Noname manuscript No.
(will be inserted by the editor)

Regular Composite Resource Partitioning and Reconfiguration in Open Systems

Received: date / Accepted: date

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 such applications. However, previous work on RRP model only discusses uniform resource environment, where there is only one physical resource or multiple physical resources with the same size of resource slices. The challenge is that an end-to-end task may experience unexpected supply shortfall if it requests resource at middle of the execution of a slice. This paper extends the RRP model to non-uniform multi-resource open system environments to tackle this problem. It first introduces a novel composite resource partition abstraction and then proposes algorithms to construct and reconfigure the composite resource partitions. Specifically, the *Acyclic Regular Composite Resource Partition (CRP)* scheduling algorithm constructs regular composite resource partitions and the *Dynamic Acyclic Composite Partition Reconfiguration (DCRP)* algorithm reconfigures the composite resource partitions in the run time upon requests of partition configuration changes. Our simulation results show that compared with the state-of-the-art methods, CRP can improve the acceptance ratio by at least 30% in non-uniform multi-resource environments, and DCRP can guarantee the resource supply during the online resource reconfiguration with moderate computational overhead. A multi-resource scheduling framework including both CPU and WiFi resources is also implemented on a real-life testbed to highlight the non-uniform multi-resource partitioning problem and evaluate the feasibility of the proposed theoretical model and scheduling algorithms in practical settings.

Keywords Regularity-based Resource Partition (RRP) · Composite Resource · Online Reconfiguration · Open System Environment

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 does not exsit a 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 in each application to schedule the tasks to meet the task-level timing constraints.

Many effective strategies have been proposed in the literature to allocate resources in such environment. Among those methods, the Regularity-based Resource Partition (RRP) model is an abstraction of component-based hierarchical scheduling systems where each component is an application with specified functional requirements and timing constraints (Feng 2004; Boudjadar et al. 2018; Li and Cheng 2017). As a hierarchical scheduling system, a component (or application) in the RRP model may consist of several sub-components. A parent component distributes its resource share to its sub-components, each of which in turn distributes it to its sub-components in a hierarchical fashion. Fig. 1 takes the CPU resource as an example, and gives an overview of the hierarchical resource scheduling model. In this example, individual applications utilize their resource interfaces to request CPU resource shares. The allocated resource shares for each application will then be distributed to its task group according to self-defined policies. Distributing resource in such fashion, the task-level scheduler of each application can independently schedule its own tasks based on the allocated resource.

Another popular approach in the literature to characterizing the resource usage interface is called the Periodic Resource Model (PRM) (Shin and Lee 2003) (or its variant the Explicit Deadline Periodic (EDP) model (Easwaran et al. 2007)). These models characterize the resource interface as a periodic execution budget, including the execution budget and a period. The EDP model extends the PRM model by using a deadline to limit the delay of the resource supply in each period.

The main difference between the RRP and PRM/EDP models is illustrated in Fig. 2. Given the resource demands of all the task groups, both RRP and EDP models will construct resource interfaces accordingly. The figure shows a possible schedule of resource allocation with a bandwidth assignment of 1/4 of the resource. The EDP model constructs a resource interface which has zero resource supply in time interval $(1, 6]$. By contrast, the RRP model bound the length of such zero-supply interval by explicitly specifying the allowed resource supply jitter. Ideally, from the application's point of view, the resource should be supplied uniformly over any time interval as if it is dedicated to the appli-

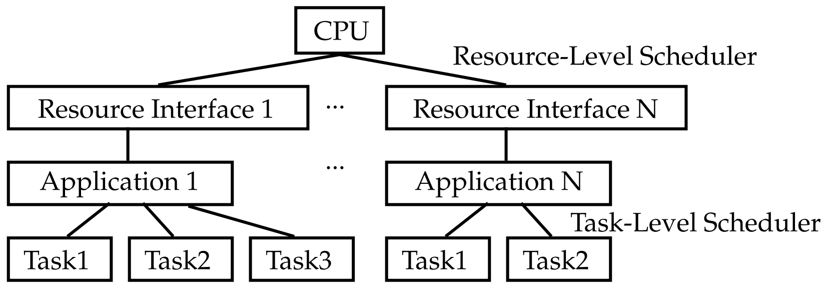


Fig. 1: Overview of the hierarchical scheduling model.

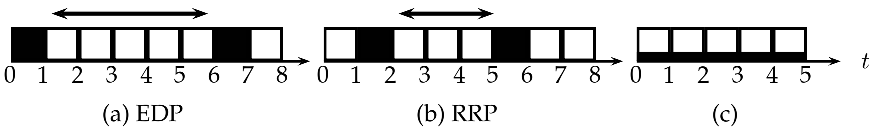


Fig. 2: (a) and (b): two possible schedules under the EDP and RRP models with a required bandwidth of $\frac{1}{4}$. (c): ideal resource supply from the application's point of view.

cation, but at a slower rate ($\frac{1}{4}$) as depicted in Fig. 2 (c). Taking the resource supply jitter into consideration, the resource supply specified by the resource interface under the RRP model better approximates the ideal supply which is uniform over any time interval. Hence, changes to the task group can be more easily accommodated by the application's own task scheduler by rescheduling tasks within its allocated resource partition. This is possible as long as the application's task utilization remains below the assigned availability factor, thus avoiding the need to change the resource interface (Feng 2004; Li and Cheng 2015). The jitter requirement however makes the designs of the scheduling algorithms under the RRP model more complex. The existing scheduling algorithms often limit the form of resource supply rate of each resource interface and give polynomial time solutions. For example, the Adjusted Availability Factor (AAF) algorithm allocates resource partitions with availability factors (supply rate) of power of $\frac{1}{2}$ (Feng 2004). Li and et al. (Li and Cheng 2012, 2017) proposed 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 improve the resource utilization overhead.

A notable limitation of previous work on the RRP model is that they only consider the uniform environment where the minimal intervals (*resource slices*) assigned to each task for execution on each physical resource are the same. Either the single resource environment (Mok and Alex 2001; Feng 2004) or multi-resource environment where all resources are restricted to have the same size for all resource slices (Li and Cheng 2012) is considered. Even in the recently introduced reconfigurable RRP model (Chen et al. 2021), the resource misalignment problem is only brought to the surface without further study. In the multi-resource open system environment, the size of the resource slices are very likely to be non-uniform across different types of physical resources. Thus the resource slices from different physical resources may not be aligned and hence end-to-end tasks might request resource at the middle of the execution of a resource slice. This *resource misalignment problem* will result in accumulated unexpected delay for end-to-end tasks when accessing multiple non-uniform physical resources in a sequential fashion, leading to task deadline misses.

