the application's task utilization remains below the assigned availability factor, thus avoiding the need to change the resource interface [3], [10].

Although the resource interface can be used to adapt to resource requirement changes in the RRP model, the reconfiguration may be performed at run time, and the real-time performance guarantee during such reconfiguration is, however, not well studied in the literature. In particular, there may exist temporary utilization overload or performance degradation during the reconfiguration. In this paper, we study the online dynamic resource reconfigurability problem in the RRP model. We first discuss the key challenges to address this problem, and then propose the performance semantics for resource partitions during the reconfiguration by introducing the concept of reconfiguration supply regularity. To handle each reconfiguration request, a three-stage algorithm is proposed to construct both the transition schedule during the reconfiguration and the new schedule after the reconfiguration. Extensive simulation-based experiments have been conducted to evaluate the performance of the proposed approach in different scenarios. A case study will be presented on a real-life autonomous car control system which requires dynamic resource reconfiguration. This application demonstrates the necessity for online resource reconfigurability to prevent system instability and the effectiveness of our approach.

In the rest of this paper, Section II summarizes the related work, and Section III reviews the RRP model. Section IV describes the main challenge of online resource partition reconfiguration, defines the semantics of performance guarantee during reconfiguration and gives the precise definition of the corresponding scheduling problem. The detail of our three-stage algorithm is provided in Section IV. The performance evaluation results and a real-life case study are presented in Section V. Section VI concludes this work.

II. Related Work

There are several related research areas on scheduling tasks with varying timing requirements such as mixed-criticality systems [11], multi-mode systems [12], [13] or both [14]–[19]. In such systems, the task parameters may change, or new/old tasks may be added/removed to/from the system depending on the current state of the system. Systems with mode changes require the design of new protocols such as [13], [20]–[24] to ensure that the mode switch is performed in a timely and safe manner in response to both internally or externally generated events. The key challenge in these protocol designs is how to ensure the schedulability of the system not only in each mode but also during the mode transition. In this paper, we focus on online resource interface reconfiguration instead of designing a new task-level mode change protocol. More specifically, a resource partition is characterized by its resource availability factor and its supply jitter bound, whereas a task is often specified by its execution time, period and deadline. The semantics of performance guarantee during the resource partition reconfiguration is also different from that of the mode switch. The existing task-level mode change protocols cannot be directly applied to resource interface reconfiguration under the RRP model. The resource interface to be studied in this paper is assigned to a group of tasks which may change their mode at runtime.

There have been some research work on the multi-mode resource interface [13], [25]–[27] where the resource interface may change. For instance, Evripidou and Burns [13] used a two-level scheduler or a hyper-visor to handle the criticality mode change. Phan et al. [25] proposed a compositional analysis of the multi-mode resource interface. Li et al. [26] used virtual machine (VM) to support multi-mode virtualization where the parameters of VM change with minimum transition latency. Nikolov et al. [27] presented a solution from task-level to application-level scheduling and mode optimization. However, none of those work studies 1) the precise performance semantics of resource interfaces during reconfiguration where performance degradation may happen and 2) how to

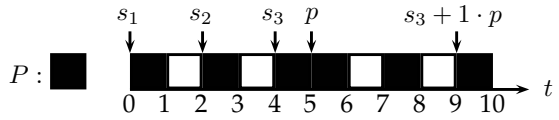

Fig. 3: Physical resource Π is allocated to resource partition P in a collection of resource slices.

schedule a set of resource partitions to meet the performance requirements.

III. RRP MODEL

This section revisits the regularity-based resource partition (RRP) model which is the foundation for the online resource partition reconfiguration problem to be elaborated in Section IV. In this paper, we assume the algorithms and models are applied to uniform resources where the size of resource slices is the same.

A. Regularity-based Resource Partition

We assume that a physical resource Π is allocated non-preemptively to one task at a time for some minimum time interval. Each of these minimum time intervals is called a resource slice and the physical resource is allocated in units of resource slices, as shown in Fig. 3. Each resource slice starts and ends at integral time boundaries. In this paper, we refer time to be these boundaries as t depicted in Fig. 3.

Definition III.1: A resource partition P on a physical resource Π is a tuple $(\mathcal{S}, p)$, where $\mathcal{S} = \{s_1, s_2, \cdots, s_n : 0 \leq s_1 < s_2 < \cdots < s_n < p\}$ is a set of n time points that denote the start times of the resource slices (called the offsets) allocated to the partition, and p is the partition period with the following semantics. The physical resource Π is available to the application tasks to which the partition P is allocated only during the time intervals $[s_k + x \cdot p,\ s_k + 1 + x \cdot p)$, $x \in \mathbb{N}, 1 \leq k \leq n$.

Definition III.2: The supply function $S(t)$ of resource partition P is the number of allocated resource slices in interval $[0, t)$.

$S(t)$ represents the amount of resource supply for resource partition P from time 0 to t. For example in Fig. 3, the resource partition is $P = (\{s_1 = 0, s_2 = 2, s_3 = 4\}, 5)$ and the supply function of P has $S(1) = 1, S(2) = 1, S(3) = 2, S(4) = 2$, and so on.

Definition III.3: The availability factor α of a resource partition $P = (\mathcal{S}, p)$ is defined as $\frac{|\mathcal{S}|}{p}$ where $|\mathcal{S}|$ is the number of elements in $\mathcal{S}$.

The RRP model characterizes the resource supply in two dimensions: (1) the resource supply rate and (2) the deviation of the resource supply from the ideal resource supply which allocates the resource evenly to the application over any time interval (*zero jitter*). The resource supply rate is defined as the *availability factor* α, and we introduce the concept of *regularity* to capture the jitter in the resource supply. In the ideal case, for any time interval of length l, the ideal resource supply to the application should be equal to $l \cdot \alpha$. The maximum supply deviation from this amount over any time interval is defined as the supply regularity.

Definition III.4: The instant regularity $I(t)$ for a resource partition P at time t is defined as $I(t) = S(t) - \alpha \cdot t$.

Definition III.5: Let a, b, k be non-negative integers. The supply regularity R of resource partition P is defined as the smallest k such that $|I(b) - I(a)| < k, \forall b \geq a$.

Definition III.6: A regular partition is a resource partition with supply regularity of 1 and an irregular partition is a resource partition with supply regularity larger than 1.

For example in Fig. 3, the availability factor α of resource partition P is $\frac{3}{5}$. The instant regularity $I(t)$ has $I(1) = \frac{2}{5}, I(2) = -\frac{1}{5}, I(3) = \frac{1}{5}$ and so on. The supply regularity R is 1 and P is regular.

B. RRP Scheduling Algorithms

Several algorithms have been developed to construct the schedule of regular and irregular resource partitions. For the single uniform resource case, the Adjusted Availability Factor (AAF) algorithm allocates resource partitions with availability factor of power of $\frac{1}{2}$ to each application [3]. By limiting the choice of availability factor, the schedule can be easily constructed if the sum of the availability factors is less than 1. This however introduces some resource utilization overhead. For the uniform multi-resource environment, the use of a combination of Magic7, PFair algorithms, and various forms of availability factors was proposed to construct the runtime schedule and greatly improves the resource utilization overhead [9], [28]. For the non-uniform multi-resource environment, the Acyclic Regular Composite Resource Partition Scheduling algorithm was proposed to schedule acyclic regular composite resource partitions where a composite resource partition is a collection of multiple resource partitions [4]. All these algorithms were designed for static resource partition construction.

IV. RESOURCE RECONFIGURABILITY IN RRP MODEL

Different from the aforementioned work on the RRP model, this paper studies the resource partition reconfigurability problem in a dynamic environment. In Section IV-A, we first present the main challenges of maintaining regularity-based resource partition in the run time and then define the performance semantics for online resource partition reconfiguration in Section IV-B. We give the formal scheduling problem formulation in Section IV-C and finally present a novel three-stage algorithm in Section IV-D for constructing the resource partitions during and after the reconfiguration.

