Real-Time Systems (2021) 57:302–345
https://doi.org/10.1007/s11241-021-09364-5

Online reconfiguration of regularity-based resource partitions in cyber-physical systems

Wei-Ju Chen[1] · Peng Wu[2] · Pei-Chi Huang[3] · Aloysius K. Mok[1] · Song Han[2]

Accepted: 12 January 2021 / Published online: 1 March 2021

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 resource requirement changes, 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 at run time. We formalize and solve the *Dynamic Partition Reconfiguration (DPR)* problem for uniform environment where the minimal intervals assigned to each task for execution on each resource are the same. Extensive experiments have been conducted to evaluate the performance of the proposed approaches 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 proposed approach in a real-world cyber-physical system.

Keywords Regularity-based resource partition (RRP) · Online reconfiguration · Open system environment

✉ Wei-Ju Chen
albertwj@cs.utexas.edu

Extended author information available on the last page of the article

1 Introduction

A cyber-physical system (CPS) may consist of multiple applications that share resources from the same resource pool. In an open system environment (Deng and Liu 1997; Herterich et al. 2015), 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.

The Regularity-based Resource Partition (RRP) model is an effective strategy to allocate resource in such environment. 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 (Deng and Liu 1997; Feng 2004; Boudjadar et al. 2018; Li and Cheng 2017). For example, an autonomous car may have an application for keeping the car in a traffic lane and another application for detecting obstacles ahead. A component/application may consist of several sub-components (sub-tasks). A parent component distributes its share of resource to its sub-components and each of which in turn distributes it to its sub-components in a hierarchical fashion. Figure 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) (Shin and Lee 2003) or its variant the Explicit Deadline Periodic (EDP) model (Easwaran et al. 2007). The PRM model characterizes the resource interface using a per-period resource budget and an execution period parameter; while the EDP model further introduces a relative deadline parameter. The EDP resource interface defines a bandwidth server such that the server supplies the required amount of resource according to the budget in every period within the relative deadline. Our RRP model (Feng 2004; Chen et al. 2017; Li and Cheng 2017) characterizes the resource interface by two parameters: an

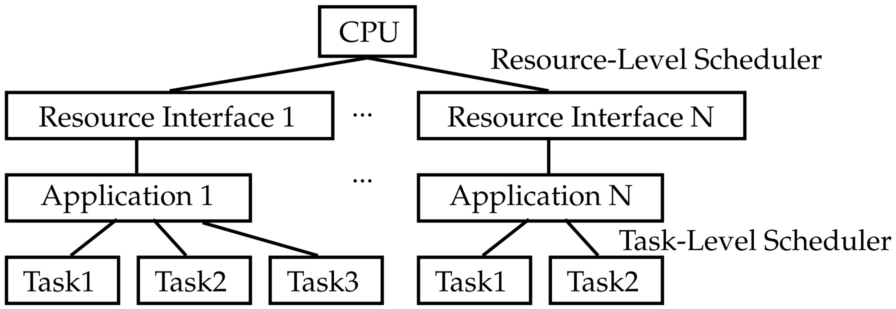


Fig. 1 Overview of the hierarchical scheduling model by taking the CPU resource as an example

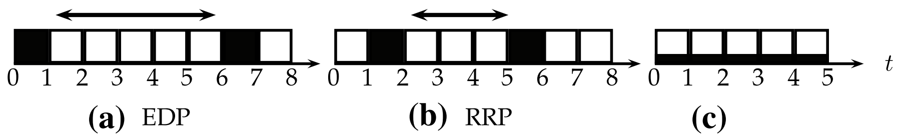


Fig. 2 **a** and **b** two possible schedules for a resource interface with a bandwidth of $\frac{1}{4}$ under the EDP and RRP models, respectively. **c** The ideal resource supply from the application's point of view with a bandwidth assignment of $\frac{1}{4}$ of the resource

availability factor and a *supply regularity*. The availability factor defines the bandwidth of the resource supply and the supply regularity defines the maximum supply deviation (or *supply jitter*) from the ideal supply bandwidth.

The main difference between the RRP and PRM/EDP 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, the resource should be supplied uniformly as if it is dedicated to the application, but at a slower rate ($\frac{1}{4}$) as depicted in Fig. 2c. However, the resource is allocated in units of some integer time intervals. The resource interface under the RRP model approximates the ideal resource supply by specifying the supply jitter. Supplying resource according to the desired fraction of resource allows that changes made to the task group can be more easily accommodated by the application within its allocated resource partition. This decreases the chance that the resource interface needs to be altered in response to the change of task group. For example, a set of periodic tasks with total utilization U can be independently scheduled on a regular partition with availability factor α by an EDF (Earliest Deadline First Liu and Layland (1973)) scheduler if $\alpha \geq U$ Feng (2004). In fact, the above mentioned periodic task scheduling problem on a logical resource can be transformed to the periodic task scheduling problem on a dedicated single resource (Li and Cheng 2015). Classic schedulers, such as EDF, DM (Deadline Monotonic (Liu and Layland 1973)) and FIFO (First In First Out) scheduler, can be reused. However, the jitter requirements make the designs of scheduling algorithms under the RRP model more complex than those under the EDP model.

Although the RRP resource interface can be used to mask the resource requirement changes within a partition, the reconfiguration among the partitions may still need to be performed at run time. To illustrate the online partition reconfiguration and its potential 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 computation requirements of the CPU are 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. Any resource interface model scheduled by a dynamic or static scheduler may lead to such problem if the performance semantics of the resource reconfiguration is not considered.

The performance semantics of a resource reconfiguration specify what timing constraints can or cannot be missed for the time intervals overlapping with the time of the resource reconfiguration. Different applications may have different performance semantics during the resource reconfiguration. One naïve choice is to ignore deadline misses during the resource reconfiguration by simply switching to the new schedule upon the time of the reconfiguration request. Our results with the car control system experiments show that this may lead to system failure, *e.g.*, car crashing. A better way is to specify certain invariant that must be maintained during the course of the resource reconfiguration such as defining and guaranteeing the resource supply during the reconfiguration.

The real-time performance guarantee of the partitions 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. Other work has investigated the dynamic reconfiguration problem (Evripidou and Burns 2016; Phan et al. 2010; Li et al. 2018; Nikolov et al. 2017). Most of those work study the task scheduling problem focusing on how to schedule a set of tasks that may transit to other modes and the real-time requirement of each task in each mode is given. In this paper, we study the problem of the partition scheduling problem where the problem is to schedule a set of partitions that may be reconfigured and there may be temporary performance degradation or performance guarantee violation depending on the performance semantics of the reconfiguration. To the best of our knowledge there is no previous work that (1) addresses the precise semantics of the resource reconfiguration that may cause system instability issues in the open system environment, and (2) considers the partition scheduling problem where performance semantics of resource reconfiguration is considered and temporary performance degradation may happen during the reconfiguration.

More specifically, in this paper we study the Dynamic Partition Reconfiguration (DPR) problem under the RRP model in uniform environment where the size of each resource slice is the same among the resources. 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. We then formalize the DPR problem and present a three-stage algorithm to construct both the transition schedule during the reconfiguration and the cyclic 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 is also presented on a real-life autonomous car control system which requires dynamic resource reconfiguration. This application

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

demonstrates the necessity for online resource reconfigurability to prevent system instability and shows the effectiveness of our approach.

In the rest of this paper, Sect. 2 summarizes the related work, and Sect. 3 reviews the RRP model. Section 4 describes the main challenge of online resource partition reconfiguration, defines the semantics of performance guarantee during reconfiguration and gives the precise definition of the DPR problem. The detail of our three-stage algorithm for solving the DPR problem is provided in Sect. 4. The performance evaluation and a real-life case study are presented in Sect. 5. Section 6 concludes this work and discusses the future work.

2 Related work

A hierarchical real-time system with timing constraints integrates a group of applications with multiple tasks on the same resource pool (Deng and Liu 1997; Feng 2004; Shin and Lee 2003). Most systems use two-level schedulers to achieve this (Deng and Liu 1997; Evripidou and Burns 2016; Biondi et al. 2018; Evripidou and Burns 2016; Li et al. 2018). One scheduler is used for scheduling applications such as assigning each application with a virtual machine. Virtual machines are scheduled by a resource scheduler and each application will have its own scheduler for scheduling the tasks in the virtual machine.

The concept of regularity was first introduced by Shirero et al. (1999) and was then extended to the regularity-based resource partition model by Mok and Alex (2001). Mok and Feng introduced the irregular partition and presented the AAF-based scheduling algorithm to schedule regularity-based resource partition (Mok and Alex 2001; Feng 2004). Li and Cheng then extended the AAF-based scheduling algorithm to uniform multi-resource environment and developed an optimized partitioning algorithm (Li and Cheng 2017). Besides the RRP model, there are plenty of studies on hierarchical scheduling which characterize resource interfaces using different models (Feng 2004; Shin and Lee 2003; Easwaran et al. 2007; Boudjadar et al. 2018). The most popular model among them is the EDP model which we discussed and compared with the RRP model in Sect. 1. In this paper, we shall focus on the dynamic partition reconfiguration problem under the RRP model.

There are several related research areas on scheduling tasks with varying timing requirements. Burns and Davis have a survey on mixed-criticality systems (Burns and Davis 2018), in which the task period, worst-case execution time and deadline depend on the system state/criticality. In multi-mode systems (de Niz and Phan 2014; Evripidou and Burns 2016; Neukirchner et al. 2013; Gu and Easwaran 2016; Hu et al. 2016; Schlatow et al. 2017; Hu et al. 2018; Davis et al. 2018), systems with mode changes require the design of new protocols such as (Real and Crespo 2004; Burns 2014; Evripidou and Burns 2016; Lee et al. 2017; Chen and Phan 2018; Xu and Burns 2019) 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. The DPR problem to be addressed in this paper faces the similar challenge where applications may suffer from system instability because of the undefined performance semantics during the resource reconfiguration. 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 regularity, 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 protocols. Thus, existing task-level mode switch 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 run time.

In addition, there was some research work on the multi-mode resource interface (Evripidou and Burns 2016; Phan et al. 2010; Li et al. 2018; Nikolov et al. 2017) where the resource interface may change for single resource environment. For instance, Evripidou and Burns (2016) used a two-level scheduler or a hyper-visor to handle the criticality mode change. Phan et al. (2010) proposed a compositional analysis of the multi-mode resource interface. Li et al. (2018) used virtual machine (VM) to support multi-mode virtualization where the parameters of the VM change with minimum transition latency. For multi-resource environment, some researchers used end-to-end reservation approaches to achieving performance isolation. For example, Buttazzo et al. (2010, 2011) proposed a method for allocating a set of parallel real-time tasks with time and precedence constraints on different multicore platforms by abstracting the computing power available into interface specifications.

Although the literature is rich, none of those work studies (1) the precise semantics of the resource reconfiguration that may cause system instability issues in the open system environment; and (2) the partition scheduling problem where the performance semantics of resource reconfiguration is considered and temporary performance degradation may happen during the reconfiguration.

3 RRP model

This section revisits the regularity-based resource partition (RRP) model which is the foundation of the dynamic partition reconfiguration problem to be elaborated in Sect. 4. We first define the time systems used in this paper and then review the important concepts in the RRP model in single-resource environment. These concepts were mostly presented in Mok and Alex (2001), Feng (2004), Chen et al. (2017).

3.1 Time systems

In the RRP model, we have three time systems as illustrated in Fig. 3. The first one is the wall clock time defined as the *physical time* τ, which is the same and synchronized among all physical resources as illustrated in Fig. 3a. For the physical resource Π, a minimum physical time interval (2 in the example shown in Fig. 3) that is non-preemptive and allocated to an application exclusively is defined as a *resource slice*.

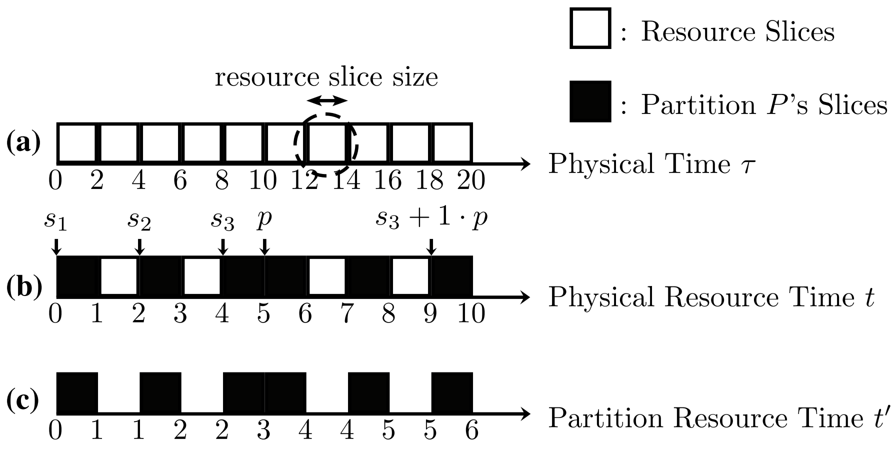


Fig. 3 **a** Physical resource Π is allocated in units of resource slices and a resource partition P is a set of allocated resource slices. **b** and **c** Show the two time systems for application utilizing the physical resource time and the partition resource time, respectively

The physical resource is allocated to the application(s) in units of resource slices as illustrated in Fig. 3b (Feng 2004), where a *resource partition P* is a set of resource slices. The second time system, *physical resource time*, is defined as follows.