In this paper, we first identify the resource misalignment problem and the challenges to solve it. Considering the problem, we then introduce a novel *composite resource partition* abstraction for non-uniform multi-resource environments. Based on this resource interface, the acyclic regular composite resource (CRP) algorithm and dynamic composite partition reconfiguration (DCRP) algorithm are proposed to construct and reconfigure the composite resource partitions, respectively. The key idea behinds CRP algorithm is to consider the resource requesting time for each resource. The resource misalignment problem can be mitigated by configuring and scheduling resource partitions in a way that resource won't be requested in the middle of any resource slice. To integrate the idea of composite resource partition with the reconfigurable RRP model, the resource requesting time and the performance degradation shall both be taken into consideration during the reconfiguration of composite resource partitions. Finally, a non-uniform multi-resource scheduling framework jointly considering the CPU and WiFi resources are presented to evaluate the feasibility of the proposed resource interface and scheduling algorithms. Extensive simulation results are also presented to give a thorough evaluation on the proposed algorithms under more general settings.

In the rest of the paper, we first review the related work in Section 2. We then review the RRP model in uniform environment, the reconfigurable RRP model extension and discuss the challenges in applying the model to non-uniform environment in Section 3. In Section 4, we introduce the concept of composite resource partition. In Section 5, we formulate the composite resource partition scheduling problem and present the CRP algorithm to construct composite resource partitions. We then study the composite resource partition reconfiguration model in Section 6 and present the detail of the DCRP algorithm. The multi-resource scheduling framework design is given in Section 7 and we present the experiment results in Section 8. Finally, Section 9 concludes the paper and discusses the future work.

2 Related Work

This section reviews the literature on resource allocation in open system environment (Section 2.1) and online mode-change resource reconfiguration (Section 2.2).

2.1 Single and Multiple Resources Allocation in Open System Environment

Many scheduling methods have been proposed in the literature to allocate resources in open system environment. Among those methods, the Regularity-based Resource Partition (RRP) model (Feng 2004; Chen et al. 2017; Li and Cheng 2017) is an abstraction of component-based hierarchical scheduling systems. The concept of regularity was first introduced by Shirero (Shirero et al. 1999) and was then extended to the regularity-based resource partition model by Mok and Alex (Mok and Alex 2001). Mok and Feng further 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 2012, 2017). More recently, the RRP model is extended to support online reconfiguration to adapt to the resource request changes in the run time (Chen et al. 2021). Besides the RRP model, there are many other studies on hierarchical scheduling which characterize resource interfaces using different models (Shin and Lee 2003; Easwaran et al. 2007; Boudjadar et al. 2018). The most popular model among them is the EDP model (Easwaran et al. 2007) which introduces a relative deadline parameters based on the PRM model. We already discussed the EDP models and compared them with the RRP model in Section 1. Most aforementioned work only discusses the resource allocation in uniform environment. However, in the multi-resource open system environment, the sizes of the resource slices are very likely to be non-uniform across different types of physical resources, and new resource interface and resource scheduling methods have to be designed.

Some research studies apply end-to-end resource reservation approaches to achieving performance isolation. For example, Buttazzo et al. (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 available computing power into interface specifications. However, these end-to-end resource reservation approaches do not consider dynamic workload in open systems, and thus cannot adapt to online resource request changes.

2.2 Online Mode-change Resource Reconfiguration

There are several related research areas on scheduling tasks with varying timing requirements. For example, Burns and Davis present a survey on mixed-criticality systems (Burns and Davis 2018), in which the task specifications depend on the system state/criticality. Jiang et al., (Jiang et al. 2019) proposed a mixed-criticality system for timely handling of I/O. It provides temporal and spatial isolation and prohibits fault propagation with small extra overhead based on hardware assisted virtualisation to offer good timing preditability. The 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) require the design of mode-change 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 switch. TODO: what are the difference among these work?

There are also some research work on the multi-mode resource interface design (Evripidou and Burns 2016; Phan et al. 2010; Li et al. 2018; Nikolov et al. 2017) where the resource interface may change in the run time for single resource environment. For instance, Evripidou and Burns (Evripidou and Burns 2016) used a two-level scheduler or a hypervisor to handle the criticality mode change. Phan et al. (Phan et al. 2010) proposed a compositional analysis of the multi-mode resource interface. Li et al. (Li et al. 2018) used virtual machine (VM) to support multi-mode virtualization where the parameters of the VM change with minimum transition latency. In an earlier work, Chen et al. presented the precise performance semantics of resource interfaces and their performance degradation during reconfiguration in RRP model in uniform single resource environment (Chen et al. 2021). In this paper, we focus on the resource interface reconfiguration in non-uniform multi-resource environment while ensuring graceful performance degradation during the reconfiguration.

3 RRP model in Uniform Environment

This section revisits the regularity-based resource partition (RRP) model in uniform environment.[1] We first define the time systems used in this paper, and review the concepts of RRP model and dynamic partition reconfiguration (DPR) in the uniform environment (Mok and Alex 2001; Feng 2004; Chen et al. 2017, 2021). We then discuss the challenges to extend the reconfigurable RRP model to non-uniform multi-resource environment.

3.1 Time Systems

In this paper, we will use two time systems in the RRP model 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. 3 (a). For the physical resource Π, a minimum non-preemptible physical time interval (2 in the example shown in Fig. 3) is defined as a *resource slice* and allocated to an application exclusively. The physical resource is allocated to the application(s) in units of resource slices as illustrated in Fig. 3 (b) (Feng

[1] The definitions of the time systems and the RRP model have been presented in (Chen et al. 2021). To make this work self-contained, we repeat their definitions in this section.

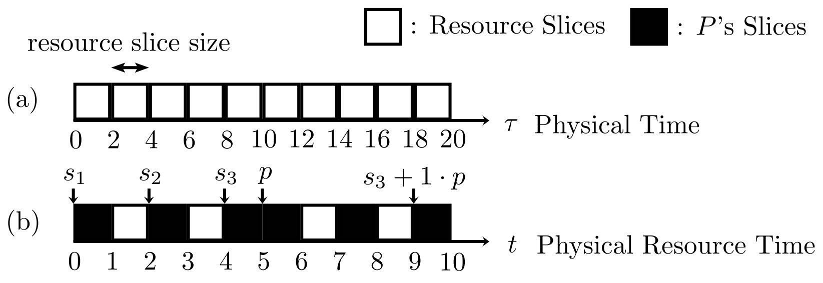


Fig. 3: (a) Physical resource Π is divided into units of resource slices with a minimum physical time interval of 2. A resource partition P is a set of resource slices allocated to an application. (b) Physical resource Π with physical resource time.

2004), where a *resource partition* P is a set of resource slices. The second time system, *physical resource time*, is defined as follows.