A. Challenges

In the RRP model, there may be multiple applications running on any physical resource and each application may request to reconfigure its resource partitions on demand. An application can issue a *Reconfiguration Request of Resource Partition* (R^3P) to request new resource partitions or reconfigure the existing ones. As illustrated in Fig. 4, the application can request to reconfigure its resource supply curve by issuing an R^3P. The system then enters the *Resource Partition Transition (RPT)* stage where resource partitions are reconfigured and performance degradation or temporary over supply may

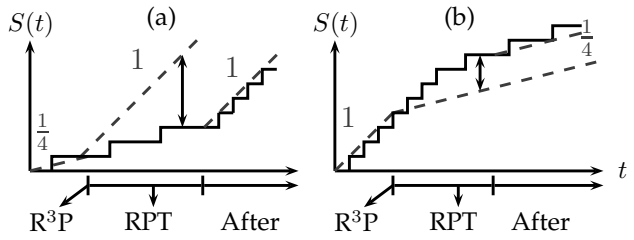

Fig. 4: The dotted line illustrates that the requested availability factor changes from $\frac{1}{4}$ and 1 to 1 and $\frac{1}{4}$ in (a), (b), respectively, at the time of R^{3}P. The arrow shows the supply deviation during the reconfiguration where there is performance degradation in (a) and resource over supply in (b), respectively.

happen as shown in Fig. 4 (a) and (b), respectively. After the RPT stage is over, the reconfigured resource partitions will supply the resource to applications in accordance with the new availability factor and new supply regularity by approximating the new ideal supply curve in a staircase function as depicted in Fig. 4 (a) (lower dotted supply curve) and (b) (upper dotted supply curve).

There are multiple challenges to handling R^{3}P appropriately. First of all, during the RPT stage, there could exist a temporary overload or schedule conflict such that the system cannot reconfigure the availability factors of some resource partitions. This will inevitably violate the performance guarantee of some resource partitions that may result in unexpected application failures. To address this issue, a formal definition of the performance semantics during the resource partition reconfiguration is needed. Secondly, even if a temporary overload does not happen during the reconfiguration, resource provisioning may suffer a serious performance degradation if we naively reschedule the resource without considering the current resource supply state of individual resource partitions. This unexpected performance degradation may cause an application utilizing this partition to miss a deadline during the transition. An example is shown in Fig. 5 (a) where a new regular resource partition P_3 joins the system at time 5. To fulfill the performance requirement of P_3, there are only two options to schedule P_3's partitions, as depicted in Fig. 5 (b) and (c), respectively. Unfortunately, P_3 will conflict with P_2 in (b) and conflict with P_1 in (c). In these two cases, even though the total utilization does not exceed 1, the system still cannot schedule the three resource partitions owning to the conflict. One can naively reschedule the resource to accommodate this change such as using the AAF or Magic7 algorithm, as described in Section III-B to compute a completely new schedule and switch to this schedule at time 5 as depicted in Fig. 5 (d). However, naively rescheduling resource may cause some resource partitions to suffer serious performance degradation and the violation of supply regularity. As illustrated in Fig.5 (d), P_2 will suffer a serious starvation interval during time 2 to 8.

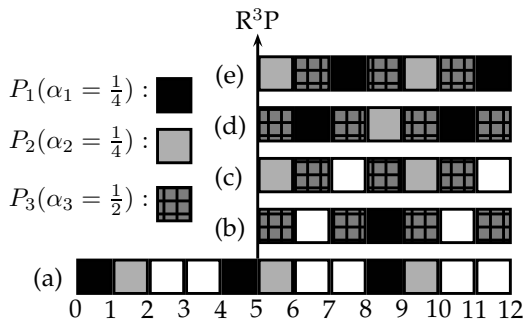

Fig. 5: There is a R^3P at time 5 requesting to add a new resource partition P_3. (a) shows the schedule without R^3P. (b) and (c) shows the R^3P will inevitably cause P_3 to conflict with either P_1 or P_2. (d) shows a naive rescheduling approach that results in a serious performance degradation in P_2 during time 2 to 8. (e) shows a schedule such that P_1, P_2 and P_3 do not suffer performance degradation and they are all reconfiguration regular.

B. RRP Model Extension for Online Reconfiguration

To address the aforementioned challenges, we now extend the RRP model to take the online reconfiguration of resource partitions into consideration and define the semantics of performance guarantee during the reconfiguration. The precise scheduling problem formulation and key ideas of the proposed algorithm will be presented in the next section.

Definition IV.1: Reconfiguration Request of Resource Partition (R^{3}P) is defined as a tuple $\lambda = \{\mathcal{P}^t, \mathcal{R}^r, T\}$ where $\mathcal{P}^t$ is the target set of resource partitions after the request; each resource partition $P_i \in \mathcal{P}^t$ has an associated reconfiguration supply regularity (see Def. IV.3) of $R_i^r \in \mathcal{R}^r$; T is the maximum time allowed for the reconfiguration to complete.

Recall that a resource partition P is a tuple $(\mathcal{S}, p)$. The value of the tuple changes as the resource partition is reconfigured at run time. We denote the value of the tuple of a resource partition P as P^o, P^d and P^t before, during and after the reconfiguration, respectively. Notice that P is assigned to the same application during these different stages except that its schedule and partition specifications (availability factor and supply regularity) can vary from stage to stage. We now categorize the resource partitions during a reconfiguration into the following four categories.

Inserted Partition: P^o has an availability factor of 0 and P^t has an availability factor larger than 0. The R^{3}P requests to add this resource partition P into the system.

Deleted Partition: P^t has an availability factor of 0 and P^o has an availability factor larger than 0. The R^{3}P requests to remove this resource partition P from the system.

Unchanged Partition: P^o and P^t have the same availability factor (larger than 0) and the same supply regularity.

Reconfigured Partition: P^o and P^t have different availability factors and/or different supply regularity (all larger than 0).

We now extend the definition of instant regularity to accommodate the change of availability factor during reconfiguration.

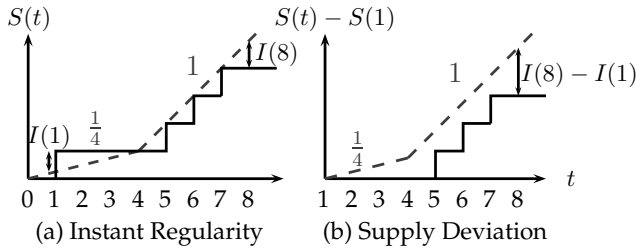

Fig. 6: Dotted line shows the ideal amount of resource supply which is $\frac{1}{4}$ from time 0 to 4 and 1 after time 4. $I(1), I(8)$ illustrates the instant regularity at time 1 and 8, respectively. $I(t)-I(1)$ illustrates the deviation of resource supply for time interval $[1, t)$.

Definition IV.2: The instant regularity $I(t)$ of a resource partition P at time $t \geq t_r$ is defined as $I(t) = S(t) - \alpha^o \cdot t_r - \alpha^t(t - t_r)$ where t_r is the time of a reconfiguration request λ; α^o/α^t are its availability factor before/after the request.

As an example shown in Fig. 6 (a), $I(1)$ indicates that there is resource over supply at time 1 while $I(8)$ indicates that there is resource under supply at time 8. Moreover, the difference of the instant regularity between any two time instants indicates the supply deviation for that time interval. As shown in Fig. 6 (b), $S(t) - S(1)$ indicates the resource supply during time interval $[1, t)$ and the dotted line illustrates the requested resource supply for time interval $[1, t)$. $I(8) - I(1)$ indicates the supply deviation for time interval $[1, 8)$.

Based on the extended definitions of the instant regularity, we now define the *reconfiguration supply regularity* as follow.

Definition IV.3: Let a, b, k be non-negative integers. The reconfiguration supply regularity of resource partition P is defined as R^r which equals to the smallest $k \geq 1$ such that $I(b) - I(a) > -k, \forall b \geq a$.

The reconfiguration supply regularity *only* defines the maximum supply shortfall while the normal supply regularity restricts *both* the maximum supply shortfall and surplus supply. This relaxation provides more flexibility when resolving the schedule conflicts during the reconfiguration. Based on this definition, the semantics of the performance guarantee for a resource partition during the reconfiguration can be illustrated in Fig 4. For any resource partition P, the amount of its supply shortfall will never exceed the reconfiguration supply regularity R^r for any time interval within or across the reconfiguration boundary. For all the other time intervals, the resource supply deviation is bounded by its normal supply regularity. In Fig. 4 (a) and (b), the resource supply deviation before/after the RPT is bounded by the normal supply regularity as the actual supply will follow the ideal supply curves. On the other hand, the resource supply may deviate for any time interval across the transition boundary which depends on its reconfiguration regularity R^r. In Fig 4 (a), the resource supply suffers a performance degradation bounded by R^r while the resource supply gets a temporary surplus in Fig 4 (b).

Similar to the definition of regular partition, we define the reconfiguration regular partition as follows.

Definition IV.4: Resource partition P is reconfiguration regular if and only if its reconfiguration supply regularity $R^r = 1$, and it is regular partition before and after the reconfiguration.

A reconfiguration regular partition P supplies the resource no less than requested fraction of resource over any time interval. To illustrate the concept, we give a numerical example in Fig. 5, where we construct two partition schedules as shown in Fig. 5 (d) and (e). In Fig. 5 (d), the maximum supply shortfall happens at time interval $[2, 8)$ for P_2 which is $I(8)-I(2) = \alpha_2 \times (-8+2) = -\frac{3}{2} < -1$ and this extra supply deviation makes P_2 not reconfiguration regular. In Fig. 5 (e), P_1 gets surplus supply while the schedule of P_2 is unchanged. This makes P_1, P_2 and P_3 all reconfiguration regular between P_3 and one of P_1 or P_2.