Definition 3.1 The physical resource time t of a physical resource Π is a function of the physical time τ such that $t = \frac{\tau}{Q}$ where Q is the resource slice size of Π.

In addition to the physical time and physical resource time, a resource partition also has a logical clock defined as the *partition resource time* which denotes the amount of resource slices this resource partition has offered by that time from physical time zero as illustrated in Fig. 3c.

Definition 3.2 The partition resource time t' of a resource partition P is defined as the amount of resource slices having been offered by that time from physical time zero.

t and t' represent the physical resource time and partition resource time, respectively. If t or t' is a non-negative integer, this indicates that it is at the boundary of a resource slice in its corresponding time system; if t or t' is a non-integer value, this indicates that it is within a resource slice in its corresponding time system. As illustrated in Fig. 3, non-negative integer $t(t')$ denotes a time at the boundaries of resource slices. In this paper, the domain of physical time is assumed to have only non-negative integers and each resource slice starts and ends at physical time integral boundaries. The scheduling decisions made by the resource-level scheduler are always at the integral domain of physical/partition resource time even though the resource slice size may be different for different physical resources. We always refer the time to be physical resource time unless we specify the time to be others in this paper.

Moreover, we also assume that the resource slices have an equal size for the same physical resource. If all physical resources to be scheduled have the same resource slice size, the resource environment is *uniform*. Otherwise, the resource environment is *non-uniform*. In this paper, we mainly focus on addressing the dynamic partition reconfiguration problem in uniform environment. The extension of the proposed approaches to the non-uniform environment will be briefly discussed in Sect. 6.

3.2 Regularity-based resource partition in uniform environment

We now give the formal definition of a regularity-based resource partition in the uniform environment.

Definition 3.3 A resource partition P on a physical resource Π is a tuple $(\mathcal{S}, p)$, where $\mathcal{S} = \{s_1, s_2, \dots, s_n : 0 \le s_1 < s_2 < \dots < s_n < p\}$ is a set of n time points that denote the start time 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 $[s_k + x \cdot p,\ s_k + 1 + x \cdot p)$, $x \in \mathbb{N}, 1 \le k \le n$.

Definition 3.4 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.

Based on the definition of the supply function, we can also redefine the partition resource time as follows.

Definition 3.5 The partition resource time t' of a resource partition P is a function of the physical resource time t of the underlying physical resource such that $t' = S(\lfloor t \rfloor) + (S(\lceil t \rceil) - S(\lfloor t \rfloor)) \cdot (t - \lfloor t \rfloor)$ where $S(t)$ is the supply function of P.

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.

Definition 3.6 The availability factor α of a resource partition $P = (\mathcal{S}, p)$ is defined as $\alpha = \frac{|\mathcal{S}|}{p}$ where $|\mathcal{S}|$ is the number of elements in $\mathcal{S}$.

Definition 3.7 The instant regularity $I(t)$ for a resource partition P at time t is defined as $I(t) = S(t) - \alpha \cdot t$.

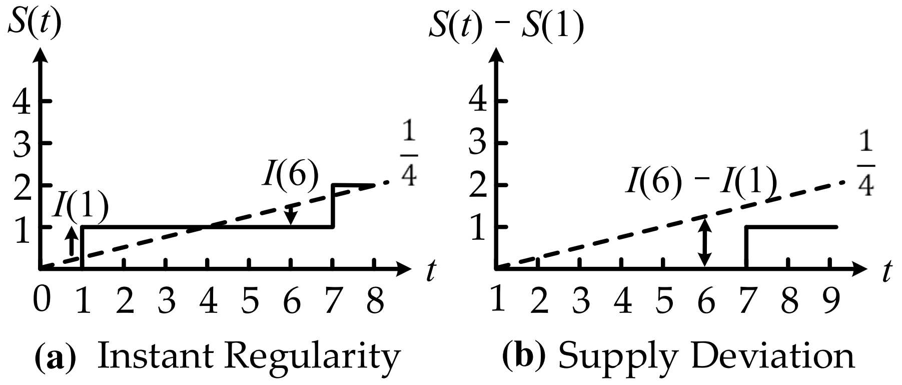


Fig. 4 Illustration of the concepts of availability, instant regularity and supply regularity in RRP model

Definition 3.8 Let a, b, k be non-negative integers. The supply regularity R of resource partition P is defined as the smallest k such that $|I(b) - I(a)| < k, \forall b \geq a$.

Figure 4a illustrates the ideal and actual resource supply of a resource partition P which is defined as $P = \big(\{s_1 = 1\}, 6\big)$. P has availability of $\frac{1}{4}$; and the supply function of P has $S(0) = 0$, $S(1) = 1$, $S(2) = 1$, $S(3) = 1$, $S(4) = 1$, $S(5) = 1$, $S(6) = 1$, $S(7) = 2$, and so on. Ideally, the resource supply should be uniformly distributed as the dash line which is equal to the availability factor times the duration as $\frac{1}{4} \cdot t$. However, resource can only be allocated to an application exclusively in units of resource slices. For this reason, the actual resource supply will be a staircase function $S(t)$ as shown using the solid line. The instant regularity at time t quantifies the gap between the ideal supply and actual supply at time t such as $I(1)$ and $I(6)$. Figure 4b illustrates the actual resource supply for time interval $[1, t)$ as $S(t) - S(1)$. $I(t) - I(1)$ is the supply deviation in this time interval. For example, $I(6) - I(1)$ is the supply deviation in time interval $[1, 6)$. The supply regularity defines the maximum supply deviation for all time intervals.

Definition 3.9 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.

As an example shown in Fig. 3, the availability factor α of the 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 thus P is a regular partition.

3.3 RRP scheduling algorithms

Several algorithms have been developed to construct the schedule of regular and irregular resource partitions based on specified availability factors and supply regularities. For uniform single-resource environment, the Adjusted Availability Factor (AAF) algorithm allocates resource partitions with availability factors of power of $\frac{1}{2}$ to each application (Feng 2004). By limiting the choice of availability factors, the schedule can be easily constructed if the sum of the availability factors is less than 1. This however introduces some resource utilization overhead. Under the RRP model, resources are provisioned with the availability factor restricted to be power of $\frac{1}{2}$ and the scheduler is not work-conserving. The scheduler will construct resource partition with the closest fraction of resource in power of $\frac{1}{2}$ and the unused resource in that partition will not be distributed to other partitions. For example, an application requesting a fraction of $\frac{2}{5}$ resource will be allocated with $\frac{1}{2}$ fraction of the resource. Those extra allocated resource cannot be utilized by other applications. For the uniform multi-resource environment, the use of a combination of Magic7, PFair algorithms (Li and Cheng 2012; Baruah et al. 1996) and various forms of availability factors were proposed to construct the runtime schedule and greatly improve the resource utilization overhead (Li and Cheng 2012, 2017). 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 (Chen et al. 2017). All these algorithms were designed for static resource partition construction. Moreover, an RRP scheduling algorithm is generally performed based on physical resource time t but it can be based on partition resource time t' with some modification to achieve hierarchical re-partitioning (Chen et al. 2017).

4 Resource reconfigurability in RRP model

Different from the aforementioned work on static resource partitioning under the RRP model, this paper studies the resource partition reconfigurability problem in dynamic environment. In Sect. 4.1, we first present the main challenges of maintaining regularity-based resource partition in the run time. We then define the performance semantics for online resource partition reconfiguration and present the formal definition of the dynamic partition reconfiguration (DPR) problem in Sect. 4.2. A novel three-stage algorithm for constructing the resource partitions during and after the reconfiguration is presented in Sect. 4.3 for uniform environment and its properties including the correctness of the algorithm are discussed in Sect. 4.4.

4.1 Challenges

In the RRP model, there may exist multiple applications running on the physical resources and each application may request to reconfigure its resource partitions on demand. In the uniform environment, 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. 5, the application can request to reconfigure its resource supply curve by issuing an R^3P to change the availability factor from $\frac{1}{4}$ to 1 at time t_1. The system then enters the *Resource Partition Transition (RPT)* stage where resource partitions are being reconfigured and performance degradation may happen during this stage as shown in Fig. 5. After the RPT stage is over at time

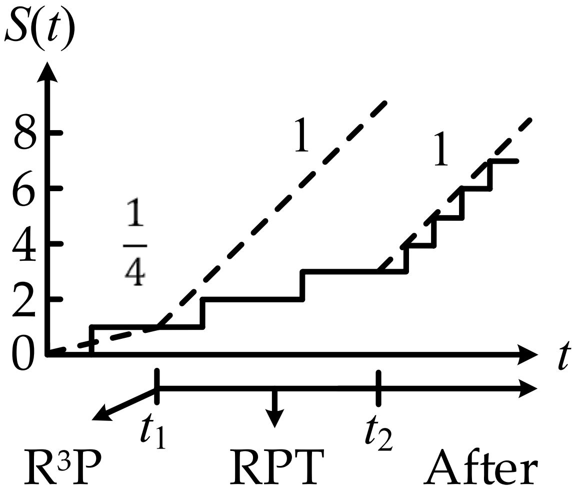


Fig. 5 The dotted line illustrates that the requested availability factor changes from $\frac{1}{4}$ to 1 at the time of R^{3}P and the partition has an availability factor of 1 after the RPT stage

t_2, the reconfigured resource partitions will supply resource to the 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. 5.

There are multiple challenges to handle 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 may violate the performance guarantee of some resource partitions and 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 if the temporary resource supply deviation is larger than the requested regularity. An example is shown in Fig. 6a where a new regular resource partition P_3 requests to join the system at time 5. P_3 should be deemed as a regular partition starting at time 5. However, this request may cause a performance violation to either of the other two resource partitions. To fulfill the performance requirement of P_3, there are only two options to schedule P_3's partitions, as depicted in Fig. 6b, 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 owing to the conflict. One can naively reschedule the resource to accommodate this change such as using the AAF or Magic7 algorithm, as described in Sect. 3.3 to compute a completely new schedule and switch to this schedule at time 5. However, naively rescheduling resource may cause some resource partitions to suffer serious performance degradation and the violation of supply regularity. Figure 6d illustrates such case, where a new schedule computed by using AAF algorithm is adopted and P_2 will suffer a serious

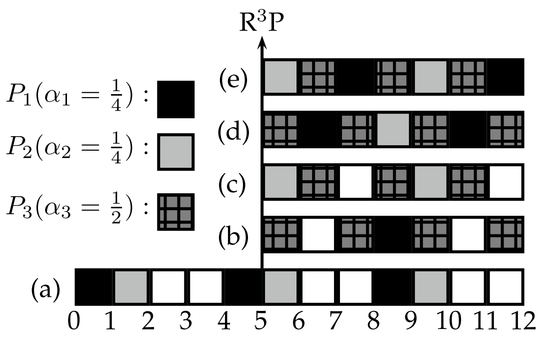

R^{3}P
$P_1(\alpha_1 = \frac{1}{4})$:
$P_2(\alpha_2 = \frac{1}{4})$:
$P_3(\alpha_3 = \frac{1}{2})$:
(e)
(d)
(c)
(b)
(a)
0 1 2 3 4 5 6 7 8 9 10 11 12

Fig. 6 There is an R^3P at time 5 requesting to add a new resource partition P_3 into the system. **a** Shows the schedule without R^3P. **b** and **c** Show that the R^3P will cause P_3 to conflict with either P_1 or P_2, respectively. **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

starvation interval during time 2 to 8, while a carefully designed schedule will be able to minimize the performance degradation, as shown in Fig. 6e.

As a summary, the challenges of handling reconfiguration requests of resource partitions in uniform environment include: (1) how to define the performance semantics during the transition among old/new and RPT stages; and (2) how to construct the schedule during and after the transition to satisfy the performance requirement of each reconfiguration request.