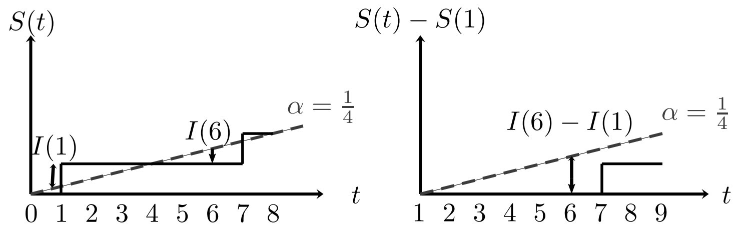
Definition 3.1: The physical resource time t of a physical resource Π is a function of the physical time τ such that $t = \frac{\tau}{Q}$ where Q is the resource slice size of Π.

As illustrated in Fig. 3, non-negative integer t denotes a time at the boundaries of resource slices and non-integer t denotes a time in the middle of a resource slice time interval. 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 resource time because resource slices are non-preemptible. In the following of the paper, we always refer the time to be physical resource time unless we specify the time to be others. Moreover, we 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*.

3.2 Regularity-based Resource Partition (RRP)

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

Definition 3.2: A resource partition P on a physical resource Π is a tuple $(\mathcal{S}, p)$, where $\mathcal{S} = \{s_1, s_2, \cdots, s_n : 0 \leq s_1 < s_2 < \cdots < s_n < p\}$ is a set of n time points that denote the start 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 $[s_k + x \cdot p,\ s_k + 1 + x \cdot p)$, $x \in \mathbb{N}, 1 \leq k \leq n$.



(a) Instant Regularity (b) Supply Deviation

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

Definition 3.3: 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 P is $(\{s_1 = 0, s_2 = 2, s_3 = 4\}, 5)$ and the supply function of P has $S(1) = 1, S(2) = 1, S(3) = 2, S(4) = 2$, and so on.

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 the concept of *supply regularity* is introduced to capture the jitter in the resource supply.

Definition 3.4: The availability factor α of a resource partition $P = (\mathcal{S}, p)$ is defined as $\frac{|\mathcal{S}|}{p}$ where $|\mathcal{S}|$ is the number of elements in $\mathcal{S}$.

Definition 3.5: The instant regularity $I(t)$ for a resource partition P at time t is defined as $I(t) = S(t) - \alpha \cdot t$.

这个小于1 的 物理意义是什么

Definition 3.6: Let a, b, k be non-negative integers. The supply regularity R of resource partition P is defined as the smallest k such that $|I(b) - I(a)| <$ ~~$k, \forall b \geq a$.~~

Fig. 4 (a) illustrates the ideal and actual resource supply of a resource partition P which has $\frac{1}{4}$ fraction of resource. 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 $I(6)$. Fig. 4 (b) 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**.

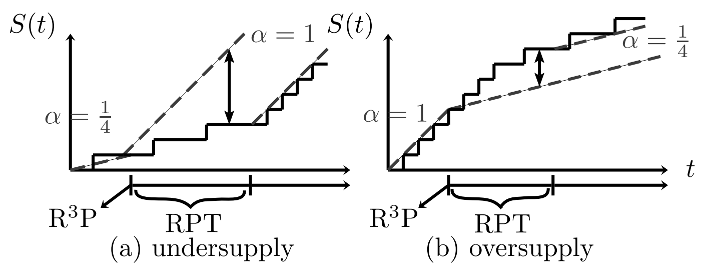


Fig. 5: The dotted line illustrates that the requested availability factor changes from $\frac{1}{4}$ and 1 to 1 and $\frac{1}{4}$ in (a), (b), respectively, at the time of R^{3}P. The arrow shows the supply deviation during the reconfiguration where there is resource undersupply in (a) and resource oversupply in (b), respectively.

Definition 3.7: 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 Reconfigurable RRP model

In the RRP model, the applications may request to reconfigure their resource partitions on demand. In the uniform environment, an application can issue a *Reconfiguration Request of Resource Partition* (R^{3}P) 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^{3}P. The system then enters the *Resource Partition Transition (RPT)* stage where resource partitions are reconfigured and temporary resource undersupply or oversupply may happen as shown in Fig. 5 (a) and (b), respectively. After the RPT stage is over, the reconfigured resource partitions will supply the resource to applications according to the new availability factor and new supply regularity by approximating the new ideal supply curve in a staircase function as depicted in Fig. 5 (a) (lower dotted supply curve) and (b) (upper dotted supply curve).

Recall that a resource partition P is a tuple $(\mathcal{S}, p)$ which describes its cyclic schedule and period. In each stage, we can describe the cyclic schedule of the resource partition using this tuple counting from time zero at the start of the stage. We thus describe the resource partition P at different stages using different superscripts. 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 (R^3P).

Definition 3.8: Reconfiguration Request of Resource Partition (R^{3}P) is defined as a tuple $\lambda = \{\mathbb{P}^n, \mathcal{A}, \mathcal{R}_r, T\}$ where $\mathbb{P}^n$ is the target set of resource partitions after the request; each resource partition $P_i^n \in \mathbb{P}^n$ has an associated availability factor of $\alpha_i^n \in \mathcal{A}$ and P_i will have reconfiguration supply regularity (see Def. 3.10) of $R_{ri} \in \mathcal{R}_r$. T is the maximum time allowed for the reconfiguration to complete.

After R^3P, the resource supply of a resource partition P is guaranteed by enforcing the availability factor and regularity of P^n. The performance semantic of the reconfiguration is achieved by enforcing the resource supply and the supply deviation of P for any time interval before, during and after the transition stage, which will be defined later. We 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 work, the performance semantic defines the maximum difference between the actual resource supply and the desired supply even during the reconfiguration. The concept of reconfiguration supply regularity is thus introduced to formally define such performance semantics. For this aim, the definition of instant regularity is extended to accommodate the change of availability factor. Following the similar notation of a resource partition P, we use α^o and α^n to denote the desired fraction of resource before and after the reconfiguration, respectively. The definition of instant regularity is thus extended as follows.

Definition 3.9: 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 $\underline{R^3P}$; 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. 6 (a), $I(1)$ indicates that there is resource oversupply at time 1 while $I(8)$ indicates that there is resource undersupply at time 8. Also, $S(b) - S(a)$ denotes the actual resource supply during 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 definitions of instant regularity, we define the *reconfiguration supply regularity* for uniform environment as follow.

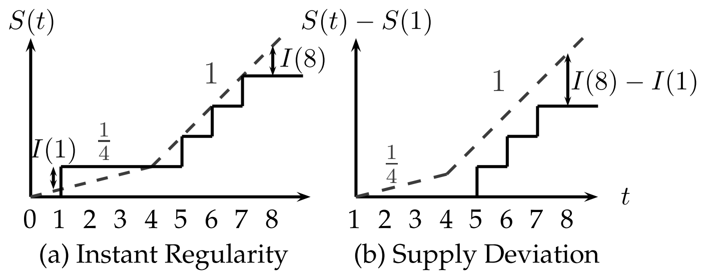


Fig. 6: Dotted line shows the ideal amount of resource supply which is $\frac{1}{4}$ from time 0 to 4 and 1 after time 4. $I(1), I(8)$ illustrates the instant regularity at time 1 and 8, respectively. $I(t) - I(1)$ illustrates the deviation of resource supply for time interval $[1, t)$.