C. Scheduling Problem Statement and Algorithm Overview

With the above model extension, we are now ready to formalize the dynamic resource partition reconfiguration problem. We first make the following assumptions.

- Only one R^{3}P is allowed during each RPT stage.
- For each resource partition P, P^o and P^t are both regular but P^t can be reconfiguration irregular, i.e., the reconfiguration regularity of P can be larger than one.
- The availability factor of P is restricted to be power of $\frac{1}{2}$, but can be relaxed to $\frac{1}{x}, x \in \mathbb{N}$.
- Resources are assumed to be uniform, i.e., the length of resource slices is the same for all resources.

Problem IV.1: Dynamic Partition Reconfiguration (DPR): Given a reconfiguration request $\lambda = \{\mathcal{P}^t, \mathcal{R}^r, T\}$ and the tuples of the resource partitions before the request $\{P_i^o \mid \exists P_i^t \in \mathcal{P}^t\}$, compute the schedule of P_i^d, P_i^t for each resource partition $P_i^t \in \mathcal{P}^t$ such that the following three conditions are satisfied:

C-1: P_i^t meets the required regularity and availability factor;

C-2: the reconfiguration regularity of P_i is R_i^r;

C-3: the length of the RPT stage is no longer than T.

By satisfying condition **C-1**, the resource partition P_i successfully reconfigures its capability to supply resource according to the updated availability factor and supply regularity; condition **C-2** bounds the maximum performance degradation of individual partitions during the reconfiguration by specifying the reconfiguration regularity; condition **C-3** specifies the maximum length of the reconfiguration transition during which the system may suffer performance degradation.

The key challenge of the DPR problem is to ensure that each resource partition is constructed in a way that it can supply enough resources (defined by its availability factor, supply regularity and reconfiguration regularity) both during and after the reconfiguration. For this purpose, every time a resource slice is allocated to a resource partition, the next resource slice to the resource partition must be allocated to satisfy the performance requirement as specified by the conditions **C-1** and **C-2**. The conditions of **C-1** and **C-2** together impose a deadline for allocating the next resource slice to the resource partition. The problem of scheduling resource partitions is

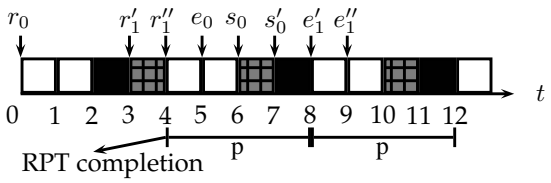

Fig. 7: An example of a unit-size transition task: its first instance is released at time r_0 with deadline e_0. Given different finish times, the release time and deadline of the next instance is computed accordingly. The schedule of the task is cyclic with period of p after the RPT stage.

akin to scheduling a set of tasks, where (1) a task instance is immediately released after the execution of its previous instance, (2) the relative deadline of the new instance is a function of the maximum supply shortfall indicator (to be defined later), and (3) each task follows a cyclic schedule after the RPT stage.

Fig. 7 depicts an example of such a task system. We assume that the task has a fixed maximum supply shortfall indicator of 5 and its deadline is the same as the indicator. It also has a period of 4 after the RPT stage. The first instance has release time $r_0 = 0$ and relative deadline $e_0 = 5$. If this instance is scheduled at time 2, it will release the second instance with release time $r'_1 = 3$ and deadline $e'_1 = 3 + 5 = 8$. This instance can also be scheduled at time 3. In this case, the second instance will be released at time $r''_1 = 4$ with its deadline $e''_1 = 4+5 = 9$. After the RPT stage is over, the task should be scheduled following a cyclic schedule with a period of 4 as illustrated in Fig. 7.

Before describing such a task system, we first define the maximum supply shortfall indicator d_i used for such a task system.

Definition IV.5: The maximum supply shortfall at any time t of a resource partition P is defined as $d(t) = \min_{b \le t}(I(t) - I(b)) = I(t) - \max_{b \le t}(I(b))$.

As illustrated in Fig. 6 (b), $I(t) - I(b)$ indicates the supply deviation of partition P in time interval $[b, t)$, and $d(t)$ defines the maximum supply shortfall for any time interval $[b, t), b \le t$.

We shall convert a partition system based on the offsets of P_i^o, performance requirement of P_i^t and reconfiguration regularity of P_i into a *unit-size transition task system*. A unit-size transition task system $\mathcal{T}$ is a set of tasks $\{\tau_1, \tau_2, \ldots, \tau_n\}$, where each task τ_i has execution time of one unit, a maximum supply shortfall d_i, a relative deadline e_i which is a function of d_i, and a period p_i[2]. In a unit-size transition task system, an instance of τ_i is immediately released after the execution of its previous instance. We will discuss how to convert the DPR problem into the unit-size transition task system and use the information from the latter to help solve the DPR problem in Section IV-D.

[2]Period of a transition task will be the same as the period of P_i^t.

Algorithm 1: Algorithm Overview for the DPR Problem

Input: The R^{3}P λ and the R^{3}P requesting time t_r
Output: Transition schedule $\mathcal{S}_i^d$ and cyclic schedule $\mathcal{S}_i^t$ for all $P_i \in \mathcal{P}$. Reject if no feasible schedule.

```
T^0 = Transformation(λ, t_r)
for t_b ← 0 to T do
    ({S_i^d | ∀P_i}, T^{t_b}) = TransitionSchedule(T^0, t_b)
    if T^{t_b} ≠ Null then
        {S_i^t | ∀P_i} = CyclicSchedule(T^{t_b})
    end
    if {S_i^t | ∀P_i} ≠ Null then
        return S_i^d and S_i^t, ∀P_i
    end
end
return NULL
```

To solve the DPR problem, we need to compute the transition schedule for the RPT stage and the cyclic schedules for all partitions after the reconfiguration. We propose a *three-stage algorithm* to break down the DPR problem into three subproblems. An overview of the algorithm is presented in Alg. 1. **Stage-1** of the algorithm transforms the partition system in the RPT into a unit-size transition task system (Alg. 2). We use $\mathcal{T}^t$ to denote the state of the task system at time t including release time r_i and deadline e_i, both are relative to t, of each task instance. In **Stage-2**, the algorithm searches for a feasible solution with an RPT duration of $t_b \le T$ and constructs the transition schedule for each P_i^d within that duration (Alg. 3). Based on the state information of the task system at the end of the RPT stage, **Stage-3** computes the cyclic schedules for individual P_i^t to meet their corresponding supply regularity and availability factor requirements (Alg. 4). The correctness of the algorithm is proved in Theorem IV.3.

D. Details of the Three-Stage Algorithm

We now present the details of the proposed algorithm.

1) Stage 1: Partition to Task Transformation: We present the details of the algorithm for transforming the partition system into a unit-size transition task system in this section. We first describe how to compute d_i and the relative deadline e_i for each instance of a task τ_i based on Def. 2.

Theorem IV.1: Given the reconfiguration request time $t_r > s_i^o$, where s_i^o is the resource slice offset of P_i^o, the maximum supply shortfall of resource partition P_i^o at time t_r, $d_i(t_r)$, can be computed as $d_i(t_r) = \alpha_i^o(s_i^o + 1 - t_1)$ where

$$t_1 = \begin{cases} t_r \bmod p_i^o + p_i^o & t_r \bmod p_i^o \le s_i^o \\ t_r \bmod p_i^o & o.w. \end{cases}$$

Proof. P_i^o (before the reconfiguration) is assumed to be regular and has an availability factor of the power of $\frac{1}{2}$, it has a single schedule offset s_i^o and will repeat with a period of p_i^o. Thus, we have $S(t_r) = S(t_1) + (t_r - t_1)/p_i^o$. Further by the definition of instant regularity (see Def. III.4) and the fact that $\frac{1}{p_i^o} = \alpha_i^o$, we have

$$I(t_r) = I(t_1) \tag{1}$$

Algorithm 2: Partition-to-Task-System Transformation

Input: The R^{3}P λ and the R^{3}P requesting time t_r
Output: $\mathcal{T}^0$, the state of the constructed unit-size transition task system at time 0.