4.2 Dynamic partition reconfiguration problem

To address the aforementioned challenges, we now extend the RRP model to take the online partition reconfiguration into consideration and define the semantics of performance guarantee during the reconfiguration. The formal definition of the dynamic partition reconfiguration problem will be presented at the end of this subsection. The key ideas of the proposed algorithms will be presented in Sect. 4.3.

Recall that a resource partition P is a tuple $(\mathcal{S}, p)$ which describes its cyclic schedule and its period. In each stage, we can describe the schedule of the resource partition using this tuple with the time zero counting from the start of the stage. We thus describe the resource partition P at different stages using different symbols. We denote the resource partition in the old stage as P^o (before reconfiguration), in the transition stage as P^t (during RPT stage) and in the new stage as P^n (after reconfiguration). We now formally define the reconfiguration request of resource partition.

Definition 4.1 Reconfiguration Request of Resource Partition (R^{3}P) is defined as a tuple $\lambda = \{\mathcal{P}^n, \mathcal{A}, \mathcal{R}^r, T\}$ where $\mathcal{P}^n$ is the target set of resource partitions after the request; each resource partition $P_i^n \in \mathcal{P}^n$ has an associated availability factor of $\alpha_i^n \in \mathcal{A}$ and P_i will have reconfiguration supply regularity (see Definition 4.3) of $R_i^r \in \mathcal{R}^r$. T is the maximum time allowed for the reconfiguration to complete.

While P^o and P^n represent the partition before and after the reconfiguration, P represents the partition over the entire time interval including those time intervals across the reconfiguration. The resource supply after the reconfiguration is guaranteed by enforcing the availability factor and regularity of P^n_i. The performance semantic for the reconfiguration is achieved by enforcing the resource supply and the supply deviation of P for any time interval including the transition stage, which will be defined later. We now classify a resource partition P during a reconfiguration into the following four categories.

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

Deleted Partition P^n 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^n have the same availability factor (larger than 0) and the same supply regularity.

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

In this paper, the performance semantic defines the maximum difference between the actual resource supply and the desired supply even during the reconfiguration. We use reconfiguration supply regularity to formally define such performance semantics. We now first extend the definition of instant regularity to accommodate the change of availability factor. The desired fraction of resource is α^o before the reconfiguration and α^n after the reconfiguration. The instant regularity is thus defined as follows.

Definition 4.2 The instant regularity $I(t)$ of a resource partition P at time $t \geq t_r$ is defined as $I(t) = S(t) - (\alpha^o \cdot t_r + \alpha^n(t - t_r))$ where t_r is the time of a R^3P; α^o and α^n are the availability factors of the resource partition P before and after the request, respectively.

As an example shown in Fig. 7a, $I(1)$ indicates that there is resource over supply at time 1 while $I(8)$ indicates that there is resource under supply at time 8. The ideal amount of resource supply which is $\frac{1}{4}$ in the time interval [0, 4) and 1 after time 4. The supply function $S(t)$ satisfies that $S(1) = 1$ and $S(8) = 4$. Thus, based on definition 4.2, the instant regularity at time 1 and 8 are $I(1) = \frac{3}{4}$ and $I(8) = -1$, respectively. Also, as shown in Fig. 7b $S(b) - S(a)$ denotes the actual resource supply during the time interval $[b, a)$ 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 definition of instant regularity, we now define the *reconfiguration supply regularity* as follows.

Definition 4.3 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 $I(b) - I(a) > -k, \forall b \geq a$.

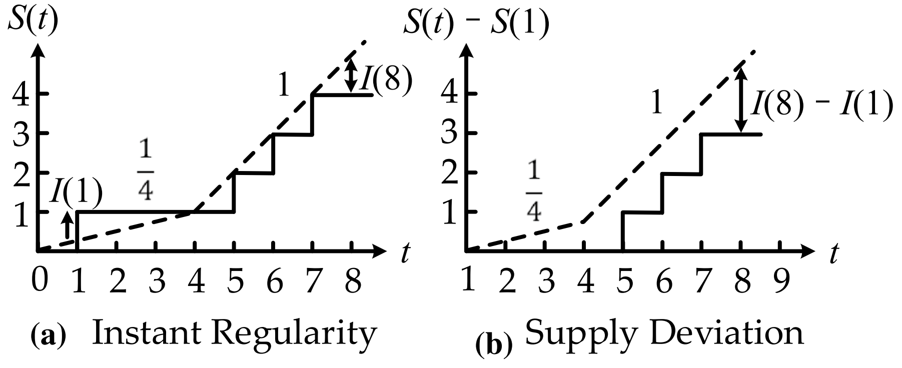


Fig. 7 Dotted line shows the ideal amount of resource supply which is $\frac{1}{4}$ in [0, 4) and 1 after time 4. $I(1) = \frac{3}{4}$ and $I(8) = -1$ in **a** illustrate the instant regularity at time 1 and 8, respectively. $I(t) - I(1)$ in **b** illustrates the deviation of resource supply for time interval $[1, t)$

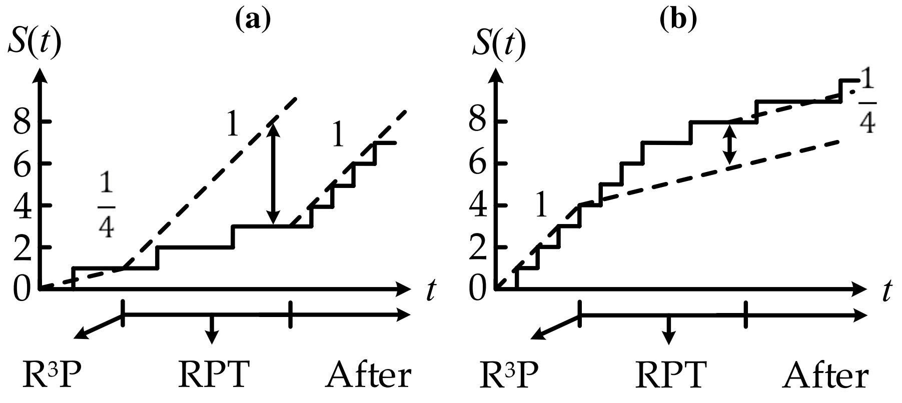


Fig. 8 The dotted line illustrates that the requested availability factor changes from $\frac{1}{4}$ to 1 and from 1 to $\frac{1}{4}$ in **a**, **b**, respectively, at the time of R^{3}P. The arrow shows the supply deviation during the reconfiguration where there is resource supply shortfall in (**a**) and resource supply surplus in (**b**), respectively

The reconfiguration supply regularity only defines the maximum supply shortfall while the normal supply regularity restricts *both* the maximum supply shortfall and supply surplus. This relaxation provides more flexibility when resolving the schedule conflicts during the reconfiguration while still restricting the maximum resource supply shortfall. Based on this definition, the semantics of the performance guarantee for a resource partition during the reconfiguration can be illustrated in Fig. 8. In Fig. 8a, b, the resource supply deviation before/after the RPT is illustrated as the gap between the actual supply and the ideal supply. It is bounded by the normal supply regularity. On the other hand, the resource supply deviation during the RPT is bounded by its reconfiguration regularity R^r. In Fig. 8a, the resource supply suffers a performance degradation for any time interval overlapped with the RPT and the supply shortfall shall be bounded by R^r. On the other hand, the resource supply has a supply surplus for any time interval overlapped with the RPT in Fig. 8b. A reconfiguration regular partition P supplies the resource no less than the requested fraction of resource. To illustrate the concept, we give a numerical example in Fig. 6, where we construct two partition schedules as shown in Fig. 6d, e, respectively. In Fig. 6d, the maximum supply shortfall happens in time interval [2, 8) for P_2 which is $I_2(t_1) - I_2(t_2) = S_2(t_1) - S_2(t_2) - \alpha_2(t_1 - t_2) = -\frac{3}{2} < -1$ and this extra supply shortfall makes P_2 not reconfiguration regular. In contrast, in Fig. 6e, P_1 has supply surplus while the schedule of P_2 is unchanged. This makes P_1, P_2 and P_3 all reconfiguration regular even if there originally exists a schedule conflict between P_3 and one of P_1 or P_2.

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

– No concurrent reconfiguration request is allowed in the system.
– For each resource partition P, P^o and P^n are both regular but P 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 the power of $\frac{1}{2}$.

Problem 4.1 Dynamic Partition Reconfiguration (DPR): Given a reconfiguration request $\lambda = \{\mathcal{P}^n, \mathcal{A}, \mathcal{R}^r, T\}$ and the resource partitions before the request $\{P_i^o \mid \exists P_i^n \in \mathcal{P}^n\}$, compute the schedules of P_i^t and P_i^n for each resource partition $\{P_i \mid \exists P_i^n \in \mathcal{P}^n\}$ such that the following three conditions are satisfied:

C-1: P_i^n is a regular partition with availability factor of α_i^n;
C-2: the reconfiguration regularity of P_i is less than 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 reconfigured 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.

4.3 Three-stage DPR algorithm

In this section, we present the three-stage algorithm to solve the DPR problem. In uniform environment, the reconfiguration of resource partitions on different physical resources can be performed independently as long as the physical resource time of every resource is synchronized. This is because an application is assumed to request resource and finish execution at resource slice boundaries. The key challenge to solve this problem is to ensure that each resource partition is reconfigured 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

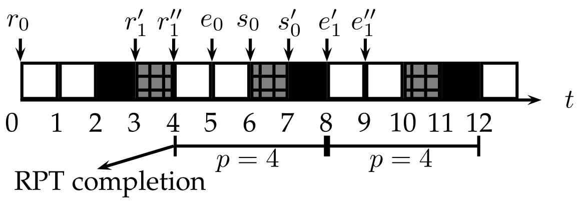


Fig. 9 An example of the task scheduling system: its first instance starts at time r_0 with deadline e_0. Given different completion times, the starting time and deadline of the next instance is computed accordingly. The schedule of the task is cyclic with period of p after the RPT stage. Please note that the resource partition may have resource slice offset s_0 or s_0' after the RPT stage

this purpose, every time a resource slice is allocated to a resource partition, the next resource slice to be allocated to this resource partition must satisfy the performance requirements as specified by the conditions **C-1** and **C-2**, which together impose a deadline for allocating the next resource slice to the resource partition. The problem of scheduling resource partitions is akin to scheduling a set of tasks. where each partition can be considered as a task and the followings need to be satisfied: (1) a task instance is immediately released upon the completion of its previous instance, (2) the deadline of the new instance depends on the availability factors and maximum supply shortfall (to be defined later) of its associated partition, and (3) each task follows a cyclic schedule after the RPT stage. Figure 9 depicts an example of such task scheduling system. To simplify the model, we assume that the task has a fixed relative deadline as 5 and 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, it 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 by following a cyclic schedule with a period of 4 as illustrated in Fig. 9. Notice that each scheduled task instance will map to the resource slice offset of the resource partition. The resource partition in Fig. 9 may have resource slice offset s_0 or s_0' after the reconfiguration.

We use $\mathcal{T}^t$ to denote the state of such partition scheduling system at time t, which includes the starting time r_i, the maximum supply shortfall d_i and the deadline e_i. The maximum supply shortfall d_i is defined as follows:

Definition 4.4 The maximum supply shortfall of a resource partition P at time t 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. 7b, $I(t) - I(b)$ indicates the supply deviation of partition P in time interval $[b, t)$, and $d(t)$ defines the maximum supply shortfall for any time interval $[b, t), b \leq t$. For the starting time, it will be the completion time of the last scheduled slice as if a new instance of task is released upon the completion of the last task as illustrated in Fig. 9. The deadline is a function of the maximum supply shortfall.