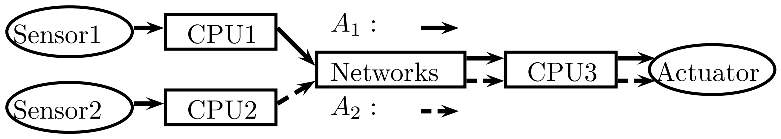


Fig. 7: Solid and dotted lines represent two end-to-end wireless networked control applications A_1 and A_2 sharing CPU and wireless network channels.

Definition 3.10: 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 $I(b) - I(a) > -k, \forall b \geq a$.

Note that the reconfiguration supply regularity *only* defines the maximum undersupply while the normal supply regularity restricts *both* the maximum undersupply and oversupply. This relaxation provides more flexibility when resolving the schedule conflicts during the reconfiguration while still ensuring the bounded resource undersupply.

3.4 Challenges of RRP model in non-uniform multi-resource environment

In the uniform environment, tasks are assumed to only access resource at the resource slice boundaries. However, this may not always be true in a practical system, especially in multi-resource environment where resources have different size of resource slices. In such environment, each application may have tasks accessing multiple physical resource in a predefined sequence. To describe this task model, we define the periodic end-to-end task as follows:

Definition 3.11: A periodic end-to-end task K is defined as $(\mathcal{PI}, \mathcal{X}, p, d)$ where $\mathcal{PI} = \{\Pi_1; \Pi_2; \cdots; \Pi_n\}$ is a sequence of n physical resources that K will access in sequence, $\mathcal{X} = \{x_1; x_2; \ldots; x_n\}$ is the corresponding requested amount of resource slices, p is the period and d is the relative deadline in units of physical time.

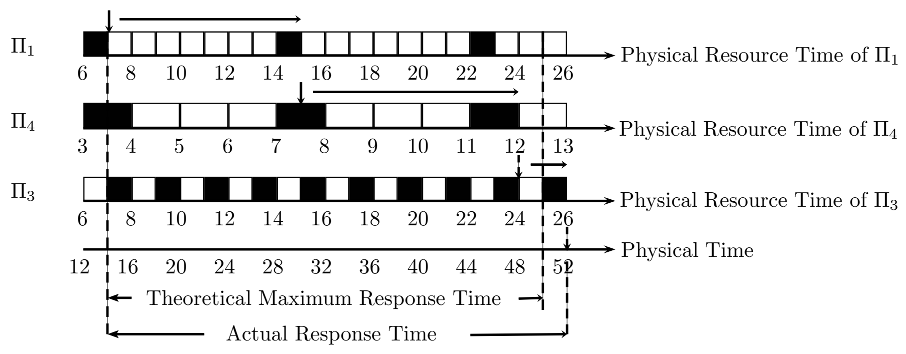


Fig. 8: Colored resource slices on physical resource Π_1, Π_4, Π_3 denote three regular resource partitions $P_{1,1}, P_{4,1}, P_{3,1}$ assigned to the application A_1.

Fig. 7 gives an example of two end-to-end wireless networked control applications in multi-resource environment. Let physical resource CPU1, CPU2, CPU3 and Network denoted as Π_1, Π_2, Π_3 and Π_4, respectively. Application A_1 (solid line) has an end-to-end task K_1 with physical resource access sequence $\mathcal{PI}_1 = \{\Pi_1; \Pi_4; \Pi_3\}$ and application A_2 (dotted line) has an end-to-end task K_2 with resource access sequence $\mathcal{PI}_2 = \{\Pi_2; \Pi_4; \Pi_3\}$. In this paper, we focus on resource-level scheduler rather than task-level scheduler. Thus, in the following paper, we will only use the response time of single task instance to evaluate the resource supply and supply deviation of resource partitions.

In non-uniform multi-resource environment, an end-to-end task may experience extra delay if resource slices are not aligned. Fig. 8 illustrates a partition schedule for application A_1. The resource slice sizes Q_1, Q_4 and Q_3 for physical resources Π_1, Π_4 and Π_3 are 2, 4 and 2 respectively. Colored resource slices denote three regular resource partitions $P_{1,1} = (\{6\}, 8), P_{4,1} = (\{3\}, 4), P_{3,1} = (\{1\}, 2)$ on physical resource Π_1, Π_4, Π_3, respectively, assigned to A_1. Note that, in this paper, we use $P_{i,j}$ to denote the partition on physical resource Π_i assigned to application A_j. These three regular partitions have availability factor $\alpha_{1,1} = \frac{1}{8}, \alpha_{4,1} = \frac{1}{4}, \alpha_{3,1} = \frac{1}{2}$, respectively. Let's consider an instance of an end-to-end task $K_1 = (\{\Pi_1; \Pi_4; \Pi_3\}, \{x_1 = 1; x_4 = 1; x_3 = 1\}, 36, 36)$ of application A_1 where the task has a relative deadline of 36 in physical time. The arrows denote the time points on each physical resource when a task instance starts requesting resource and finishes its execution. Given the resource supply of these partitions, the theoretical maximum response time of such task instance is $\sum\limits_{i=1,4,3} \lceil \frac{x_i}{\alpha_{i,1}} \cdot Q_i \rceil = 36$ in physical time. However, this task instance has to wait 2 extra time units during the physical time interval $[30, 32)$ on resource Π_4 because it cannot start executing right away in the middle of an resource slice of Π_4. This prolongs the latency to 38 in physical time and causes the instance miss the deadline. We call this resource misalignment problem.

Compensating the loss of resource supply because of resource misalignment problem with either increasing the availability factor or setting a smaller deadline for the task is often not viable. For the former approach, the system will need to provide extra $\frac{1}{2}$ of resource to a resource partition with α of $\frac{1}{2}$. For the latter approach, the deadline may need to be set very small and this may cause the system to be not schedulable. In this work, we mitigate the resource misalignment problem by judiciously scheduling resource slices such that no task will start requesting resource at the middle of a resource slice of its resource partition.

The resource misalignment problem will also happen during the reconfiguration of resource partitions. Avoiding task requesting resource at the middle of a resource slice of its resource partition while ensuring the resource supply and supply deviation during reconfiguration makes things much more complicated and increases the complexity of the algorithm. In this paper, we will propose an online algorithm with acceptable run-time complexity and schedulability.

4 Composite Resource Partition: Definition and Scheduling

To deal with the aforementioned challenges in non-uniform multi-resource environment, we need to extend the RRP model to consider requesting time of a resource partition. In this section, we introduce the concept of composite resource partition by extending the RRP model with the idea of effective supply regularity to address the resource misalignment problem.