1 **Procedure Transformation** (λ, t_r)
2 **for** $P_i \in \mathcal{P}$ **do**
3 $d_i = 0$
4 **if** P_i *is a reconfigured* or *unchanged* partition **then**
5 **if** $t_r \leq s_i^o$ **then**
6 $d_i = -\alpha_i^o \cdot t_r$
7 **else**
8 $t_t = t_r \mod p_i^o$
9 **if** $t_t \leq s_i^o$ **then**
10 $t_t = t_t + p_i^o$
11 **end**
12 $d_i = \alpha_i^o(s_i^o + 1 - t_t)$
13 **end**
14 **end**
15 $r_i = 0$; $e_i = \lfloor (R_i^r + d_i)/\alpha_i^t \rfloor$; $p_i = p_i^t$
16 **end**
17 **return** $\mathcal{T}^0$

For the same reason, we have $S(t) = S(s_i^o + 1) + \lfloor (t - (s_i^o + 1))/p_i^o \rfloor, \forall t > s_i^o$. Again by the definition of instant regularity and the fact that $\frac{1}{p_i^o} = \alpha_i^o$, we have

$$I(s_i^o + 1 + t) \leq I(s_i^o + 1)\ \forall t \in \mathbb{N} \tag{2}$$

By the definition of the maximum supply shortfall, Eq. 1, Eq. 2 and the fact that $S(t_1) = 1$, we have $d_i(t_r) = I(t_1) - I(s_i^o + 1) = \alpha_i^o(s_i^o + 1 - t_1)$. This completes the proof. □

Alg. 2 transforms each resource partition P_i to a task τ_i by computing its maximum supply shortfall indicator d_i and relative deadline e_i at the reconfiguration request time t_r. It computes $d_i(t_r)$ for each task (Line 7–12) according to Theorem IV.1. The release time and relative deadline of the first instance of τ_i is set as $r_i = 0$ and $e_i = \lfloor (R_i^r + d_i)/\alpha_i^t \rfloor$, respectively. Its period p_i is set equal to the period p_i^t of P_i.

2) Stage 2: Transition Schedule Computation: Given a unit-size transition task system that is transformed from a partition system in Stage 1 and with a time budget t_b to complete the reconfiguration, this stage computes the transition schedule for P_i^d by following two heuristic principles: (1) we employ the *deferrable scheduling (DS)-EDF* algorithm [29] where tasks are scheduled according to their earliest deadlines but each task is scheduled as late as possible to reduce the total number of instances during the RPT stage; and (2) if the deadline of a task instance calculated through the DS-EDF algorithm is larger than the time budget t_b, the algorithm will try to schedule it in an idle slice before t_b so that the deadline of the next instance of that task can be further deferred when entering Stage 3. This will significantly increase the schedulibility of the cyclic schedule construction in Stage 3.

Algorithm 3: Transition Schedule Computation

Input: The R^{3}P λ, the time budget t_b and $\mathcal{T}^0$, the state of transition task system at the time of R^{3}P
Output: Transition schedule $\{\mathcal{S}_i^d \mid \forall i\}$ and the state of transition task system at the end of RPT stage $\mathcal{T}^{t_b}$. Reject if no feasible schedule.

1 **Procedure TransitionSchedule** $(\mathcal{T}^0, t_b)$
2 **for** $t_t \leftarrow 0$ *to* t_b **do**
3 $m[t_t] = 0$ //initialize the data structure for schedules
4 **end**
5 Enqueue all task τ_i into a queue Q in the ascending order following (1) deadline e_i and (2) period p_i
6 **while** $Q \neq \emptyset$ **do**
7 Dequeue τ_i from Q
8 $l =$ **DS-EDF**(r_i, e_i, m, t_b)
9 **if** $l =$ *NULL* **then**
10 **if** $e_i \leq t_b$ **then**
11 **return** NULL // Deadline will miss
12 **end**
13 //no idle resource slice can be utilized, update the task state for Stage 3
14 $r_i = 0$
15 $e_i = e_i - t_b$
16 **else**
17 Add l to $\mathcal{S}_i^d$
18 $d_i = \min(0, d_i + 1 - \alpha_i^t(l + 1 - r_i))$
19 $r_i = l + 1$
20 $e_i = \lfloor (R_i^r + d_i)/\alpha_i^t \rfloor + l + 1$
21 Enqueue τ_i to Q
22 **end**
23 **end**
24 **return** $(\{\mathcal{S}_i^d \mid \forall P_i\}, \mathcal{T}^{t_b})$
25 **Procedure DS-EDF** (r, e, m, t_b)
26 **if** $e > t_b$ **then**
27 $e = t_b$
28 **end**
29 **for** $t_t \leftarrow e - 1$ *to* r **do**
30 **if** $m[t_t] = 0$ **then**
31 $m[t_t] = 1$
32 **return** t_t
33 **end**
34 **end**
35 **return** NULL

Fig. 8 gives an example to illustrate the two heuristic principles. We use r_i^j and e_i^j to denote the release time and deadline of the j-th instance of task τ_i, respectively. Each task has a relative deadline of 4. The algorithm first schedules the first instance of task 1 and 2 at time 3 and 2, respectively, as late as possible but before their deadlines by following principle (1). The derived deadline of the second instance of each task is larger than the time budget $t_b = 6$. The algorithm then utilizes the idle slices at time 4 and 5 to further defer their deadlines following principle (2). The second instance

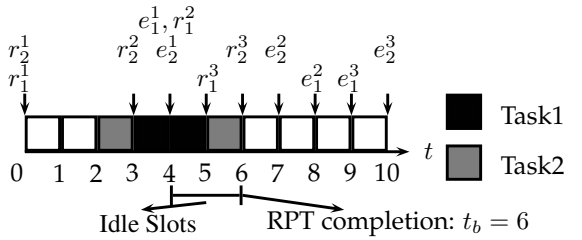

Fig. 8: An example of the transition schedule computation.

of task τ_2 is first scheduled, due to its smaller deadline, at time 5; task τ_1 is then scheduled at time 4. This defers the deadlines of task 1 and 2 further to e_1^3 and e_2^3 (instead of e_1^2 and e_2^2), respectively.

Note that for each new released task instance, its maximum supply shortfall indicator d_i and relative deadline e_i need to be recomputed. The next theorem shows how to update d_i.

Theorem IV.2: If there is exactly one resource slice scheduled at time $t+\delta-1$ in the interval $[t, t+\delta)$ for resource partition P_i, then $d_i(t+\delta)$, the maximum supply shortfall at time $t+\delta$, can be computed from $d_i(t)$ as follows.

$$d_i(t+\delta) = \min(0, d_i(t) + 1 - \alpha_i^t \cdot \delta) \tag{3}$$

Proof. According to Definition IV.5, there exists $b \leq t$ such that $I(b) \geq I(b')\ \forall b' \leq t$ and $d_i(t) = I(t) - I(b)$. Either (1) $I(t+\delta) \leq I(b)$ or (2) $I(t+\delta) > I(b)$ is true.

For case (1), by definition of instant regularity, we have

$$d_i(t+\delta) = S(t+\delta) - \alpha_i^o \cdot t_r - \alpha_i^t(t+\delta-t_r) - I(b) \tag{4}$$

Since there is only one resource slice scheduled at time $t+\delta-1$ during the time interval $[t, t+\delta)$, we have $S(t+\delta) = S(t)+1$. By substituting $S(t+\delta)$ in Eq. 4, the definitions of instant regularity and maximum supply shortfall, we have

$$d_i(t+\delta) = 1 + d_i(t) - \alpha_i^t \cdot \delta \tag{5}$$

For case (2) where $I(b) < I(t+\delta)$, we have $d_i(t+\delta) = I(t+\delta) - I(t+\delta) = 0$. By combining the two cases, we complete the proof. □

Alg. 3 summarizes the procedure for computing the transition schedule. At each scheduling decision, the algorithm schedules the task with the earliest deadline using the DS-EDF procedure (Line 8). The procedure constructs the transition schedule by (1) scheduling tasks as late as possible and (2) utilizing the idle slice before the time budget is used up. After each instance is scheduled, the indicator d_i, release time r_i and deadline e_i of the task are updated based on Theorem IV.2 (Line 18-20). If all task instances have deadlines larger than the time budget t_b and no more idle resource slice can be utilized, the algorithm updates the state of the task system and enters Stage 3 (Line 9-15).

Algorithm 4: Cyclic Schedule Computation

Input: $\mathcal{T}^{t_b}$, the sate of the task system after the RPT.
Output: Cyclic schedule $\{\mathcal{S}_i^t \mid \forall i\}$. Otherwise, reject.