Algorithm 1: Algorithm Overview for the DPR Problem

```
Input: The R^3P $\lambda$ and the R^3P requesting time $t_r$.
Output: Transition schedule $\mathcal{S}_i^t$ and cyclic schedule $\mathcal{S}_i^n$ for all $i$ s.t. $P_i^n \in \mathcal{P}^n$.
        Reject if no feasible schedule.
1  $\mathcal{T}^0$ = Initialization($\lambda, t_r$) // Stage-1
2  for $t_b \leftarrow 0$ to $T$ do
3      ($\{\mathcal{S}_i^t \mid \forall i\}, \mathcal{T}^{t_b}$) = TransitionSchedule($\mathcal{T}^0, t_b$) // Stage-2
4      if $\mathcal{T}^{t_b} \neq Null$ then
5          $\{\mathcal{S}_i^n \mid \forall i\}$ = CyclicSchedule($\mathcal{T}^{t_b}$) // Stage-3
6      end
7      if $\{\mathcal{S}_i^n \mid \forall i\} \neq Null$ then
8          return ($\{\mathcal{S}_i^t | \forall i\}, \{\mathcal{S}_i^n | \forall i\}$)
9      end
10 end
11 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 based on above mentioned partition scheduling system. We propose a *three-stage algorithm* to break down the DPR problem into three sub-problems. An overview of the algorithm is presented in Algorithm 1. **Stage-1** of the algorithm initialize the state of the partition system (Algorithm 2). In **Stage-2**, the algorithm searches for a feasible solution with an RPT duration of $t_b \leq T$ and constructs the transition schedule for each P_i^t within that duration (Algorithm 3). Based on the state information of the partition system at the end of the RPT stage, **Stage-3** computes the cyclic schedules for individual P_i^n to meet their corresponding supply regularity and availability factor requirements (Algorithm 4). The correctness of the algorithm is proved in Theorem 4.3.

We now present the details of the proposed algorithm.

Algorithm 2: Partition System Initialization

```
Input: The R$^3$P $\lambda$ and the R$^3$P requesting time $t_r$
Output: $\mathcal{T}^0$, the state of the partition system at the time reconfiguration.
Procedure Initialization($\lambda, t_r$)
    for $i \in \{j \mid \exists P_j^n \in \mathcal{P}^n\}$ do
        $d_i = 0$
        if $P_i$ is a reconfigured or unchanged partition then
            if $t_r \le s_i^o$ then
                $d_i = -\alpha_i^o \cdot t_r$
            else
                $t_1 = t_r \mod p_i^o$
                if $t_1 \le s_i^o$ then
                    $t_1 = t_1 + p_i^o$
                end
                $d_i = \alpha_i^o(s_i^o + 1 - t_1)$
            end
        end
        $r_i = 0$; $e_i = \lfloor (R_i^r + d_i)/\alpha_i^n \rfloor$;
    end
    return $\mathcal{T}^0$
```

4.3.1 Stage 1: partition system initialization

Algorithm 2 presents the algorithm details for initializing the state of the partition system $\mathcal{T}^0$. A key step in the algorithms is to compute the maximum supply shortfall d_i and the deadline e_i. The following theorem shows how d_i can be computed.

Theorem 4.1 *Let P_i^o be the resource partition representation of the resource partition P_i before the reconfiguration request time t_r and $t_r > s_i^o$ where s_i^o is the resource slice offset of P_i^o, the maximum supply shortfall of the resource partition P_i 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*

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

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 Definition 3.7) and the fact that $\frac{1}{p_i^o} = \alpha_i^o$, we have

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

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

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

By the definition of the maximum supply shortfall, Eqs. (1), (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. □

Algorithm 2 initializes the maximum supply shortfall d_i and deadline e_i at the reconfiguration request time t_r of each resource partition P_i. For Line 5–6, it is a special case where the resource partition has not yet offered any resource so the supply shortfall is $-\alpha_i^o \cdot t_r - 0$ where α_i^o is the availability factor of P_i^o. Note that s_i^o is the only resource slice offset of P_i^o. For Line 7–12, it computes $d_i(t_r)$ for partition P_i according to Theorem 4.1. The starting time and deadline of the first instance of τ_i is set as $r_i = 0$ and $e_i = \lfloor (R_i^r + d_i)/\alpha_i^n \rfloor$, respectively.

4.3.2 Stage 2: transition schedule computation

Algorithm 3: Transition Schedule Computation

```
Input: The R^3P $\lambda$, the time budget $t_b$ and $\mathcal{T}^0$, the state of the partition
       scheduling system at the time of R^3P
Output: Transition schedule $\{\mathcal{S}_i^t \mid \forall i\}$ and the state of the partition system at the
        end of the RPT stage $\mathcal{T}^{t_b}$. Reject if no feasible schedule is found.
Procedure TransitionSchedule($\mathcal{T}^0, t_b$)
    for $t_t \leftarrow 0$ to $t_b$ do
        $m[t_t] = 0$ //initialize the data structure for schedules
    end
    Enqueue all partition $P_i^t$ into a queue $Q$ in the ascending order following (1)
      deadline $e_i$ and (2) period $p_i^n$
    while $Q \neq \emptyset$ do
        Dequeue $P_i^t$ from $Q$
        $l$ = DS-EDF($r_i, e_i, m, t_b$)
        if $l = NULL$ then
            if $e_i \leq t_b$ then
                return NULL // Deadline will miss
            end
            //no idle resource slice can be utilized, update the system state
            $r_i = 0$
            $e_i = e_i - t_b$
        else
            Add $l$ to $\mathcal{S}_i^t$
            $d_i$ = UpdateShortfall($d_i, l, r_i$)
            $r_i = l + 1$
            $e_i = \lfloor (R_i^r + d_i)/\alpha_i^n \rfloor + l + 1$
            Enqueue $P_i^t$ to $Q$
        end
    end
    return ($\{\mathcal{S}_i^t \mid \forall i\}, \mathcal{T}^{t_b}$)
Procedure DS-EDF($r, e, m, t_b$)
    if $e > t_b$ then
        $e = t_b$
    end
    for $t_t \leftarrow e - 1$ to $r$ do
        if $m[t_t] = 0$ then
            $m[t_t] = 1$
            return $t_t$
        end
    end
    return NULL
Procedure UpdateShortfall($d_i, l, r_i$)
    return $\min(0, d_i + 1 - \alpha_i^n(l + 1 - r_i))$
```

Given a partition system computed in Stage 1 and with a time budget t_b to complete the reconfiguration, this stage computes the transition schedule for P_i^t by following two heuristic principles: (1) we employ the *deferrable scheduling (DS)-EDF* algorithm Han et al. (2012) where partitions are scheduled according to their earliest deadlines but each partition is scheduled as late as possible to make room for other

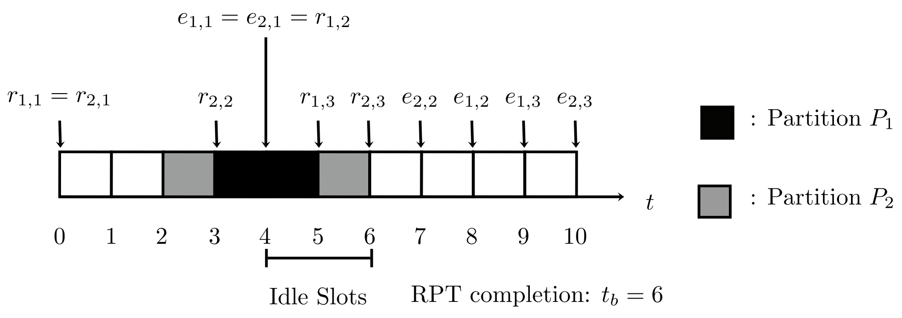


Fig. 10 An example of the transition schedule computation

partitions during the RPT stage; and (2) if the deadline of a partition 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 its next deadline can be further deferred when entering Stage 3. This will significantly increase the schedulibility of the cyclic schedule construction in Stage 3. In the following, we first give an example, and then present the algorithm details.

Figure 10 gives an example to illustrate the two heuristic principles to schedule the two partitions with a reconfiguration length of 6. We use $r_{i,j}$ and $e_{i,j}$ to denote the j-th starting time and relative deadline of partition P_i, respectively. Each partition has a relative deadline of 4. At the beginning, the two partitions P_1 and P_2 have starting time $r_{1,1} = r_{2,1} = 0$ and deadline $e_{1,1} = e_{2,1} = 4$. The algorithm schedules partition with the earliest deadline (ties will be broken arbitrarily) and picks a latest unassigned resource slice between the starting time and deadline of the partition to be scheduled by following principle (1). Hence, in this example, partition P_1 is assigned a resource slice at time 3 first. The starting time and deadline of partition P_1 is then updated as $r_{1,2} = 4$ and $e_{1,2} = 8$, respectively. Next, partition P_2 is picked to be scheduled because now it has the earliest deadline 4 and it is assigned an unassigned resource slice at time 2. The starting time and deadline of partition P_2 is updated to $r_{2,2} = 3$ and $e_{2,2} = 7$, respectively. In the next steps, the derived deadlines of all the partitions are 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). Partition P_2 is first scheduled, due to its earlier deadline $e_{2,2} = 7$, at time 5 and its deadline is further deferred to $e_{2,3} = 10$; partition P_1 is then scheduled at time 4 and its deadline is updated to $e_{1,3} = 9$.

We then present the details of Algorithm 3. In the following, we first present how the maximum supply shortfall can be updated at the time of resource slice assignment.

The maximum supply shortfall of each resource partition will decrease linearly and the deadline will not change during the time interval it is not allocated with any resource slice. We thus only need to update the maximum supply shortfall d_i and deadline e_i of the resource partition after it is scheduled a resource slice. The following theorem shows how to update d_i after the resource partition is allocated a resource slice.

Theorem 4.2 *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.*

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

Proof According to Definition 4.4, there must exist a $b' \leq t$ such that $I(b') \geq I(b)\ \forall b \leq t$ and thus $d_i(t) = I(t) - I(b')$ for some 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, Definition 4.4 and the fact that there is exactly one resource slice scheduled at time $t + \delta - 1$ in the interval $[t, t + \delta)$, we have

$$d_i(t + \delta) = S(t + \delta) - \alpha_i^o \cdot t_r - \alpha_i^n(t + \delta - t_r) - I(b') \tag{4}$$

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

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

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. $\square$

Algorithm 3 summarizes the procedure for computing the transition schedule. The procedure constructs the transition schedule by (1) scheduling partitions as late as possible and (2) utilizing the idle slice before the time budget is used up. For the loop in Line 5–23, the partition is scheduled according to the earliest deadline. For each partition P_i^t, the DS-EDF procedure takes its starting time r_i, deadline e_i, the current schedule m, the maximum duration of the schedule t_b and assigns an resource slice to P_i^t. The current schedule m records the owner of each resource slice at time t with $m[t]$ and $m[t] = 0$ indicates that the resource slice starting at t is unassigned as in Line 30. This procedure finds an latest available resource slice at the time between the starting time and the deadline of P_i^t as in Line 29–34. The resource partition P_i^t is added back to the queue with the maximum supply shortfall based on Theorem 4.2 once assigned as in Line 17–21. If there is no available resource slice for resource partition P_i^t in the RPT stage and its deadline is before the end of the RPT stage, a deadline miss will happen and the algorithm simple rejects as in Line 9–12. If all partitions 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 partition system and enters Stage 3 (Line 13–15). Notice that the period of a regular partition P_i^n with availability factor α_i^n will be $p_i^n = \alpha_i^n$.

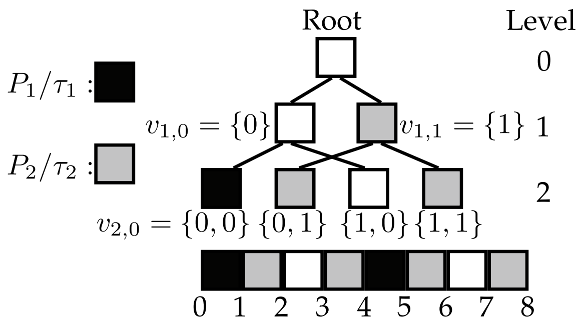


Fig. 11 The schedules can be encoded as a tree where each resource partition is assigned a sub-tree exclusively

Algorithm 4: Cyclic Schedule Computation