4.1 Effective Supply Regularity

We first extend the definition of a resource partition with the *request offsets* to take the resource requesting time into consideration.

Definition 4.1: A resource partition P on a physical resource Π in non-uniform environment is a tuple $(\mathcal{S}, O, p)$, where $\mathcal{S} = \{s_1, s_2, \cdots, s_n : 0 \le s_1 < s_2 < \cdots < s_n < p\}$ is a set of n time points that denote the start time of the resource slices (called the *slice offsets*) allocated to the partition, and $O = (\mathcal{O} = \{o_1, o_2, \cdots, o_{n'}, : 0 \le o_1 < o_2 < \cdots < o_{n'} < \overline{p}\}, \overline{p})$ denotes a set of n' time points when the resource may be requested (called the *request offsets*); p and $(\overline{p})$ are the partition period and offset period, respectively. The physical resource Π is available to the application to which the partition P is allocated only during the time intervals $[s_k + x \cdot p,\ s_k + 1 + x \cdot p)$, $x \in \mathbb{N}, 1 \le k \le n$ and application may request resource at $[o_k + x \cdot \overline{p},\ o_k + 1 + x \cdot \overline{p})$, $x \in \mathbb{N}, 1 \le k \le n'$.

The request offsets of a resource partition are decided according to how the resource partition can be requested for resource. For example, the 3 request offsets of the resource partitions $P_{1,1}, P_{4,1}, P_{3,1}$ in Fig. 8 can be represented as $O_{1,1} = (\{0, 1, 2, 3, 4, 5, 6, 7\}, 8), O_{4,1} = (\{3.5\}, 4), O_{3,1} = (\{0\}, 2)$. The RRP model in uniform environment assumes that the request offsets are at all integer physical resource time points. In the rest of the paper, we still assume that tasks can only have integer execution time but can request resource at non-integer physical resource time points. We still refer the time to be physical resource time but the physical resource time now can be non-integer, denoting that the time point is in the middle of a resource slice. With this extension, we further extend the definition of the supply function as follows.

Definition 4.2: For any non-negative resource time t, the supply function of the resource partition P at time t equals to $S(\lfloor t \rfloor)$. That is, $S(t) = S(\lfloor t \rfloor)$.

Based on Definition 4.2, the resource supply in time interval $[0, t)$ is equal to the resource supply in time interval $[0, \lfloor t \rfloor)$ since there is no complete resource slice in time interval $[\lfloor t \rfloor, t)$.

Lemma 4.1: The resource supply of resource partition P in time interval $[a, b)$ is $S(b) - S(a) - 1$ if a is non-integer and there is a resource slice s_i s.t. $\lfloor a \rfloor = s_i$. Otherwise, it is $S(b) - S(a)$.

As an example shown in Fig. 8, if an instance of the end-to-end task requests resource from resource partition $P_{4,1}$ at its physical resource time point 7.5, it cannot start execution before the end of the current resource slice because the scheduling decisions are always made at the integral domain of physical resource time. The effective supply regularity is hence defined as follows.

Definition 4.3: Let e, x_1, x_2 be non-negative integers, $\mathcal{S} = \{s_1, s_2, \cdots, s_n\}$ be the slice offsets of ~~the~~ resource partition P, p be the period of P and $O = (\{o_1, o_2, \cdots, o_{n'}\}, \overline{p})$ be the request offset of P. The effective supply regularity $\overline{R}$ of a resource partition P is defined as the smallest integer k such that for any $s = s_i + x_1 \cdot p, o = o_j + x_2 \cdot \overline{p}$ and e, where $0 < i < n, 0 < j < n', e, x_1, x_2 \in \mathbb{N}$

$$\begin{cases} |I(o+e) - I(o) - 1| < k & \text{if } o \notin \mathbb{N} \text{ and } \exists s = \lfloor o \rfloor \\ |I(o+e) - I(o)| < k & \text{otherwise} \end{cases}$$

The effective supply regularity defines the maximum supply deviation from the ideal resource supply with regards to the application requesting time. Note that, this definition is backward compatible with the original definition of supply regularity where the request offsets are assumed to include every integer time point.

Definition 4.4: A resource partition is effective regular if and only if it has effective supply regularity of 1.

The following Lemma states that a regular partition may not be effective regular.

Lemma 4.2: A regular resource partition P is not effective regular if there exists non-negative integers i, j, x_1, x_2 such that $s_i + x_1 \cdot p = \lfloor o_j + x_2 \cdot \overline{p} \rfloor$.

Proof If there exists such $s = s_i + x_1 \cdot p$ and $o = o_j + x_2 \cdot \overline{p}$ for a regular partition P, by Def. 4.3 and 3.5, we have

$$|S(o+p) - S(o) - \alpha \cdot p - 1| < \overline{R} \tag{1}$$

for a time interval of the partition period p. Also, from Def. 4.1, P has $|\mathcal{S}|$ resource slices over the time interval of p; and from Def. 3.4 and 4.2, we have

$$S(o+p) - S(o) = |\mathcal{S}| = \alpha \cdot p \tag{2}$$

So, we have $1 < \overline{R}$ and hence P is not effective regular.

4.2 Composite Resource Partition

We now define composite resource partition and its regularity.

Definition 4.5: A composite resource partition C is defined as a tuple $(\mathbb{P}, \Pi^c, E)$ where $\mathbb{P}$ is a set of resource partitions, Π^c is a set of physical resource and E is a binary relation on Π^c. The partition scheduled on physical resource $\Pi_i \in \Pi^c$ is denoted as P_i and E represents the possible access order of the resource partitions.

The composite resource partition can be simply considered as a collection of resource partition on a set of physical resource with a predetermined resource access order denoted as E. For example in Fig. 7, let Π_1, Π_4 and Π_3 denote the CPU1, Network and CPU3 resources respectively, and application A_1 has resource access order $E = \{(\Pi_1, \Pi_4), (\Pi_4, \Pi_3)\}$. The resource-level scheduler will construct a composite resource partition $C = (\{P_{1,1}, P_{4,1}, P_{3,1}\}, \Pi^c = \{\Pi_1, \Pi_4, \Pi_3\}, E)$ for A_1 with scheduling assignment as illustrated in Fig. 8.

Definition 4.6: A composite resource partition C is regular if and only if all of its resource partitions are effective regular. If not, it is irregular.

5 Composite Resource Partition Scheduling Problem

In this section, we consider how to schedule composite resource partitions in multi-resource environment. We first formulate the Acyclic Regular Composite Resource Partition (CRP) scheduling problem and prove its complexity. A feasible partition scheduling condition is proved as a theorem. Build upon this feasible condition, we then propose two CRP algorithms for the general problem and online scheduling.

5.1 Problem Formulation and Complexity

We first define the problem and prove its complexity.