1 **Procedure CyclicSchedule**($\mathcal{T}^{t_b}$)
2 Enqueue all task τ_i into a queue Q in the ascending order following (1) period p_i and (2) deadline e_i
3 **for** $t_t \leftarrow 0$ *to* p_{max} **do**
4 $m[t_t] = 0$
5 **end**
6 **while** $Q \neq \emptyset$ **do**
7 Dequeue τ_i
8 $l =$ **DS-EDF**$(0, e_i, m, p_i)$
9 **if** $l = NULL$ **then**
10 **return** NULL
11 **end**
12 Add l to $\mathcal{S}_i^t$
13 **for** $t_t \leftarrow 0$ *to* $p_{max}/p_i - 1$ **do**
14 $m[l + t_t \times p_i] = 1$
15 **end**
16 **end**
17 **return** $\{\mathcal{S}_i^t \mid \forall i\}$

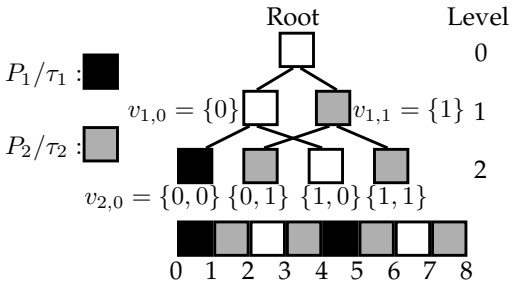

Fig. 9: The schedules can be encoded as a tree where each resource partition is assigned a sub-tree exclusively.

3) Stage 3: Cyclic Schedule Computation: Based on transition schedule computation from Alg. 3, this stage computes the cyclic schedule $\{\mathcal{S}_i^t \mid \forall i\}$ for every partition to meet its required regularity and availability factor after the reconfiguration. Before presenting the details of the algorithm, we first introduce a tree representation of the cyclic schedule. In this work, we encode a regular resource partition schedule as a tree structure called Index Schedule ($\mathcal{IS}$)-tree as depicted in Fig. 9. At each level $i \geq 0$ in the $\mathcal{IS}$-tree, there are 2^i number of nodes and each node represents a schedule assignment of a resource partition. Each node $v_{i,j}$ at level $i > 0$ represents a tree and is indexed as $v_{i,j} = \{x_1, \ldots, x_{2^i}\}$ where $x_k \in \{0, 1\}$, and the root node is indexed as $\{\}$. Each node $v_{i,j}$ has one left node and one right node indexed as $v_{i,j}^l = \{0, x_1, \cdots, x_{2^i}\}$ and $v_{i,j}^r = \{1, x_1, \cdots, x_{2^i}\}$, respectively. The binary coding of $v_{i,j}$ can be converted into a numerical value as $|v_{i,j}|$. A resource partition P with its schedule encoded as $v_{i,j}$ has access to resource in $[|v_{i,j}| + k \times 2^i, |v_{i,j}| + k \times 2^i + 1)\ \forall k \in \mathbb{N}$. For example, as illustrated in Fig. 9, resource partition P_1 assigned with node $v_{2,0}$ in the $\mathcal{IS}$-tree has access to the resource slices in $[0 + k \times 2^2, 0 + k \times 2^2 + 1)\ \forall k \in \mathbb{N}$. The value

of largest period p_{max} among all the partitions is denoted as $\max_i(p_i)$ where p_i is the period of τ_i.

Given the state of the transition task system at the end of the RPT, Alg. 4 assigns a node in the $\mathcal{IS}$-tree for each task using the DS-EDF procedure in Alg. 3. For a task with period p_i the procedure schedules the task as late as possible at level x where $2^x = p_i$. This procedure repeats until all the tasks have been assigned with an appropriate tree node. As an example in Fig. 9, the DS-EDF procedure will search for an available node at level 1 of the $\mathcal{IS}$-tree for τ_2 which has a deadline of 2 and a period of 2. Suppose that $v_{1,1}$ is available and assigned to τ_2, Alg. 4 will then mark all its child nodes ($v_{2,1}$ and $v_{2,3}$) as unavailable (Line 13-14). Note that for simplicity we use an array to implement the $\mathcal{IS}$-tree structure in the algorithm.

E. Properties of the DPR Algorithm

This section presents some important properties of the DPR algorithm, including its time complexity, correctness of the algorithm and some feasibility analysis.

Time complexity: We begin with the time complexity analysis of the DPR algorithm. In Stage 1, Alg. 2 has a complexity of $O(N)$, where N is number of resource partitions, as the transformation of a partition to a unit-size transition task is an $O(1)$ operation by Theorem IV.1. In Stage 2, Alg. 3 has a complexity of $O(NlogN)$, which is for building up the queue, plus $O((2T+N)\cdot logN)$ for dequeue and enqueue $2T+N$ times, and plus the complexity of the DS-EDF procedure which is $O(T^2)$. In Stage 3, Alg. 4 has a time complexity of $\sum_i(p_i + \frac{p_{max}}{p_i})$ to search available nodes for each task at level x with $2^x = p_i$ and mark the unavailable nodes. Alg. 4 also involves $O(2NlogN)$ the enqueue (dequeue) operations. This brings the total time complexity of the DPR algorithm to $O(N + T(4NlogN + 2TlogN + T^2 + \sum_i(\frac{p_{max}}{p_i} + p_i)))$.

Correctness: We now show the correctness of DPR algorithm.

Theorem IV.3: If the DPR algorithm terminates successfully, then the solution will satisfy all three conditions **C-1** to **C-3** as specified in the dynamic partition reconfiguration problem.

Proof. In Stage 3, Alg. 4 computes a schedule for each regular partition P_i with its targeted availability factor. This satisfies condition **C-1**. In Stage 2, Alg. 3 computes the transition schedule with a time budget $b_t \leq T$ and hence condition **C-3** is satisfied. We only need to prove that condition **C-2** is also satisfied where the reconfiguration regularity of P_i is R_i^r.

Step (1) To show that $\min(I(b) - I(a)) > -R_i^r,\ \forall b \geq a$, we only need to consider time intervals $[a, b)$, where b is at the start of a scheduled slice. For any other time interval $[a, b')$, we can always find a b such that there is no slice scheduled in time interval $[b', b)$ and this implies $S(b') = S(b)$, and $I(b) \leq I(b')$ by the definition of instant regularity (Def. IV.2). Hence, $I(b') - I(a) > -R_i^r$ if $I(b) - I(a) > -R_i^r$. Furthermore, by the definition of maximum supply shortfall (Def. IV.5), we only need to prove $d_i(b) > -R_i^r$ for all such b. Because P_i^o is regular and $-R_i^r \leq -1 < d_i(b), \forall b \leq t_r$ where t_r is the time of R^{3}P request, we can prove by induction that $d_i(b) > -R_i^r$ for all such time instant $b > t_r$.

Step (2) Assume that $b_0 > t_r$ is at the start of first resource slice after the time of R^{3}P request t_r. By the definition of maximum supply shortfall (Def. IV.5) and the fact that P_i^o is regular, there exists $x \leq t_r$ such that

$$d_i(t_r) = I(t_r) - I(x) > -1 \tag{6}$$

By Def. IV.2 and $S(t_r) = S(b_0)$, $I(b_0) = I(t_r) - \alpha_i^t(b_0 - t_r)$. Also, $I(x) \geq I(t_r) \geq I(x')\ \forall x' \leq b_0$ and $x' \geq t_r$ by the definition of maximum supply shortfall (Def. IV.5) and $S(t_r) = S(b_0)$. It follows that $d_i(b_0) = I(t_r) - \alpha_i^t(b_0 - t_r) - I(x)$. Furthermore by Eq. 6, we have

$$d_i(b_0) = d_i(t_r) - \alpha_i^t(b_0 - t_r) \tag{7}$$

According to the computation of deadline in Alg. 2, we have $b_0 - t_r \leq (R_i^r + d_i(t_r))/\alpha_i^t - 1$. From Eq. 7, we have $d_i(b_0) \geq -R_i^r + \alpha_i^t > -R_i^r$. We hence assume $d_i(b_k) > -R_i^r$, where b_k is at the start of some scheduled slice. We proceed to show that $d_i(b_{k+1}) > -R_i^r$ where $S(b_{k+1}) = S(b_k + 1)$.

Step (3) By the same reason in **Step (2)** to get Eq. 7, we have

$$d_i(b_{k+1}) = d_i(b_k + 1) - \alpha_i^t(b_{k+1} - (b_k + 1)) \tag{8}$$

From Alg. 4 and Alg. 3, consecutive resource slices are scheduled before their relative deadlines. We have $b_{k+1} - (b_k + 1) \leq$

$$\begin{cases} p_i^t - 1 = \frac{1}{\alpha_i^t} - 1 & or \\ (R_i^r + d_i(b_k + 1))/\alpha_i^t - 1 \end{cases}$$

Substituting $b_{k+1} - (b_k + 1)$ in Eq. 8, we have $d_i(b_{k+1}) \geq$

$$\begin{cases} d_i(b_k + 1) - 1 + \alpha_i^t & or \\ -R_i^r + \alpha_i^t \end{cases}$$

We hence have $d_i(b_{k+1}) > -R_i^r$ because $d_i(b_k + 1) > -R_i^r + 1 - \alpha_i^t$ by Thm. IV.2 and the fact that $d_i(b_k) > -R_i^r$.