```
Input: $\mathcal{T}^{t_b}$, the sate of the partition system after the RPT.
Output: Cyclic schedule $\{\mathcal{S}_i^n \mid \forall i\}$. Reject is no feasible schedule is found.
Procedure CyclicSchedule($\mathcal{T}^{t_b}$)
    Enqueue all partition $P_i^n$ into a queue $Q$ in the ascending order following (1)
      period $p_i^n$ and (2) deadline $e_i$
    for $t_t \leftarrow 0$ to $p_{max}$ do
        $m[t_t] = 0$ //initialize the data structure for schedules
    end
    while $Q \neq \emptyset$ do
        Dequeue $P_i^n$
        $l =$ DS-EDF$(0, e_i, m, p_i^n)$
        if $l = NULL$ then
            return NULL
        end
        Add $l$ to $\mathcal{S}_i^n$
        for $t_t \leftarrow 0$ to $p_{max}/p_i^n - 1$ do
            $m[l + t_t \times p_i^n] = 1$
        end
    end
    return $\{\mathcal{S}_i^n \mid \forall i\}$
```

4.3.3 Stage 3: cyclic schedule computation

Based on each transition schedule computed from Algorithm 3, which has a length smaller than T, this stage will compute one cyclic schedule $\{\mathcal{S}_i^n \mid \forall i\}$ 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. 11. 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 $v_{i,j}$ denotes the node is j-th node at depth i, $0 < k < 2^i$, $x_k \in \{0, 1\}$, and the root node is indexed as $\{\}$. Each node $v_{i,j}$ has one left node and one right node indexed as $v^l_{i,j} = \{0, x_1, \dots, x_{2^i}\}$ and $v^r_{i,j} = \{1, x_1, \cdots, x_{2^i}\}$, 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. 11, 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 P_i.

Given the state of the partition system at the end of the RPT, Algorithm 4 assigns a node in the $\mathcal{IS}$-tree for each partition using the DS-EDF procedure in Algorithm 3. For a partition with period p^n_i the procedure schedules the partition as late as possible at level x where $2^x = p^n_i$. This procedure repeats until all the tasks have been assigned with an appropriate tree node. As an example in Fig. 11, the DS-EDF procedure will search for an available node at level 1 of the $\mathcal{IS}$-tree for P_2 which has a deadline of 2 and a period of 2. Suppose that $v_{1,1}$ is available and assigned to P_2, Algorithm 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.

4.4 Analyses and properties of the DPR algorithm

This section presents some important analyses and properties of the DPR algorithm, including its time complexity, correctness of the algorithm, completeness, some feasibility analysis, and its support for recursive reconfiguration.

Time complexity We begin with the time complexity analysis of the DPR algorithm. In Stage 1, Algorithm 2 has a complexity of $O(N)$, where N is number of resource partitions as the computation of the maximum supply shortfall is a $O(1)$ operation by Theorem 4.1. In Stage 2, Algorithm 3 has a complexity of $O(N)$ for building up the queue, a complexity of $O((2T + N) \cdot logN)$ for dequeuing and enqueuing $2T + N$ times; and plus the complexity of the DS-EDF procedure which is $O(T^2 + NT)$. In Stage 3, Algorithm 4 has a time complexity of $\sum_i(p^n_i + \frac{p_{max}}{p^n_i})$ to search available nodes for each task at level x with $2^x = p^n_i$ and mark the unav ailable nodes. Algorithm 4 also involves $O(N + NlogN)$ queue operations. This brings the total time complexity of the DPR algorithm to $O(NlogN + NT + TlogN + T^2 + \sum_i(\frac{p_{max}}{p^n_i} + p^n_i)))$.

Correctness The following theorem shows the correctness of the DPR algorithm.

Theorem 4.3 *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, Algorithm 4 computes a schedule for each regular partition P^n_i with its targeted availability factor α^n_i. This satisfies condition **C-1**. In Stage 2, Algorithm 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 (Definition 4.2). Hence, $I(b') - I(a) > -R_i^r$ if $I(b) - I(a) > -R_i^r$. Furthermore, by the definition of maximum supply shortfall (Definition 4.4), 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 the 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 the first resource slice after the time of the R^{3}P request t_r. By the definition of maximum supply shortfall (Definition 4.4) 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 Definition 4.2 and $S(t_r) = S(b_0)$, we have $I(b_0) = I(t_r) - \alpha_i^n(b_0 - t_r)$. 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 (Definition 4.4) and $S(t_r) = S(b_0)$. It follows that $d_i(b_0) = I(t_r) - \alpha_i^n(b_0 - t_r) - I(x)$. Furthermore by Eq. (6), we have

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

According to the computation of deadline in Algorithm 2, we have $b_0 - t_r \leq (R_i^r + d_i(t_r))/\alpha_i^n - 1$. From Eq. (7), we have $d_i(b_0) \geq -R_i^r + \alpha_i^n > -R_i^r$. 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

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

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

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

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

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

We hence have $d_i(b_{k+1}) > -R_i^r$ because $d_i(b_k + 1) > -R_i^r + 1 - \alpha_i^n$ by Theorem 4.2 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. $\square$

Completeness and feasibility analysis The following two theorems prove the completeness of Algorithm 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, respectively.

Theorem 4.4 *Given the state of a partition system $\mathcal{T}^{t_b}$ at the end of the RPT stage, Algorithm 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 Algorithm 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 Algorithm 4. The algorithm schedules partitions according to (1) period p_i and then (2) deadlines e_i in the ascending order; also, the partitions are scheduled as late as possible. Hence, for any partition P_i having schedule encoded as $v_i = v_{j,k}$ and any other node at the same level $v_i' = v_{j,l}$, one of the following conditions must be true: (1) $|v_i'| \geq e_i$; (2) $|v_i'| < |v_i|$; (3) v_i' is assigned to another partition P_i' with $e_i' \leq e_i$. (4) v_i' is a descendent of a parent node assigned to another partition with a period smaller than p_i.

The proof proceeds by transforming any feasible schedule s into a schedule conforming the aforementioned property by adjusting the schedule of each partition P_i according to the following queue. The partitions are sorted according to their (1) period p_i and (2) deadline e_i in an ascending order. For each partition P_i in this queue which is assigned node $v_i = v_{j,k}$, we check whether there is a node $v_i' = v_{j,l}$ at the same level as v_i and invalidating all of the above four conditions. If there exists such a node v_i', we swap the entire sub-tree v_i with sub-tree v_i'. We shall prove that one of the above four conditions will be true for P_i and none of the deadlines will be violated.

If such v_i' exists, it must be true that $|v_i'| < e_i$, $|v_i'| > |v_i|$ and one of the followings is true.

Case 1: The v_i' is not assigned to any partition.

Case 2: v_i' is assigned to another partition P_i' with $e_i' > e_i$.

Case 3: A descendant node of v_i' is assigned to another partition P_i'.

Any partition P_i' assigned on the sub-tree v_i' must either have $p_i = p_i'$ and $e_i' > e_i$ (case 2) or $p_i < p_i'$ (case 3). Because $|v_i| < |v_i'|$, P_i''s schedule will be earlier after the swap and thus this won't cause the deadline miss for P_i'. For example in Fig. 11, partition P_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 partitions, any pair of nodes will conform to one of the four conditions resulting an equivalent schedule computed by Algorithm 4. This completes the proof. □

Theorem 4.5 *An* R^3*P is always feasible if every partition $P_i \in \mathcal{P}^n$ has reconfiguration regularity R_i^r no less than 2.*

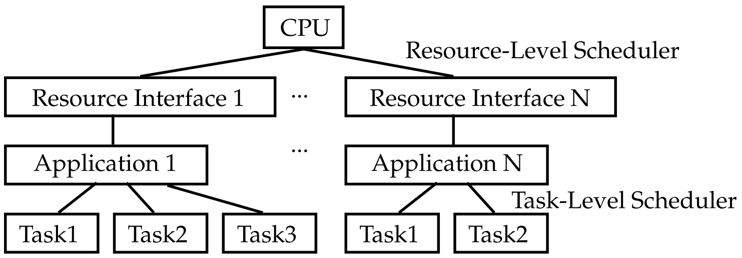


Fig. 12 Normal scheduling hierarchy

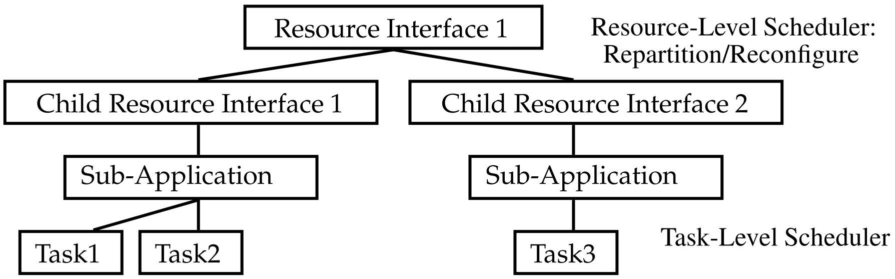


Fig. 13 Scheduling hierarchy for hierarchical repartitioning and recursive reconfiguration

Proof One can simply set the time budget of reconfiguration to zero and perform the DPR algorithm. By Definitions 3.8, 3.9, 4.4 and the fact that resource partition has reconfiguration regularity $R_i^r > 1$, we have $d_i(t_r) > -1$. The deadline of each partition P_i^n will be $e_i = \lfloor (R_i^r + d_i(t_r))/\alpha_i^n \rfloor \geq \frac{1}{\alpha_i^n} = p_i^n$. If $e_i \geq p_i^n$ holds for every partition P_i^n and the total utilization of all partition is no greater than 1, Algorithm 4 can compute the feasible schedule by allocating schedule in the ascending order of the periods. This completes the proof. □

Support of Recursive Reconfiguration: We first review the concept of hierarchical repartitioning (Chen et al. 2017) and introduce the concept of recursive reconfiguration based on the similar characteristics of hierarchical repartitioning.

In these two concepts, the partitioning and reconfiguration algorithms can be performed on a resource partition based on it logical clock instead of on a physical resource. For example, Fig. 12 illustrates a scheduling hierarchy in which a CPU resource is partitioned into several resource interfaces by some partitioning algorithms (Li and Cheng 2012; Chen et al. 2017). As the same algorithm applied here, a resource-level scheduler can repartition or reconfigure the Resource Interface to construct or reconfigure the Child Resource Interfaces as illustrated in Fig. 13. This can be done recursively in the hierarchy. This also significantly isolates the resource scheduler in the scheduling hierarchy from each other such that each scheduler can independently repartition or reconfigure its resource without the complete

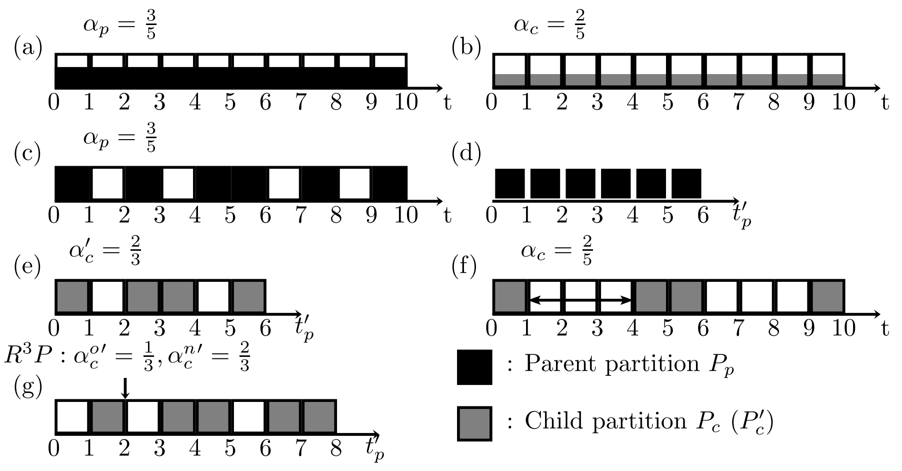

(a)
$\alpha_p = \frac{3}{5}$
(b)
$\alpha_c = \frac{2}{5}$
(c)
$\alpha_p = \frac{3}{5}$
(d)
(e)
$\alpha'_c = \frac{2}{3}$
(f)
$\alpha_c = \frac{2}{5}$
$R^3P : \alpha_c^{o\prime} = \frac{1}{3}, \alpha_c^{n\prime} = \frac{2}{3}$
(g)
0 1 2 3 4 5 6 7 8 9 10 t
t'_p
■ : Parent partition P_p
▨ : Child partition P_c (P'_c)

Fig. 14 Concepts of hierarchical repartitioning and recursive reconfiguration in which repartitioning and reconfiguration algorithms can be performed based on the partition resource time t'_p instead of physical resource time t

knowledge of its parent partition and without the need to alter the schedule governed by other scheduler.

Recall that we have a physical resource time system t and a partition resource time system t'. Here, we denote the parent partition as P_p and its partition resource time as t'_p. Without the loss of generality, we use P_c to denote a child partition of P_p in the physical resource time system and use P'_c to denote the same child partition of P_p in the partition resource time system. P'_c has an availability factor α'_c and a period p'_c.