Problem V.1: *Acyclic Regular Composite Resource Partition (CRP) Scheduling Problem*: Given the resource demand from each application A_i, which includes the desired availability factor $\alpha_{j,i}$ for the effective regular partitions $P_{j,i}$ on physical resource Π_j and the resource access order E_i, the CRP problem is to construct a regular composite resource partition $C_i = (\mathbb{P}_i, \Pi^c, E_i)$ for each

A_i with the assumption that the total resource access order $E^t = \{(\Pi_j, \Pi_k) \mid (\Pi_j, \Pi_k) \in E_i, \forall i\}$ is assumed to have no cycles.

The CRP problem is a special case of the general composite resource partition scheduling problem which can be proved to be NP-hard in Theorem 5.1. We first prove the following Lemma to be used later.

Lemma 5.1: For a resource partition P with an availability factor of $\frac{1}{m}$ where m is a positive integer, it is regular if and only if every successive resource slices are m slices apart.

Proof We first prove the sufficient condition. Let t_1, d, m be integers and $t_2 = t_1 + d + x \cdot m,\ 0 \le d < m, x \ge 0$, where m is the partition period of P. If every successive resource slices are m slices apart, we have

$$x \le S(t_2) - S(t_1) \le x + 1 \tag{3}$$

Because $t_2 - t_1 = d + x \cdot m$ and $\alpha = \frac{1}{m}$, we have

$$-1 < S(t_2) - S(t_1) - \alpha(t_2 - t_1) < 1 \tag{4}$$

From Eq. 3 and 4, we have $|I(t_2) - I(t_1)| < 1,\ \forall t_2, t_1$ which means P is regular by Def. 3.6.

We then prove the necessary condition that if P is regular, every successive resource slices are m slices apart by contradiction. Assume there are two successive resource slices starting at resource time s_i, s_{i+1} that are not m slices apart, $s_{i+1} - s_i$ is either less than m or larger than m. If it is less than m,

$$I(s_{i+1} + 1) - I(s_i) \ge S(s_{i+1} + 1) - S(s_i) - \alpha * m = 1 \tag{5}$$

If it is larger than m,

$$I(s_{i+1}) - I(s_i + 1) \le S(s_{i+1}) - S(s_i + 1) - \alpha * m = -1 \tag{6}$$

For both cases, the supply regularity of P is higher than one.

We now proceed to prove that CRP problem is NP-hard.

Theorem 5.1: The CRP problem is NP-hard.

Proof To prove that the CRP problem is NP-hard, we reduce a single server periodic maintenance problem (PMP_1) (Mok et al. 1989) into a special case of the CRP decision problem.

The k-server periodic maintenance problem is NP-Complete (Mok et al. 1989) in the strong sense for $k \ge 1$ where k represents the number of machines allowed to be maintained concurrently. The input of the PMP_1 problem is as follows: Given a multi-set of positive integers $M = \{m_1, m_2, \cdots, m_n\}$, where m_i represents the maintenance period of machine i. A 1-server schedule N for M is an infinite sequence of one multi-set over $\{1, 2, \cdots, n, blank\}$, such that successive occurrences of i are exactly m_i slots apart.

The special case of the CRP decision problem is when the set of regular composite resource partitions to be constructed is all on one physical resource

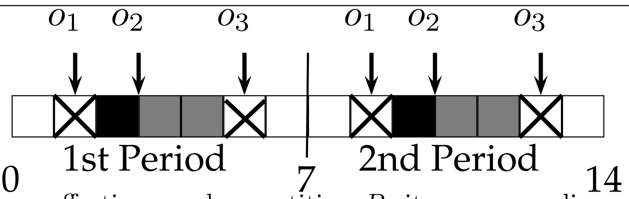


Fig. 9: For an effective regular partition P, its resource slices can only be allocated either all in black time interval, all in gray time interval or all in white time interval but not in the crossed out time interval. Moreover, there should be exactly one resource slice scheduled in each time interval.

Π, each composite resource partition has exactly one effective regular partition (see Def. 4.4) and tasks are assumed to request resource only at integer physical resource time for each effective regular partition. Hence, each effective regular partition can be considered as a regular partition. The decision problem is to decide whether there exists a valid schedule for each resource partition P_i.

We reduce an instance of the PMP_1 problem into the above special case of the CRP decision problem. For each element $m_i \in M$, we construct a regular resource partition P_i with availability factor $\alpha_i = \frac{1}{m_i}$. The distance between any two successive resource slices of P_i is exactly m_i apart by Lemma 5.1. This reduction can be done in polynomial time since it takes $O(1)$ time for each element in M.

If there exists a solution in the PMP_1 problem, we can construct the resource slice schedule $\mathcal{S}_i$ of resource partitions P_i as follows. Let the partition period p_i be equal to m_i and the only slice offset in $\mathcal{S}_i$ equals to the position of the first occurrence of i in the PMP_1 schedule. Because the successive occurrences of i are exactly m_i apart and $p_i = m_i$, resource partition P_i has $\alpha_i = \frac{1}{m_i}$ and it is regular by Lemma 5.1. On the other hand, if there exists a solution for this special CRP problem, we can construct a schedule for PMP_1 so that the occurrences of i are at $s_i + x \cdot p_i, \forall x \in \mathbb{N}$ where s_i is the only resource slice offset of P_i. This is also a valid schedule for PMP_1 because the distance between any two successive resource slices for resource partition P_i is exactly m_i by Lemma 5.1.

5.2 Feasible Condition

Next, we show the feasible scheduling condition for the effective regular partitions. We first give the assumptions to the CRP problem: 1) the α of each partition is limited to be $1/m$ for some integer m; 2) the period of request offsets $\overline{p}$ is $1/\alpha$. These two assumptions give rise to the important theorem in the next section for solving this problem.

Theorem 5.2: Let a partition P has resource slice offsets $\{s_1, s_2, \ldots, s_{n_2}\}$, period p and request offsets $O = (\{o_1, o_2, \ldots, o_{n_1}\}, \overline{p})$. P is effective regular if and only if $\exists k \in \mathbb{N}, 0 \le k \le n_1$ such that

$$o_k + (i-1)\overline{p} \le s_i \le o_{k+1} + (i-1)\overline{p} - 1, \qquad \text{for } 0 < i \le n_2$$

where $o_0 = o_{n_1} - \overline{p}$ and $o_{n_1+1} = o_1 + \overline{p}$.