From **Step (1)** and by mathematical induction using **Step (2)** and **(3)**, we show $\min(I(b) - I(a)) > -R_i^r, \forall b \geq a$. This completes the proof. □

Completeness and feasibility analysis: The following two theorems prove the completeness of Alg. 4 to compute the cyclic schedule in Stage 3 of the DPR algorithm and present a sufficient condition to perform a feasible R^{3}P.

Theorem IV.4: Given the state of a transition task system $\mathcal{T}^{t_b}$ at the end of the RPT stage, Alg. 4 can compute a feasible cyclic schedule if and only if $\mathcal{T}^{t_b}$ has feasible cyclic schedules starting at time t_b.

Proof. We prove this theorem by showing that any feasible cyclic schedule s of $\mathcal{T}^{t_b}$ can be systematically transformed to an equivalent schedule s' computed by Alg. 4 and the intermediate schedule is feasible in each step of the transformation. We note that a schedule with the following property is equivalent to the schedule computed by Alg. 4. For any task τ_1 having schedule encoded as $v_1 = v_{i,j}$ and any other node at the same level $v_2 = v_{i,k}$, one of the following conditions must be true: (1) $|v_2| \geq e_1$; (2) v_2 is assigned to a task τ_2

with $e_2 \leq e_1$. (3) v_2 is a child of a node assigned to a task with smaller period. (4) $|v_2| < |v_1|$.

The proof proceeds by transforming any feasible schedule s into a schedule conforming the aforementioned property by adjusting the schedule of each task τ_i. The tasks are sorted according to their (1) period p_i and (2) deadline e_i in an ascending order. For each task τ_i in this queue which is assigned node $v_i = v_{j,k}$, we check whether there is a nodes $v'_i = v_{j,k'}$ invalidating all of the 4 conditions. If there exists such node v'_i, we swap the entire sub-tree v_i with sub-tree v'_i and none of the deadlines will be violated. Any task τ' assigned on the sub-tree v'_i must either have $p_i < p'_i$ or $p_i = p'_i$ and $e_i < e'_i$. Suppose a task τ'_i is at node v'^k_i on the original sub-tree v'_i and will be placed at the corresponding position v^k_i on the new sub-tree v_i. Because $|v_i| < |v'_i|$, we have $|v^k_i| < |v'^k_i|$ and the τ'_i's deadline e'_i will not be violated after the swap. For example in Fig. 9, task τ_2 on sub-tree $v_{1,1}$ can be swapped to $v_{1,0}$ without violating its deadline.

After each swapping step, the schedule remains valid. After adjusting all the tasks, any pair of nodes will conform to one of the 4 conditions resulting an equivalent schedule computed by Alg. 4. This completes the proof. □

Theorem IV.5: An R^{3}P is always feasible if every partition $P_i \in \mathcal{P}^t$ has reconfiguration regularity R^r_i no less than 2.

Proof. One can simply set the time budget of reconfiguration to zero and perform the DPR algorithm. By Def. III.5, Def. III.6, Def. IV.5 and fact that resource partition has reconfiguration regularity $R^r_i > 1$, we have $d_i(t_r) > -1$. The deadline of each task τ_i will be $e_i = \lfloor (R^r_i + d_i(t_r))/\alpha^t_i \rfloor \geq \frac{1}{\alpha^t_i} = p_i$. If $e_i \geq p_i$ holds for every task τ_i and the total utilization of task is no greater than 1, Alg. 4 can compute the feasible schedule by allocating schedule in the ascending order of the periods. □

V. Performance Evaluation

In this section, we provide an experimental evaluation of the performance of the DPR algorithm. The simulation results are presented in Section V-A. We also applied the resource reconfigurability model to a real-life autonomous control system and demonstrate its effectiveness in Section V-B.

A. Simulation-based experiments

We first study the performance of the DPR algorithm using simulation with different settings. In the experiments, the availability factors of each individual partition before and after R^{3}P are randomly sampled from $\frac{1}{2^i}$ $(1 \leq i \leq 7)$. The reconfiguration supply regularity R^r_i of each resource partition is randomly sampled from $[1, 5]$ and the transition budget T is randomly sampled from $[0, 20]$. The number of partitions is randomly sampled from $[1, 15]$. We also compare the performance of the DPR algorithm with the optimal results which are computed by using an integer linear programming solver [30]. The solver is encoded to search for one valid schedule given the requirement of each R^3P for 600 seconds on a machine with Core(TM) i-5 3.5 GHz CPU.

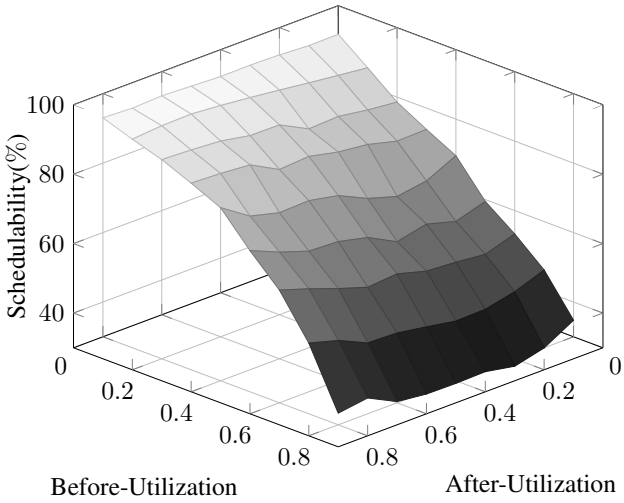

Fig. 10: R^{3}P schedulability with different utilization settings

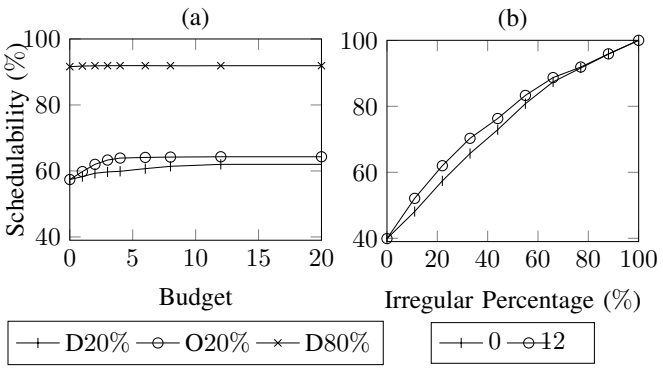

Fig. 11: Comparison of R^{3}P schedulability between different reconfiguration regularity and budget.

In the first set of experiments, we configure each resource partition to be reconfiguration regular and set the transition budget to be 0. These experiments aim to provide insights on the schedulability of R^{3}P with different partition utilization changes as the general case. From Fig. 10, it can be observed that the schedulability greatly depends on the before-utilization (the total utilization before R^{3}P) while the after-utilization (the total utilization before R^{3}P) has little effect.

From the general case, we can see that the schedulability of DPR algorithm is low when both after-utilization and before-utilization are high. Next we explore the schedulability of the DPR algorithm under heavy before-utilization and after-utilization settings. In the second set of experiments, they are both set to be 0.9 but the reconfiguration supply regularity and transition budget can be higher than 1 for the R^{3}P requests in these experiments. In Fig. 11 (a), each line represents a different fraction of partitions that have $R^r_i > 1$ and the x-axis denotes the transition budget. Line D20% (D80%, respectively) illustrates the results with 20% (80%, respectively) resource partitions having $R^r_i > 1$ from the DPR algorithm while line O20% illustrates the results with 20% resource partition having $R^r_i > 1$ from the integer linear programming solver. In Fig. 11 (b), each line represents a different budget

Application	Task	Context	RT Req. (ms)
PID Controller	Control Loop	Straight Ahead	(0.4, 128)
PID Controller	LIDAR Sensor	Straight Ahead	(0.4, 128)
Vison Controller	Image Recog.	Straight Ahead	(110, 200)
Vison Controller	Camera	Straight Ahead	(50, 200)
Communication	Send Signal	Both	(0.3, 128)
PID Controller	Control Loop	Turn Corner	(0.4, 64)
PID Controller	LIDAR Sensor	Turn Corner	(0.4, 64)

TABLE I: The autonomous control system has three applications. Each application consists of multiple tasks along with their real-time requirements in different contexts.

while the x-axis denotes the fraction of partitions that have $R_i^r > 1$. We make three important observations here: (1) the transition budget can help improve the schedulability but it has its limitation and only works for some extreme cases. Providing more budget does not necessarily improve schedulability. (2) The DPR algorithm performs comparably to any optimal approach in terms of schedulability. However, the solver takes 113 seconds on average while the DPR algorithm takes 0.00038 seconds to compute the schedule for budget equals to 20. (3) In Fig. 11 (b), it can be observed that the increase of R_i^r can significantly improve schedulability. Moreover, when all partitions have $R_i^r > 1$, the schedulability is 100% as proved in Thm. IV.5.