Figure 14 gives an example to illustrate the concepts of hierarchical repartitioning and recursive reconfiguration. In the RRP model, we can repartition a parent resource partition P_p with $\alpha_p = \frac{3}{5}$ as illustrated in Fig. 14a to create a child partition P_c with $\alpha_c = \frac{2}{5}$ as illustrated in Fig. 14b by allocating exactly $\frac{2}{3}$ fraction of resource supply of P_p. In reality, the resource-level scheduler may perform the partitioning algorithm to create a schedule as illustrated in Fig. 14c and P_p can be considered as if it is a physical resource which has sequential resource slices to offer in the partition resource time system as illustrated in Fig. 14d. The application-level scheduler may then perform the same partitioning algorithm based on the partition resource time to construct a child partition P'_c with availability of $\alpha'_c = \frac{2}{3}$ as illustrated by Fig. 14e. In the physical resource time system, P_c has a schedule as illustrated in Fig. 14f and P_c has an availability factor $\alpha_c = \alpha_p \cdot \alpha'_c = \frac{2}{5}$. Similarly, we can also reconfigure the child partition P_c/P'_c by performing the reconfiguration algorithm based on its parent's partition resource time. In Fig. 14g, P'_c reconfigures its availability factor from $\alpha_c^{o\prime} = \frac{1}{3}$ to $\alpha_c^{n\prime} = \frac{2}{3}$.

However, there is still one issue to pay attention when performing hierarchical repartitioning and recursive reconfiguration based on the partition resource time system. The child partition P'_c may be regular in the partition resource time system but P_c may not be regular in the physical resource time system, which is what the application actually cares about. As illustrated in Fig. 14e, f, P_c' is regular in (e) but P_c is not regular in (f). There is a significant supply shortfall as indicated by the arrow in Fig. 14f which makes P_c not regular.

In the following, we will first quantify the supply deviation of P_c in physical resource time system given P_c' in partition resource time system. Let $S_p(t), \alpha_p, I_p(t)$ be the supply function, availability factor and instant regularity of P_p in the physical resource time system; $S_c(t), \alpha_c, I_c(t)$ be those of P_c in physical time system; $S_c(t)', \alpha_c', I_c(t)'$ be those of P_c' in parent partition time system; and P_c is partitioned from P_p such that $\alpha_c = \alpha_p \cdot \alpha_c'$.

Theorem 4.6 *The resource supply deviation of P_c can be computed as $I_c(a) - I_c(b) = \alpha_c'(I_p(a) - I_p(b)) + I_c'(a') - I_c'(b')$ for all $a \geq b$; and their corresponding partition resource time a' and b'.*

Proof By Definition 3.7, for any physical resource time a and b ($a \geq b$) and their corresponding partition resource time a' and b' we have

$$S_p(a) - S_p(b) = I_p(a) - I_p(b) + \alpha_p(a - b) \tag{9}$$

and

$$S_c'(a') - S_c'(b') = I_c'(a') - I_c'(b') + \alpha_c'(a' - b') \tag{10}$$

By Definition 3.5, the partition resource time advances by one when the parent resource partition offers a resource slice, which means $a' - b' = S_p(a) - S_p(b)$. By substituting $a' - b'$ in Eq. (10) with Eq. (9), we have

$$\begin{aligned} S_c'(a') - S_c'(b') = I_c'(a') - I_c'(b') + \alpha_c'(I_p(a) - I_p(b) \\ + \alpha_p(a - b)) \end{aligned} \tag{11}$$

Also, $S_c'(a') = S_c(a)$ and $S_c'(b') = S_c(b)$ because the numbers of resource slices assigned to P_c and P_c' are the same in both time systems. Combine this fact and Definition 3.7 with Eq. (11), we have $I_c(a) - I_c(b) = \alpha_c'(I_p(a) - I_p(b)) + I_c'(a') - I_c'(b')$ □

Theorem 4.6 tells us that the parent partition P_p may contribute extra supply deviation in the amount of $\alpha_c'(I_p(a) - I_p(b))$ at most when doing hierarchical repartitioning. Based on this result, we have the following two theorems.

Theorem 4.7 *The supply regularity of the child partition P_c, R_c, is less than or equal to the smallest positive integer k such that $k \geq \alpha_c' \cdot R_p + R_c'$ where R_p and R_c' are the supply regularity of P_p and P_c', respectively.*

Proof By Theorem 4.6, we have $I_c(a) - I_c(b) = \alpha_c'(I_p(a) - I_p(b)) + I_c'(a') - I_c'(b')$. We also have $|I_p(a) - I_p(b)| < R_p$ and $|I_c'(a') - I_c'(b')| < R_c'$ because P_p and P_c' have supply regularity of R_p and R_c', respectively. Hence, $|I_c(a) - I_c(b)| < \alpha_c' R_p + R_c'$. □

We also have a similar theorem for reconfiguration supply regularity.

Theorem 4.8 *The reconfiguration supply regularity of the child partition P_c, R_c^r, is less than or equal to the smallest positive integer k such that $k \geq \alpha_c' \cdot R_p^r + R_c^{r'}$ if P_c is also reconfigured or $k \geq \alpha_c' \cdot R_p + R_c^{r'}$ if P_p is not under reconfiguration.*

Proof The proof is similar to the proof of Theorem 4.7 by finding the range of $I_c(a) - I_c(b)$. □

The above two theorems give bounds on the supply regularity and reconfiguration supply regularity, respectively. However, the bound can be improved if the application-level scheduler either has knowledge of the term $\alpha_c'(I_p(a) - I_p(b))$ or has the control over the $I_c'(a') - I_c'(b')$ term. For example, if α_c' has the form of one over some integer, and P_p and P_c' both are regular, then P_c must also be regular (Chen et al. 2017). Moreover, when computing the maximum supply shortfall of each child partition in the DPR algorithm, the system can add one extra term $-\alpha_c' \cdot R_p$ to the computation of maximum supply shortfall to each child resource partition (Line 12 in Algorithm 2 and Line 18 in Algorithm 3) to compensate the extra supply deviation from P_p. This will enable the recursive reconfiguration without the knowledge of other partitions or alter the schedule governed by other schedulers.

5 Performance evaluation

In this section, we provide a comprehensive experimental evaluation on the performance of the DPR algorithm. The simulation results are presented in Sect. 5.1. We also applied the RRP-based dynamic resource reconfigurability model to a real-life autonomous control system and demonstrate its effectiveness in Sect. 5.2.

5.1 Simulation-based experiments

In this section, we first compare the performance of the DPR algorithm with a naive algorithm and an integer linear programming (ILP) based optimal approach. We then present the performance evaluation on the DPR algorithm with different settings. In the experiments, the availability factors of each individual partition before and after R^{3}P are randomly sampled in the set of $\frac{1}{2^i}$ $(1 \leq i \leq 7)$. The reconfiguration supply regularity R_i^r of each resource partition is randomly sampled from [1, 5] and the transition time budget T is randomly sampled from [0, 20]. The number of partitions is randomly sampled from [10, 15]. In our experiments, each parameter is sampled from the given range following the uniform distribution. The experiments to evaluate the performance of the DPR algorithm were conducted with both a small and large parameter range. The results obtained from both experiments showed the similar trend. For the simplicity of presentation and due to the high computation overhead to derive the results from the ILP-based optimal algorithm with the large parameter range, in this paper we only showed the performance comparison among the naive algorithm, DPR algorithm and ILP-based optimal approach with a small range of parameters.

5.1.1 DPR algorithm v.s. the naive algorithm and ILP-based solution

In this subsection, we first describe the naive algorithm and the ILP-based optimal solution, and then compare the performance of the DPR algorithm and these two methods. In the naive algorithm, the system computes a new cyclic schedule using the AAF algorithm at the time of the R^3P based on the information P^n of each new partition. It then changes the schedule to be the new one immediately. The partition system is schedulable if and only if the reconfiguration supply regularity and normal regularity of each resource partition during and after the reconfiguration is not violated in the computed schedule.

The optimal approach finds a feasible solution by brute force search using an integer linear programming solver (Gurobi 2019). Given n resource partitions and let $s_{i,j}$ be the offset of the j-th resource slice of the partition P_i, the solver computes the offsets by the constraints encoded as follows.

As the specific amount of slices for each partition to be scheduled is unknown, we assume that the total amount of the resource slices of each partition P_i is bounded by $T + \frac{p_{max}}{p_i}$. In addition, we employ the binary auxiliary variable $u_{i,j}$ to represent that if the j-th slice of partition τ_i is scheduled in the transition schedule or the cyclic schedule. For instance, if $u_{i,j} = 0$, the corresponding slice of partition must be scheduled in the transition schedule. Otherwise, the corresponding slice of partition must be scheduled in the cyclic schedule.

Auxiliary variable constraints

$$u_{i,j} \in \{0, 1\} \tag{12}$$

where $i \in \{1, 2, ..., n\}$ and $j \in \{1, 2, ..., T + \frac{p_{max}}{p_i}\}$.

In the transition stage, we assume that the maximum amount of the resource slices of each partition P_i is bounded by the time budget T. Based on the partition system, we constrain the offset $s_{i,1}$ by the release time r_i and deadline e_i. For the remaining resource slices of each partition P_i, we constrain their offsets based on the corresponding reconfiguration supply regularity R^r_i.

Transition schedule constraints

$$\begin{aligned} &s_{i,1} \le e_i \\ &s_{i,1} \ge r_i + 1 \\ &s_{i,j} < s_{i,j+1} + M \cdot u_{i,j} \\ &s_{i,j} \ge s_{i,j+1} - p^t_i(1 + R^r_i) - 1 - M \cdot u_{i,j} \\ &s_{i,j} \le s_{i,k} + p^t_i(j - k + R^r_i) + 1 + M \cdot u_{i,j} \\ &s_{i,j} \le T + M * u_{i,j} \end{aligned} \tag{13}$$

where $i \in \{1, 2, ..., n\}$, $j \in \{1, 2, ..., T\}$ and $1 \le k < j$ are integers. Let M be a sufficiently large number. If $u_{i,j} = 1$, then the inequalities hold regardless of the variable

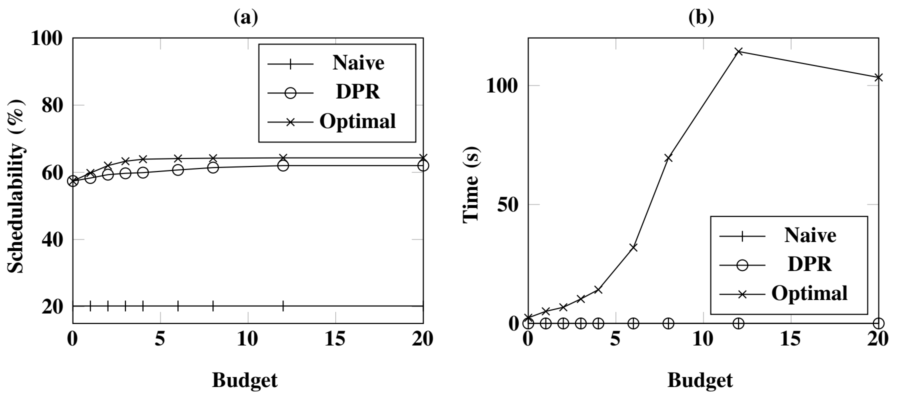


Fig. 15 A comparison among the naive algorithm, DPR algorithm and ILP-based optimal approach

values. This indicates that these constraints do not need to be satisfied if the offsets are not scheduled in the transition schedule. In the cyclic schedule, we employ the period p_i^t to constrain the offsets.

Cyclic schedule constraints

$$\begin{aligned} s_{i,j} &\leq s_{i,j+1} - p_i^t + M(1 - u_{i,j}) \\ s_{i,j} &\geq s_{i,j+1} - p_i^t - M(1 - u_{i,j}) \\ s_{i,j} &> T - M(1 - u_{i,j}) \end{aligned} \tag{14}$$

where $i \in \{1, 2, ..., n\}$ and $j \in \{1, T + \frac{p_{max}}{p_i} - 1\}$. The constraints above need to be satisfied when it holds that $u_{i,j} = 1$.

Since every resource slice of each partition cannot be scheduled in the same time unit, the offsets cannot be equal.

Resource constraints

$$s_{i,j} \neq s_{k,l} \qquad \forall i \neq k \tag{15}$$

where $i, k \in \{1, 2, ..., n\}$ and $j, l \in \{1, T + \frac{p_{max}}{p_i}\}$.