Theorem 5.2 provides the feasible scheduling condition for effective regular partition, where resource slices should be scheduled in periodic time intervals enclosed by a pair of two neighboring request offsets, o_k and o_{k+1}, with a period of $\overline{p}$. Moreover, no resource slice should be scheduled to overlapping any request offsets. For example, in Fig. 9, resource partition P has request offset $O = (\{o_1 = 1.5, o_2 = 3, o_3 = 5.5\}, \overline{p} = 7)$. Each crossed-out time interval, for example, $[1 + 7x, 2 + 7x)$ and $[5 + 7x, 6 + 7x)$, $\forall x \in \mathbb{N}$, indicates there is a non-integer request offset o_i and we should not schedule a slice on that interval. Each colored time interval indicates a pair of neighboring request offsets, for example, $[2 + 7x, 3 + 7x), [3 + 7x, 5 + 7x), [6 + 7x, 8 + 7x)$, $\forall x \in \mathbb{N}$ are colored black, gray and white respectively. For example in Fig. 9, an effective regular resource partition P can have all its resource slices scheduled in the white time intervals such as $P = (\{0, 6\}, 14)$, $P = (\{7, 13\}, 14)$ or in the gray time intervals like $P = (\{3, 11\}, 14)$.

Proof We first prove the sufficient condition by showing that the effective supply regularity is one for any possible task arrival time point b and candidate of task leaving time point a. Recall that tasks have integer execution resource time, so let a, b be the following

$$\begin{cases} b = o_j + x \cdot \overline{p} & \forall j, x \in \mathbb{N} \\ a = b + y \cdot \overline{p} + m & \forall y, m \in \mathbb{N} \text{ and } 0 \le m < \overline{p} \end{cases}$$

If such k (o_k) exists, there will be exactly one resource slice in each time interval $[o_k + z \cdot \overline{p},\ o_{k+1} + z \cdot \overline{p})$, $\forall z \in \mathbb{N}$ and there is no resource slice at $\lfloor o_j \rfloor$ for all non-integer o_j. So we have

$$y \le S(a) - S(b) \le y + 1 \tag{7}$$

Also, $\alpha(a - b) = \alpha \cdot y \cdot \overline{p} + \alpha \cdot m = y + \alpha \cdot m$ because $\overline{p} = \frac{1}{\alpha}$. We then have

$$-\alpha \cdot m \le S(a) - S(b) - \alpha(a - b) \le 1 - \alpha \cdot m \tag{8}$$

From Definition 3.5,

$$-\alpha \cdot m \le I(a) - I(b) = S(a) - S(b) - \alpha(a - b) \le 1 - \alpha \cdot m \tag{9}$$

Because $m < \overline{p} = \frac{1}{\alpha}$,

$$-1 < I(a) - I(b) < 1 \tag{10}$$

Equation 10 holds for all possible task arrival time point b and candidate of leaving time point a, P is effective regular by Definition 4.3, P is effective regular.

Next, we prove the necessary condition by contradiction. Assume such k does not exist then there must exist d, g, x such that one of the following cases is true

$$\begin{cases} \textbf{Case 1: } s_d = \lfloor o_g + x \cdot \overline{p} \rfloor \text{ for a non-integer } o_g \\ \textbf{Case 2: } \begin{cases} o_g + x \cdot \overline{p} \le s_d \le o_{g+1} + x \cdot \overline{p} - 1 \\ s_d < s_{d+1} \le o_g + (x+1)\overline{p} - 1 \end{cases} \\ \textbf{Case 3: } \begin{cases} o_g + x \cdot \overline{p} \le s_d \le o_{g+1} + x \cdot \overline{p} - 1 \\ o_{g+1} + (x+1)\overline{p} - 1 < s_{d+1} \end{cases} \end{cases}$$

Note that $s_{n_2+1} = s_1 + p$. For **Case 1**, a resource slice is scheduled in the crossed-out time interval and P is not effective regular by Lemma **??**. For **Case 2** and **Case 3**, resource slices are scheduled in different colored time intervals.

Case 2:

Let task arrival time point $b = o_g + x \cdot \overline{p}$ and candidate of leaving time point $a = s_{d+1} + 1$. Because there are two resource slices s_d and s_{d+1} in time interval $[b,\ a)$ and $a - b \le \overline{p} = \frac{1}{\alpha}$, we have

$$S(a) - S(b) - \alpha(a-b) = 2 - \alpha(a-b) \ge 1$$

By Definition 3.5, $|I(a) - I(b)| \ge 1$ and thus $\overline{R} > 1$ by Definition 4.3. P is not effective regular.

Case 3:

Again let $b = o_{g+1} + x \cdot \overline{p}$ and $a = o_{g+1} + (x+1)\overline{p}$. Because there is no resource slice scheduled in time interval $[b,\ a)$ and $a - b = \overline{p} = \frac{1}{\alpha}$, we have

$$S(a) - S(b) - \alpha(a-b) = 0 - 1 = -1$$

The same reason as the one in the Case 2, P is not effective regular.

5.3 CRP Algorithm

Next, we propose the CRP algorithms for general problem and online scheduling by first giving an overview of the proposed algorithm in Alg. 1 and then describing each step.

Based on the Theorem 5.2, scheduling the partitions on a physical resource is equivalent to pick a pair of two neighboring request offsets (TODO: successive?) and to schedule exactly a resource slice in each periodic interval for each partition. In the CRP algorithm, we first perform a topology sort on the total resource access order E^t, which is the union of the access order of all the application A_i in Alg. 1. This will generate a linear order $\mathcal{T}' = \{\Pi'_1; \Pi'_2; \cdots; \Pi'_n\}$ for all the physical resources. The algorit~~hm then constructs all the effective~~ regular resource partitions on each physical resource following the order in $\mathcal{T}'$. For each physical resource Π_j, it first computes the request offset $O_{j,i}$ of each partition $P_{j,i}$ on this physical resource in line 4-6. The schedules are then

Algorithm 1: The CRP Algorithm

```
Input: The desired availability factor $\alpha_{j,i}, \forall j$, access order $E_i$ for all application
       $A_i$ and total access order $E^t = \{(\Pi_j, \Pi_k) \mid (\Pi_j, \Pi_k) \in E_i, \forall i\}$.
Output: The schedules $\{S_{j,i} \mid \forall i, j\}$ of each effective regular partitions $P_{j,i}$.
1  Enqueue all physical resource $\Pi_j$ into a queue $Q$ following the topological sorted
     order of $E^t$
2  while $Q \neq \emptyset$ do
3      Dequeue $\Pi_j$ from $Q$
4      for all $\alpha_{j,i}$ do
5          $O_{j,i}$ = RequestOffset($\alpha_{j,i}, \{P_{k,i} \mid (\Pi_k, \Pi_j) \in E_i\}$)
6      end
7      $\{S_{j,i} \mid \forall i\}$ = ConstructSchedule($\{(\alpha_{j,i}, O_{j,i}) \mid \forall i\}$)
8      if $\{S_{j,i} \mid \forall i\} = NULL$ then
9          return NULL
10     end
11 end
12 return $\{S_{j,i} \mid \forall i, j\}$
```

constructed based on the desired availability factor $\alpha_{j,i}$ and the request offsets $O_{j,i}$ for each application A_i in line 7.