B. Case Study on the Autonomous F1/10 Model Car

We implemented the RRP resource model and the DPR approach on an F1/10 autonomous model car system to demonstrate the benefit of resource partitioning and reconfiguration in dynamic environment and compared the effectiveness of the DPR approach with a naive approach.

Our F1/10 autonomous car is built on the Traxxas Slash model car with the following major hardware components [31] as shown in Fig. 12 (a): 1) NVIDIA Jetson Tx2 embedded AI computing platform [32] running the software stack, 2) LIDAR sensor to measure the distance to surrounding objects, and 3) Zed stereo camera to capture front image. For the software stack, we integrated the LitmusRT framework [33], [34], a real-time extension of Linux kernel 4.9.30, with the NVIDIA downstream kernel 4.4 to provide the resource partitioning function using the P-RES scheduler. To support applications in user space, we also implemented a library based on the Robot Operating System (ROS) framework [35] to enable the ROS applications to (1) operate as sets of periodic processes and (2) request static and online resource reconfiguration.

We now explain how our F1/10 model car system achieves dynamic resource reconfiguration. Suppose that the car control system aims to race in a $35m$ by $8m$ rectangular track as fast as possible while avoiding any obstacles. The control system has three applications running: PID Controller, Vision Controller, and Communication. As summarized in Table I, each application contains a set of tasks and each task is associated with the corresponding real-time requirements (worst-case execution time, task period). The requirement of each application on this car system is specified by its developer and can vary on different hardware platforms. The PID Controller generates control signals to the motor and steering system to avoid obstacles, and it coordinates the control loop task with the LIDAR sensor task. The Vision Controller identifies the traffic signs (red circles) that are used to indicate that there exists a corner ahead. The Communication application couples the control system and the motor and steering system by exchanging sensing and control messages. The environment of our case study can be classified into two contexts: "Straight Ahead" and "Turn Corner". In the Straight Ahead context, the system allocates most resources to the Vision Controller for detecting the traffic sign while moving as fast as possible in our laboratory corridor (Fig. 12 (b)). After the traffic sign is detected, the system enters the Turn Corner context where it slows down and makes the turn (Fig. 12 (c)). After the car goes around the corner, the system enters the Straight Ahead context again. During these context switch, the car control system will adapt its application and reconfigure the resource interfaces accordingly so that the requirement of each application in different contexts can be satisfied[3].

Our case study uses a simple application transition model. We demonstrate the PID Controller and Communication applications cannot tolerate any extra latency while the Vision Controller application can tolerate an extra delay less than $100ms$ during context transitions. The system enters the "Turn Corner" context when the Vision Controller detects a traffic sign, and it enters the "Straight Ahead" context when the PID Controller finishes executing the "Turn Corner" operation. The relationships between the behavior and requirement of each task in every context is summarized in Table I. The last column in the table gives the real-time requirement of each task in milliseconds which represents the worst-case execution time and the period of the task. Note that the execution time of the Vision Controller varies a lot in the test scenarios and depends on the speed of the image processing application. When the goal is to finish the race as fast as possible, the faster the Vision Controller can finish processing and identify the traffic signs ahead, the faster the car can run. In the experiments, we scheduled all the three applications on one Denver core on Jetson Tx2 [32] and run all other non-real time tasks on the other cores. This minimizes the impact of the resource slice blocked by the kernel interrupt handling to avoid failures of the control system. Based on the requirement of each application, the system allocates resource partition P_1 to the PID Controller, P_2 to the Vision controller and P_3 to the Communication. Each application runs a round-robin scheduler to schedule its own task group. The control system issues a reconfiguration request when entering each context. When entering the Straight Ahead context, $\lambda_s = \{\mathcal{P}_s^t, \mathcal{R}_s^r, 100\}$ where $P_1^s, P_3^s \in \mathcal{P}_s^t$ are regular with availability factor $\alpha_1^s = \alpha_3^s = \frac{1}{128}$ while $P_2^s \in \mathcal{P}_s^t$ has $\alpha_2^s = \frac{63}{64}$. The reconfiguration regularity of P_1, P_2, P_3 is $1, 1, 100$, respectively. When entering the Turn Corner context, $\lambda_c = \{\mathcal{P}_c^t, \mathcal{R}_c^r, 100\}$ where $P_1^c, P_3^c \in \mathcal{P}_c^t$ are regular partitions with availability factor $\alpha_1^c = \frac{1}{64}, \alpha_3^c = \frac{1}{128}$ while P_2^c is a deleted partition which has $\alpha_2^c = 0$. The reconfiguration

[3]A demonstration video can be found on Youtube in the following link: http://www.youtube.com/watch?v=8b-MMP3-cug

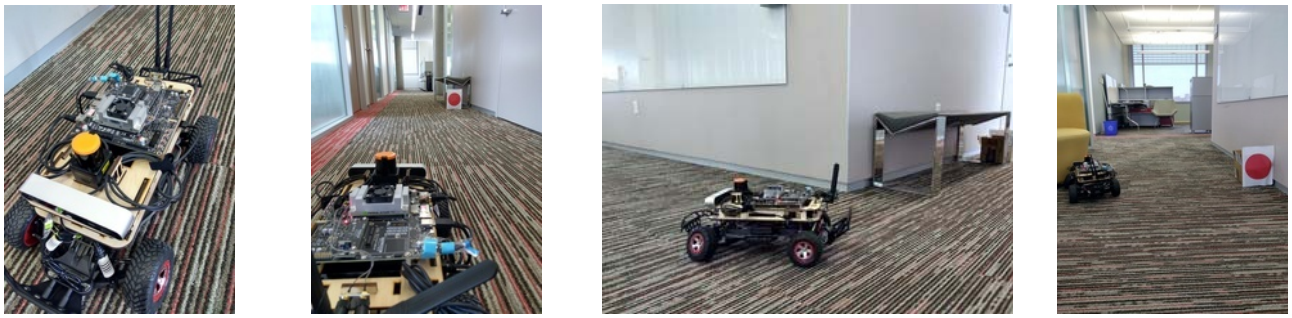

Fig. 12: A case study of dynamic partition reconfiguration on the autonomous F1/10 model car: (a) hardware components, (b) the "Straight Ahead" context, (c) the "Turn Corner" context, and (d) a system failure observed using the naive algorithm.

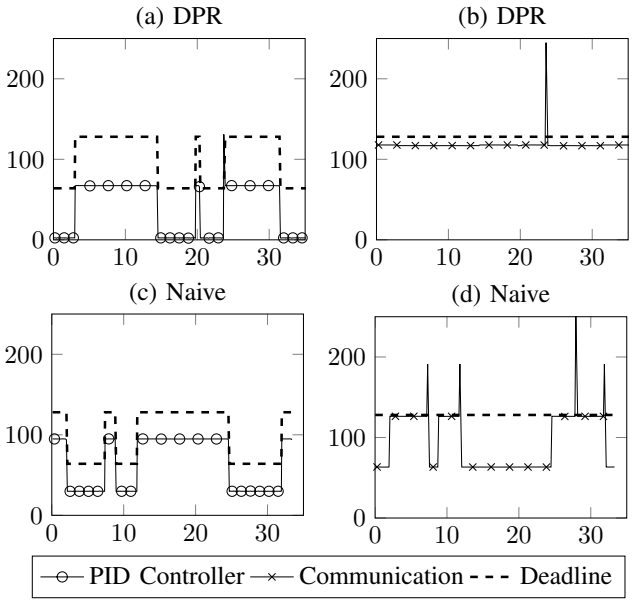

Fig. 13: The response time (ms) of each application at each time point (seconds) using the DPR algorithm (a), (b) and using the naive algorithm (c), (d).

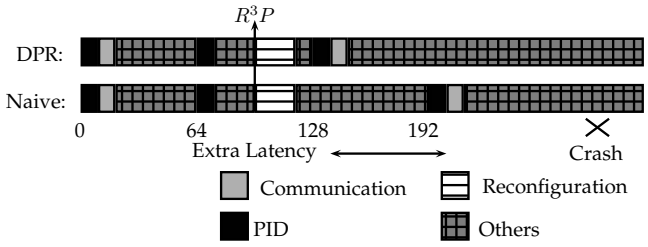

Fig. 14: The schedules computed by the DPR and Naive algorithms.

regularity of P_1, P_3 is 1 for the Turn Corner R^{3}P. Each reconfiguration request has a response time of $50ms$ in the experiments.