In Fig. 15, we compare the schedulibility of each approach in (a) and computation time in (b). 80% of the resource partitions are set to be reconfiguration regular. The reconfiguration supply regularity of the remaining 20% of the resource partitions are uniformly sampled from [2, 5]. In Fig. 15a, we can see that the DPR algorithm performs comparably to the optimal approach in terms of schedulability while the naive algorithm has very low schedulability. Among all of our testings, the DPR algorithm performs 4% worse than the ILP-based optimal approach at worst. Also, the DPR algorithm outperforms naive algorithm by a huge margin because the naive algorithm does not consider the supply shortfall of each partition at the time of the R^3P. This will incur serious supply shortfall for some of the partitions and violate the performance requirements. In Fig. 15b, we can see that the ILP solver takes 103 seconds on average while the DPR algorithm takes 0.0003 and the naive algorithm

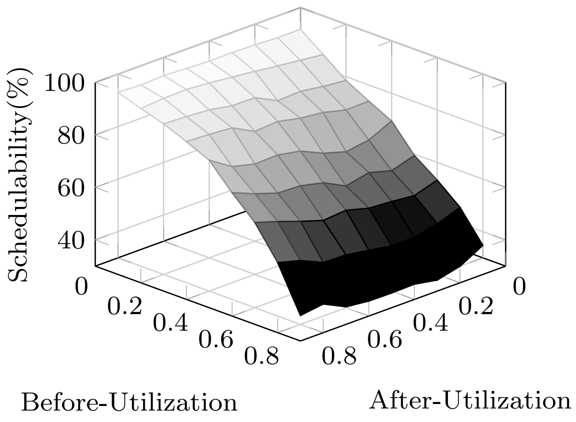


Fig. 16 R^3P schedulability with different utilization settings

takes 0.0001 seconds to compute the schedule for budget equals to 20. Please note that the ILP-based approach is only allowed to search for a valid schedule given the requirement of each R^3P for 600 seconds on a machine with Core(TM) i-5 3.5 GHz CPU. Quite some instances for the ILP-based approach run over 600 seconds and are abandoned. This explains the trend of the ILP-based results after budget 10.

5.1.2 Performance of the DPR algorithm in different settings

We now evaluate the performance of the DPR algorithm under different parameter settings. 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^3P with different partition utilization changes as the general case. From Fig. 16, it can be observed that the schedulability greatly depends on the before-utilization (the total utilization before R^3P) while the after-utilization (the total utilization after R^3P) has little effect. As the before-utilization increases, the schedulability drops significantly. This is because when the before-utilization is higher, partitions are more likely to be blocked from being scheduled by other partitions which has high utilization and supply shortfall. On the other hand, the change of after-utilization has not much impact for the schedulability.

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^3P requests in these experiments. In Fig. 17a, each line represents a different fraction of partitions that have $R_i^r > 1$ and the x-axis denotes the transition budget. Line 20% (55%, 80%, respectively) illustrates the results with 20% (55%, 80%, respectively) resource partitions having $R_i^r > 1$ from the DPR algorithm. In Fig. 18b, each line (budget 0 and budget 12) represents a different budget 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

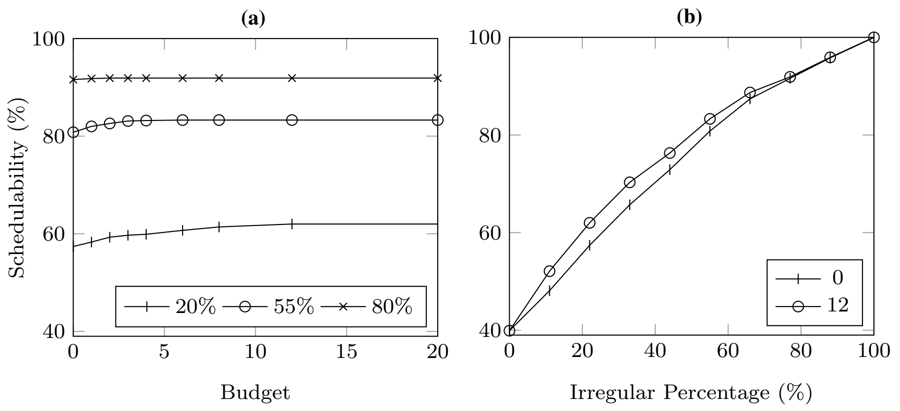
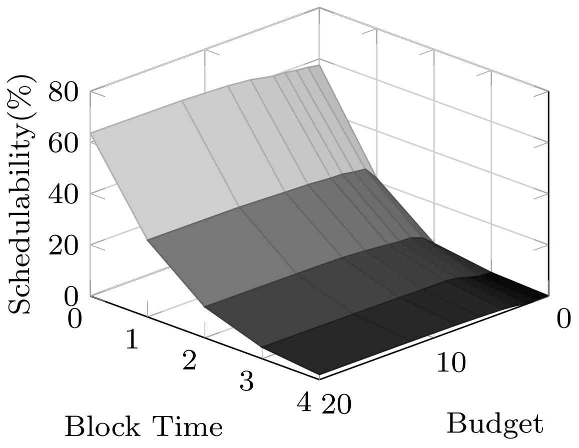


Fig. 17 A comparison of R^{3}P schedulability with varied reconfiguration regularity and budget

Fig. 18 The impact of the reconfiguration operation cost



improve the schedulability but it only works for some extreme cases. Providing more budget does not necessarily improve the schedulability. (2) In Fig. 17b, it can be observed that the increase of R_i^r can significantly improve the schedulability. Moreover, when all partitions have $R_i^r > 1$, the schedulability is 100% as proved in Theorem 4.5.

In the last set of experiments, we create a more practical environment where the cost of the reconfiguration operation is not zero. The operation may include changing the scheduler, setting the schedules and configuring the underlying resources. In this experiment, the reconfiguration operation will block the first few resource slices from being utilized. In Fig. 18, we can see that the block time has significant impact on the schedulability while the budget may alleviate such problem by roughly 10% in some settings. This is because the reconfiguration operation can be considered as a head of line blocking task which has the highest priority and a sizable non-preemptive execution time. If there exists a partition in needs of resource supply, it will be less likely to meet its requirement when the operation cost is high. The budget itself does not offer too much help in this case.

5.2 Case study on the autonomous F1/10 model car

We implemented the RRP resource model and the DPR algorithm 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 algorithm with a naive algorithm.

Our F1/10 autonomous car is built on the Traxxas Slash model car with the following major hardware components (F1tenth 2019) as shown in Fig. 19a: (1) NVIDIA Jetson Tx2 embedded AI computing platform (NVIDIA 2019) 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 (Calandrino et al. 2006; Brandenburg 2011), 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 (ROS 2019) 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 35 m by 8 m 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 1, 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. It also 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. 19b). After the traffic sign is detected, the system enters the Turn Corner context where it slows down and makes the turn (Fig. 19c). 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[2].

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

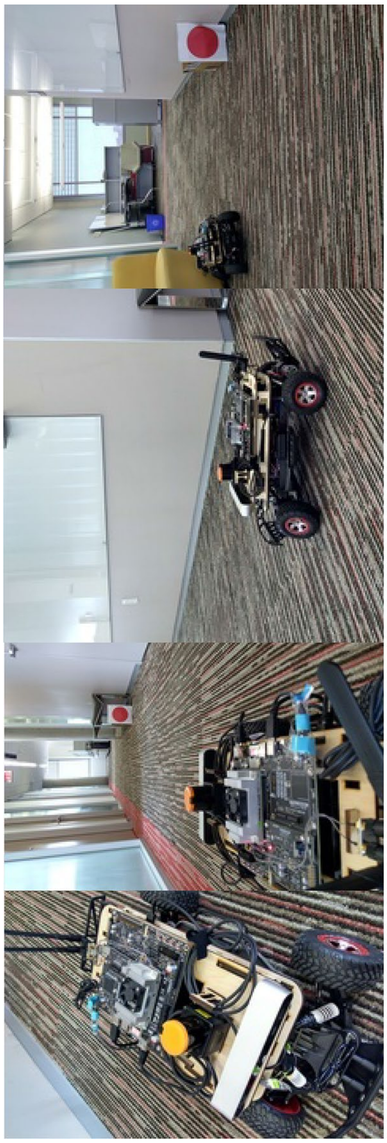


Fig. 19 A case study of dynamic partition reconfiguration on the autonomous F1/10 model car: **a** hardware components, **b** the "Straight Ahead" context, **c** the "Turn Corner" context, and **d** a system failure observed using the naive algorithm

Table 1 The autonomous control system has three applications

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)

Each application consists of multiple tasks along with their real-time requirements in different contexts

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 100 ms 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 1. 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 (NVIDIA 2019) 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. Based on the application requirement as specified in Table 1, the resource requirement for each application is assigned and configured as follows. When entering the Straight Ahead context, $\lambda_s = \{\mathcal{P}^n_s, \mathcal{A}_s, \mathcal{R}^r_s, 100\}$ where $P^s_1, P^s_3 \in \mathcal{P}^n_s$ are regular with availability factor $\alpha^s_1 = \alpha^s_3 = \frac{1}{128}$ while $P^s_2 \in \mathcal{P}^n_s$ has $\alpha^s_2 = \frac{63}{64}$. The reconfiguration regularity of P_1, P_2, P_3 is 1, 1, 100, respectively. When entering the Turn Corner context, $\lambda_c = \{\mathcal{P}^n_c, \mathcal{A}_c, \mathcal{R}^r_c, 100\}$ where $P^c_1, P^c_3 \in \mathcal{P}^n_c$ are regular partitions with availability factor $\alpha^c_1 = \frac{1}{64}, \alpha^c_3 = \frac{1}{128}$ while P^c_2 is a deleted partition which has $\alpha^c_2 = 0$. The reconfiguration regularity of P_1, P_3 is 1 for the Turn Corner R^{3}P. Each reconfiguration request has a response time of 50 ms 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 50*ms* 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.

Figure 20 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. 20a 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 50 ms. On the other hand, a naive algorithm will be more likely to undersupply the resource partitions during reconfiguration as shown in Fig. 20d. Spikes of response time are often close to the time of reconfiguration and the spikes indicate that P_3 are reconfiguration irregular. The performance degradation at time 32 causes the car to fail in avoiding obstacles as illustrated in Fig. 19d. Fig. 21 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 algorithm. This results in long latency and eventually causes the car to crash. Note that the spike at time 28 in Fig. 20d is caused by the fact that the ROS library is not a hard real-time library which sometimes takes longer to execute certain functionality.

6 Conclusion and future work

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.

There are several limitations to the proposed work which restricts its applicability. As the future work, we will explore solutions to relax these restrictions. In the following, we will discuss four issues regarding (1) different partition transition and reconfiguration semantics including overlapping requests; (2) different forms of availability factors; (3) reconfiguration in non-uniform environment (multi-resource environment); and (4) resource reconfiguration when the DPR algorithm is not able to construct a valid schedule.

Partition transition and reconfiguration semantics In this paper, we proposed a specific transition model regarding the performance degradation which is defined as reconfiguration supply regularity. However, different applications may require different transition models in order to preserve their system stability. By studying the needs of other applications, we can generalize the desired transition and

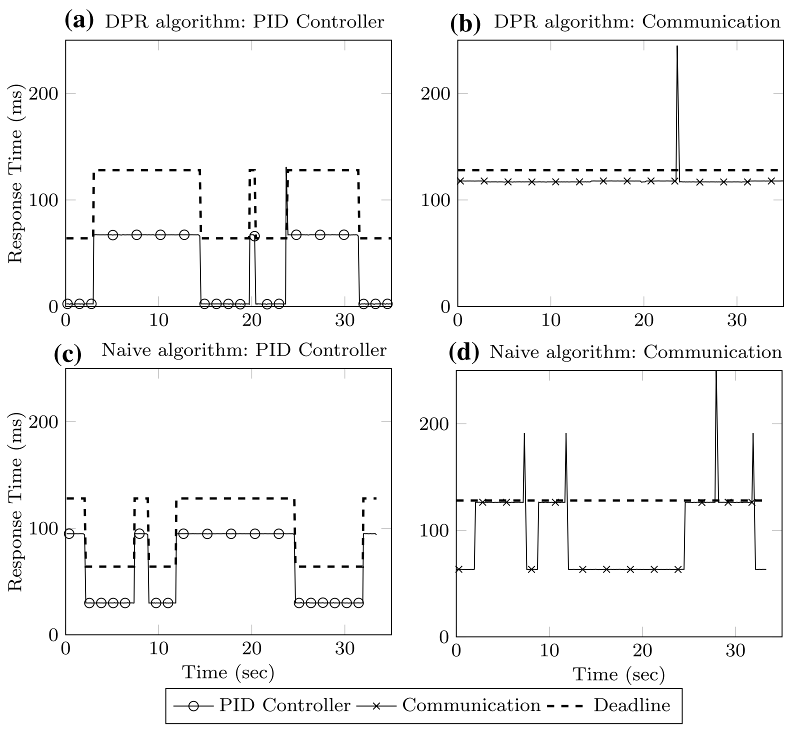


Fig. 20 The response time (ms) of each application at each time point (seconds) using the DPR algorithm (**a**), (**b**) and using the naive algorithm (**c**), (**d**)

reconfiguration semantics by specifying the change of resource supply rate and the bound of resource supply deviation at different time. For example, under appropriate concurrency semantics, our approach can be applied to handle concurrent requests by recomputing both the transition and cyclic schedules upon the arrival of new requests. Also, the system may have a more complex reconfiguration semantics by allowing a partition to request different amount of resource slices in each stage. For example, in the RPT stage, each partition may request a temporary resource supply surge or release more resource supply and demand another amount of resource supply after the RPT stage. This allows a finer-granularity coordination between the applications running on the same resource pool.

Different forms of availability factors In this paper, we restrict the availability factor to be the power of $\frac{1}{2}$. The scheduling problem with arbitrary availability factors is already known to be NP-hard even for slightly more relaxed assumptions than a single harmonic chain (Chen et al. 2017). To achieve better average-case system utilization, some other forms such as the Magic-7 or the combined approximation sequences proposed by Li and Cheng (2017) may be used in the 3-stage algorithm since they share similar characteristics as the one used in this paper. However, using other complex forms of availability factors may benefit the average resource

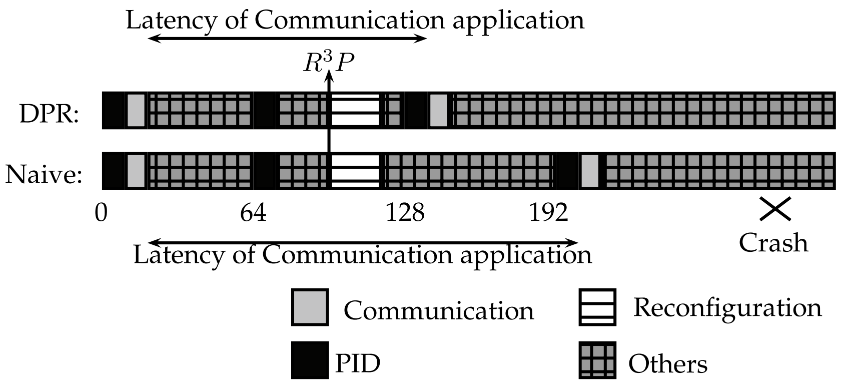

Latency of Communication application
R^3P
DPR:
Naive:
0
64
128
192
Crash
Latency of Communication application
Communication
Reconfiguration
PID
Others

Fig. 21 The schedules computed by the DPR and Naive algorithms during the context switch from "Turn Corner" to "Straight Ahead"

utilization in the static allocation scheme (Li and Cheng 2012). This may also significantly increase the complexity of the online reconfiguration algorithm.

Non-uniform environment The proposed solution in this paper is limited to uniform/single resource type. To extend it to non-uniform environment, one way is to simply apply the 3-stage algorithm on each physical resource. However, this may potentially increase the regularity of each resource partition by one. To further improve it, the key challenge is that end-to-end tasks may request resource during the execution of some resource slices in the non-uniform multi-resource environment where resource slices may have different sizes. Unexpected interruption of scheduled resource supply will break most of the extant models (Chen et al. 2017). If all the potential task request time points on a resource are known a priori, our proposed algorithm can be readily applied by modifying the computation of maximum supply shortfall of each resource partition based on these time points rather than the start of every resource slice. Further by not scheduling resource slices conflicting with these time points, the effective reconfiguration supply regularity and normal effective regularity can be achieved. However, for concurrent non-uniform resource reconfiguration, the task request time points on a resource can only be deduced after the reconfiguration of other resources are done unless this resource is the initial resource used by the end-to-end task. In the future work, each resource partition with the associated application may be assumed to have a predetermined set of possible task request time points such that each resource can be configured independently. This, however, will reduce the schedulability of the system.

No valid schedule can be found by the DPR algorithm The DPR algorithm proposed in this paper provides a solution for system to reconfigure its resource supply distribution for its application to coordinate and respond to the environment change. However, given an R^3P, there may not even exist a solution. There are a few approaches to handling this issue when the reconfiguration request is rejected. One is to define a fail-safe context in which a set of essential applications can always be safely reconfigured in the system at run time. The remaining unused resource can be reconfigured to supply other non-essential applications. The other approach is to define different priorities for the applications and let the system to satisfy the reconfiguration requests from higher priority applications first. Both directions will be explored as our future work.

Acknowledgements The work reported herein is supported by the Office of Naval Research under ONR Award N00014-17-1-2216 and by the National Science Foundation under NSF Award CNS-2008463.

References

Baruah SK, Cohen NK, Plaxton CG, Varvel DA (1996) Proportionate progress: a notion of fairness in resource allocation. Algorithmica 15(6):600–625

Biondi A, Buttazzo G, Bertogna M (2018) A design flow for supporting component-based software development in multiprocessor real-time systems. Real-Time Syst 54(4):800–829

Boudjadar J, Kim JH, Phan LTX, Lee I, Larsen KG, Nyman U (2018) Generic formal framework for compositional analysis of hierarchical scheduling systems. In: 21st IEEE International Symposium on Real-Time Distributed Computing (ISORC). IEEE, pp 51–58

Brandenburg B (2011) Scheduling and locking in multiprocessor real-time operating systems. PhD thesis, The University of North Carolina at Chapel Hill

Burns A (2014) System mode changes-general and criticality-based. In: Proc. of 2nd Workshop on Mixed Criticality Systems (WMC), pp 3–8

Burns A, Davis RI (2018) A survey of research into mixed criticality systems. ACM Comput Surv 50(6):82

Buttazzo G, Bini E, Wu Y (2010) Partitioning parallel applications on multiprocessor reservations. In: 22th Euromicro Conference on Real-Time Systems (ECRTS)

Buttazzo G, Bini E, Wu Y (2011) Partitioning real-time applications over multicore reservations. IEEE Trans Ind Inform 7(2):302–315

Calandrino JM, Leontyev H, Block A, Devi UC, Anderson JH (2006) LITMUSRT: a testbed for empirically comparing real-time multiprocessor schedulers. In: 27th IEEE Real-Time Systems Symposium (RTSS). IEEE, pp 111–126

Chen T, Phan LTX (2018) 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, pp 105–116

Chen WJ, Huang PC, Leng Q, Mok AK, Han S (2017) Regular composite resource partition in open systems. In: 38th IEEE Real-Time Systems Symposium (RTSS). IEEE, pp 34–44

Davis RI, Altmeyer S, Burns A (2018) Mixed criticality systems with varying context switch costs. In: 24th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS). IEEE, pp 140–151

de Niz D, Phan LT (2014) 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, pp 111–122

Deng Z, Liu JS (1997) Scheduling real-time applications in an open environment. In: 18th IEEE Real-Time Systems Symposium (RTSS). IEEE, pp 308–319

Easwaran A, Anand M, Lee I (2007) Compositional analysis framework using EDP resource models. In: 28th IEEE Real-Time Systems Symposium (RTSS). IEEE, pp 129–138

Evripidou C, Burns A (2016) Scheduling for mixed-criticality hypervisor systems in the automotive domain. In: WMC 2016 4th International Workshop on Mixed Criticality Systems

F1tenth (2019) F1tenth. http://f1tenth.org/

Feng AX (2004) Design of real-time virtual resource architecture for largescale embedded systems. PhD thesis, Department of Computer Science, The University of Texas at Austin

Gu X, Easwaran A (2016) Dynamic budget management with service guarantees for mixed-criticality systems. In: 2016 IEEE Real-Time Systems Symposium (RTSS). IEEE, pp 47–56

Gurobi (2019) Gurobi. http://gurobi.com/

Han S, Chen D, Xiong M, Lam KY, Mok AK, Ramamritham K (2012) Schedulability analysis of deferrable scheduling algorithms for maintain ingreal-time data freshness. IEEE Trans Comput 63(4):979–994

Herterich MM, Uebernickel F, Brenner W (2015) The impact of cyber-physical systems on industrial services in manufacturing. Procedia Cirp 30:323–328

Hu B, Huang K, Chen G, Cheng L, Knoll A (2016) Adaptive workload management in mixed-criticality systems. ACM Trans Embed Comput Syst 16(1):14

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

Lee J, Chwa HS, Phan LT, Shin I, Lee I (2017) Mc-adapt: adaptive task dropping in mixed-criticality scheduling. ACM Trans Embed Comput Syst 16(5s):163

Li Y, Cheng AM (2012) Static approximation algorithms for regularity-based resource partitioning. In: 33rd IEEE Real-Time Systems Symposium (RTSS). IEEE, pp 137–148

Li Y, Cheng AM (2017) Toward a practical regularity-based model: the impact of evenly distributed temporal resource partitions. ACM Trans Embed Comput Syst 16(4):111

Li Y, Cheng AMK (2015) Transparent real-time task scheduling on temporal resource partitions. IEEE Trans Comput 65(5):1646–1655

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

Liu CL, Layland JW (1973) Scheduling algorithms for multiprogramming in a hard-real-time environment. J ACM 20(1):46–61

Mok AK, Alex X (2001) Towards compositionality in real-time resource partitioning based on regularity bounds. In: 22nd IEEE Real-Time Systems Symposium (RTSS). IEEE, pp 129–138

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

Nikolov V, Wesner S, Frasch E, Hauck FJ (2017) A hierarchical scheduling model for dynamic soft-real-time system. In: 29th Euromicro Conference on Real-Time Systems (ECRTS), Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik

NVIDIA (2019) Embedded systems developer kits. http://devblogs.nvidia.com/jetson-tx2-delivers-twice-intelligence-edge/

Phan LT, Lee I, Sokolsky O (2010) Compositional analysis of multi-mode systems. In: 22nd Euromicro Conference on Real-Time Systems (ECRTS). IEEE, pp 197–206

Real J, Crespo A (2004) Mode change protocols for real-time systems: a survey and a new proposal. Real-Time Syst 26(2):161–197

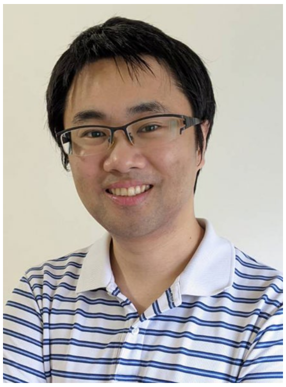
ROS (2019) Ros framework. http://wiki.ros.org/

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

Shin I, Lee I (2003) Periodic resource model for compositional real-time guarantees. In: 24th IEEE Real-Time Systems Symposium (RTSS). IEEE, pp 2–13

Shirero S, Takashi M, Kei H (1999) On the schedulability conditions on partial time slots. In: International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)

Xu H, Burns A (2019) A semi-partitioned model for mixed criticality systems. J Syst Softw 150:51–63



Wei-Ju Chen received the B.S. degree and the M.S. degree from the National Taiwan University, Taiwan, in 2005 and 2012. He received his Ph.D. from the University of Texas at Austin in 2020. His research interests include real-time systems, wireless networks and scheduling algorithms.

Peng Wu received the B.S. degree and M.S. degree from the Department of Electrical Engineering at Southwest Jiaotong University, Chengdu, China, in 2012 and 2015. He is currently a Phd student in the Department of Computer Science and Engineering at the University of Connecticut. His research interests include wireless networked control systems, real-time systems, and scheduling algorithms.

Pei-Chi Huang received her Ph.D. from the University of Texas at Austin, Texas, USA, in 2017. She is currently an assistant professor with the Robotics, Networking, Artificial intelligence (R. N. A.) Laboratory, in the department of Computer Science at University of Nebraska Omaha. Her research interests include cyber-physical systems, robotics, and machine learning.

Aloysius K. Mok holds the Quincy Lee Centennial Professorship in Computer Science at the University of Texas at Austin. Dr. Mok received his Ph.D., M.S. and B.S. degrees from the Massachusetts Institute of Technology and his research is in real-time systems, cyber-physical systems. He was a past recipient of the IEEE TC on Real-Time Systems Award and has received commendations from the United States Air Force for his advisory work on advanced systems.

Song Han received the B.S. degree from Nanjing University in 2003, the M.Phil. degree from the City University of Hong Kong in 2006, and the Ph.D. degree from the University of Texas at Austin in 2012, all in Computer Science. He is currently an Associate Professor in the Department of Computer Science and Engineering at the University of Connecticut. His research interests include cyber-physical systems, real-time and embedded systems, and wireless networks.

Authors and Affiliations

Wei-Ju Chen[1] · **Peng Wu**[2] · **Pei-Chi Huang**[3] · **Aloysius K. Mok**[1] · **Song Han**[2]

Peng Wu
peng.wu@uconn.edu

Pei-Chi Huang
phuang@unomaha.edu

Aloysius K. Mok
mok@cs.utexas.edu

Song Han
song.han@uconn.edu

1 The University of Texas at Austin, Austin, TX, USA

2 The University of Connecticut, Storrs, CT, USA

3 The University of Nebraska at Omaha, Omaha, NE, USA