We capture the performance of the system by measuring the response time of every task instance of the PID Controller and Communication applications. Failure to guarantee enough resource supply to these applications may result in system failure. For system running the DPR algorithm, it not only considers the current supply shortfall of each resource partition but also the response time of a reconfiguration. The operation of a reconfiguration is considered as a single instance transition task which has both execution time and deadline of $50ms$ released at the time of each reconfiguration request. For the case where system running a naive algorithm, the system simply computes two schedules for each context and swaps them during each reconfiguration.

Fig. 13 shows the response time and deadline of each task instance. When the system runs the DPR algorithm, the response time of most task instances are stable and below the corresponding deadline as shown in Fig. 13 (a) and (b). There may be unusual response time spikes such as around time 25. The activity and scheduling traces indicate that it is because the reconfiguration operation takes longer than the estimated 50ms. On the other hand, a naive algorithm will be more likely to under-supply the resource partitions during reconfiguration as shown in Fig.13 (c) (d). Spikes of response time are often close to the time of reconfiguration and the spikes indicate that P_1 and P_3 are reconfiguration irregular. The performance degradation at time 32 causes the car to fail in avoiding obstacles as illustrated in Fig. 12 (d). Fig. 14 illustrates the different schedules computed by DPR and naive algorithm in this condition where the context changes from the Turn Corner to the Straight Ahead. One can see that there is a huge starvation interval between time 2 and 193 for the Communication application in the naive approach. This results in the extra latency and eventually causes the car to crash. Note that the spike at time 28 in Fig. 13 (d) is caused by the fact that the ROS library is not a hard real-time library which sometimes takes longer to execute certain functionality.

VI. Conclusion

In this paper, we study the dynamic partition reconfiguration (DPR) problem by: (1) Proposing a precise semantics of resource provisioning during resource reconfiguration for CPS in the open system environment; this helps to avoid unexpected system instability problems. (2) Presenting a novel DPR algorithm to satisfy the performance requirements of partition reconfiguration requests. (3) Demonstrating the benefit of the DPR approach on a real-life open system application.

ACKNOWLEDGEMENT

The work reported herein is supported by the Office of Naval Research under ONR Award N00014-17-1-2216.

REFERENCES

[1] Z. Deng and J.-S. Liu, "Scheduling real-time applications in an open environment," in *18th IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 1997, pp. 308–319.

[2] M. M. Herterich, F. Uebernickel, and W. Brenner, "The impact of cyber-physical systems on industrial services in manufacturing," *Procedia Cirp*, vol. 30, pp. 323–328, 2015.

[3] A. X. Feng, "Design of real-time virtual resource architecture for largescale embedded systems," Ph.D. dissertation, Department of Computer Science, The University of Texas at Austin, 2004.

[4] W.-J. Chen, P.-C. Huang, Q. Leng, A. K. Mok, and S. Han, "Regular composite resource partition in open systems," in *38th IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2017, pp. 34–44.

[5] I. Shin and I. Lee, "Periodic resource model for compositional real-time guarantees," in *24th IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2003, pp. 2–13.

[6] I. Shin, A. Easwaran, and I. Lee, "Hierarchical scheduling framework for virtual clustering of multiprocessors," in *20th Euromicro Conference on Real-Time Systems (ECRTS)*. IEEE, 2008, pp. 181–190.

[7] A. Easwaran, M. Anand, and I. Lee, "Compositional analysis framework using edp resource models," in *28th IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2007, pp. 129–138.

[8] J. Boudjadar, J. H. Kim, L. T. X. Phan, I. Lee, K. G. Larsen, and U. Nyman, "Generic formal framework for compositional analysis of hierarchical scheduling systems," in *21st IEEE International Symposium on Real-Time Distributed Computing (ISORC)*. IEEE, 2018, pp. 51–58.

[9] Y. Li and A. M. Cheng, "Toward a practical regularity-based model: The impact of evenly distributed temporal resource partitions," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, no. 4, p. 111, 2017.

[10] Y. Li and A. M. K. Cheng, "Transparent real-time task scheduling on temporal resource partitions," *IEEE Transactions on Computers*, vol. 65, no. 5, pp. 1646–1655, 2015.

[11] A. Burns and R. I. Davis, "A survey of research into mixed criticality systems," *ACM Computing Surveys (CSUR)*, vol. 50, no. 6, p. 82, 2018.

[12] D. de Niz and L. T. Phan, "Partitioned scheduling of multi-modal mixed-criticality real-time systems on multiprocessor platforms," in *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2014, pp. 111–122.

[13] C. Evripidou and A. Burns, "Scheduling for mixed-criticality hypervisor systems in the automotive domain," in *WMC 2016 4th International Workshop on Mixed Criticality Systems*, 2016.

[14] M. Neukirchner, K. Lampka, S. Quinton, and R. Ernst, "Multi-mode monitoring for mixed-criticality real-time systems," in *2013 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*. IEEE, 2013, pp. 1–10.

[15] X. Gu and A. Easwaran, "Dynamic budget management with service guarantees for mixed-criticality systems," in *2016 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2016, pp. 47–56.

[16] B. Hu, K. Huang, G. Chen, L. Cheng, and A. Knoll, "Adaptive workload management in mixed-criticality systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, no. 1, p. 14, 2016.

[17] J. Schlatow, M. Möstl, R. Ernst, M. Nolte, I. Jatzkowski, M. Maurer, C. Herber, and A. Herkersdorf, "Self-awareness in autonomous automotive systems," in *Proceedings of the Conference on Design, Automation & Test in Europe*. European Design and Automation Association, 2017, pp. 1050–1055.

[18] B. Hu, L. Thiele, P. Huang, K. Huang, C. Griesbeck, and A. Knoll, "Ffob: efficient online mode-switch procrastination in mixed-criticality systems," *Real-Time Systems*, pp. 1–43, 2018.

[19] R. I. Davis, S. Altmeyer, and A. Burns, "Mixed criticality systems with varying context switch costs," in *24th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2018, pp. 140–151.

[20] J. Real and A. Crespo, "Mode change protocols for real-time systems: A survey and a new proposal," *Real-time systems*, vol. 26, no. 2, pp. 161–197, 2004.

[21] A. Burns, "System mode changes-general and criticality-based," in *Proc. of 2nd Workshop on Mixed Criticality Systems (WMC)*, 2014, pp. 3–8.

[22] J. Lee, H. S. Chwa, L. T. Phan, I. Shin, and I. Lee, "Mc-adapt: Adaptive task dropping in mixed-criticality scheduling," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, no. 5s, p. 163, 2017.

[23] T. Chen and L. T. X. Phan, "Safemc: A system for the design and evaluation of mode-change protocols," in *25th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2018, pp. 105–116.

[24] H. Xu and A. Burns, "A semi-partitioned model for mixed criticality systems," *Journal of Systems and Software*, vol. 150, pp. 51–63, 2019.

[25] L. T. Phan, I. Lee, and O. Sokolsky, "Compositional analysis of multi-mode systems," in *22nd Euromicro Conference on Real-Time Systems (ECRTS)*. IEEE, 2010, pp. 197–206.

[26] H. Li, M. Xu, C. Li, C. Lu, C. Gill, L. Phan, I. Lee, and O. Sokolsky, "Multi-mode virtualization for soft real-time systems," in *24th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2018, pp. 117–128.

[27] V. Nikolov, S. Wesner, E. Frasch, and F. J. Hauck, "A hierarchical scheduling model for dynamic soft-realtime system," in *29th Euromicro Conference on Real-Time Systems (ECRTS)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[28] Y. Li and A. M. Cheng, "Static approximation algorithms for regularity-based resource partitioning," in *33rd IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2012, pp. 137–148.

[29] S. Han, D. Chen, M. Xiong, K.-Y. Lam, A. K. Mok, and K. Ramamritham, "Schedulability analysis of deferrablescheduling algorithms for maintainingreal-time data freshness," *IEEE Transactions on Computers*, vol. 63, no. 4, pp. 979–994, 2012.

[30] G. Developer. Gurobi. [Online]. Available: http://gurobi.com/

[31] F. Team. F1tenth design. [Online]. Available: http://f1tenth.org/

[32] NVIDIA. Embedded systems developer kits and modules. [Online]. Available: http://devblogs.nvidia.com/jetson-tx2-delivers-twice-intelligence-edge/

[33] J. M. Calandrino, H. Leontyev, A. Block, U. C. Devi, and J. H. Anderson, "LITMUSRT: A testbed for empirically comparing real-time multiprocessor schedulers," in *27th IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2006, pp. 111–126.

[34] B. Brandenburg and J. H. Anderson, "Scheduling and locking in multiprocessor real-time operating systems," Ph.D. dissertation, The University of North Carolina at Chapel Hill, 2011.

[35] R. Developers. Ros framework. [Online]. Available: http://wiki.ros.org/