For example in Fig. 7, application A_1 and A_2 are sharing physical resource CPU1, CPU2, CPU3 and wireless network channels, which are denoted as Π_1, Π_2, Π_3 and Π_4. A_1 and A_2 will be assigned a regular composite resource partition $C_1 = (\{P_{1,1}, P_{4,1}, P_{3,1}\}, E_1)$ $C_2 = (\{P_{2,2}, P_{4,2}, P_{3,2}\}, E_2)$ respectively. The topology sort can generate a linear order $E^t = \{\Pi_1; \Pi_2; \Pi_4; \Pi_3\}$. The algorithm first computes the schedules of the resource partitions on Π_1 and Π_2 based on their request offset decided by the sensor reading available times and frequencies. Based on the resource schedules of $P_{1,1}$ and $P_{2,1}$, we can compute the request offset $O_{4,1}$ and $O_{4,2}$ of $P_{4,1}$ and $P_{4,2}$. The schedules of $P_{4,1}$ and $P_{4,2}$ can then be computed based on $O_{4,1}$ and $O_{4,2}$. The same procedure will be repeated for Π_4 and Π_3.

The CRP problem is then broken down to two stages: 1) the computation of the request offsets and 2) the computation of the resource partition schedules given the request offsets. In the following, we address these two sub-problems.

5.3.1 Compute the Request Offsets

Algorithm 2 shows how to compute the request offset O of the resource partition P on physical resource Π. The inputs of the algorithm are the set of resource partitions $\mathcal{P}' = \{P_i : (\Pi_i, \Pi) \in E\}$ such that the application A may first request resource from Π_i and then request resource from Π right after. The algorithm then computes the possible time points that an instance of end-to-end task may request resource from $\Pi(P)$. For each resource partition P_i, a task instance can only finish execution at the end of a resource slice of P_i, which is $s_k + 1 + p_i \cdot x$, for some $s_k \in \mathcal{S}_i, x \in \mathbb{N}$ with respect to the physical resource time of Π_i , and request resource from $\Pi(P)$ at $s'_k + x \cdot p'_i, x \in \mathbb{N}$ with respect to the physical resource time of Π. In line 8, the algorithm computes

Algorithm 2: Compute the request offset O of a resource partition P

```
Input: The desired availability factor $\alpha$ and resource slice size Q of $P$, the set of
       resource partitions $\mathcal{P}' = \{P_i \mid (\Pi_i, \Pi) \in E\}$ and their resource slice size $Q_i$.
Output: The request offset $O = (\mathcal{O}, \overline{p})$ of $P$
1  Procedure RequestOffset($\alpha, \mathcal{P}'$)
2      $\mathcal{O} = \{\}$
3      $\overline{p} = \frac{1}{\alpha}$
4      for $P_i \in \mathcal{P}'$ do
5          $p'_i = p_i \cdot Q_i/Q$ //The resource slice offsets from $P_i$ will repeat for every $p'_i$
6          $H = LCM(p'_i, \overline{p})$ //The corresponding request offsets will repeat for every
             $H$ units of physical resource time of $\Pi$
7          for $s_k \in \mathcal{S}_i$ do
8              $s'_k = (s_k + 1) \cdot Q_i/Q$
9              for $x \leftarrow 0$ to $H/\overline{p} - 1$ do
10                 Insert($\mathcal{O}, s'_k + x \cdot p'_i \ (mod\ \overline{p})$) // Include all the possible time points
11             end
12         end
13     end
14     return $O = (\mathcal{O}, \overline{p})$
```

such time points. and adds all such time points to $\mathcal{O}$, which has a period of $\overline{p} = \frac{1}{\alpha}$ in line 9. Note that, in line 5-6, it computes the physical resource time unit conversion.

5.3.2 Compute the Partition Schedules

Given the request offsets of each partitions, the general problem to compute the schedule is NP-hard. The PMP_1 (Mok et al. 1989) can be reduced to this problem by setting the request offsets to be all integer time points. The proof will be similar to the one in Theorem 5.1. In the following, we shall gives two algorithms to construct the schedules given the request offsets. One trades complexity for better schedulability so it is good for offline scheduling. One trades schedulability for complexity so it is good for online scheduling. For both algorithms, we seek to construct a cyclic schedule with a hyper period length of all partition periods based on Theorem 5.2 and this can be done in two steps. A) The first step is to pick a pair of two neighboring request offsets which will enclose a periodic intervals for each partition. B) The second step is to schedule exactly a resource slice in each periodic interval chosen in first step for each partition. The offline and online algorithm differs in the ways of how to select the pair of neighboring request offsets and how to schedule resource slices in each periodic interval.

Alg. 3 shows the offline algorithm which has high complexity but better schedulability. For A) step, the algorithm tries to construct cyclic schedules for every partition by testing every combination of neighboring request offsets from each partition as the line 2. For B) step, given a combination of request offset pairs from each partition, the algorithm uses an EDF algorithm to compute the slice schedule in each periodic interval until a cyclic schedule is found.

Algorithm 3: Offline Algorithm

```
   Input: The availability factor $\alpha_{j,i}$ and request offset $O_{j,i}$ of each partition to be
          scheduled on physical resource $\Pi_j$.
   Output: The schedules of each partition $(\{\mathcal{S}_{j,i} \mid \forall i\}$. Otherwise, reject.
1  Procedure ConstructSchedule($\{(\alpha_{j,i}, O_{j,i}) \mid \forall i\}$)
2      for Each combination of request offset pairs do
3          for $t_t \leftarrow 0$ to the hyper period $H$ do
4              $m[t_t] = 0$
5          end
6          for Each request offset pair $(o_i^s, d_i^e)$ do
7              $r_{j,i} = \lfloor o_i^s \rfloor$
8              $d_{j,i} = \lfloor d_i^e \rfloor$
9              if $r_{j,i} \geq d_{j,i}$ then
10                 $d_{j,i} \pm = \overline{p}_{j,i}$
11             end
12         end
13         Enqueue all partitions $P_{j,i}$ into a queue $Q$ following the deadline
14         while $Q \neq \emptyset$ do
15             Dequeue $P_{j,i}$ from Q
16             if $Next(\mathcal{S}_{j,i}, r_{j,i}) \leq d_{j,i}$ or $d_{j,i} >= H$ and $Next(\mathcal{S}_{j,i}, 0) \leq d_{j,i} \mod H$
                then
17                 continue
18             $t = \mathrm{EDF}(r_{j,i}, d_{j,i}, m)$
19             if $t = NULL$ then
20                 break
21             end
22             Insert($\mathcal{S}_{j,i}, t$)
23             $r_{j,i} += \overline{p_{j,i}}$
24             $d_{j,i} += \overline{p_{j,i}}$
25             Enqueue $P_{j,i}$ into $Q$ with deadline $d_{j,i}$
26         end
27         if $Q = \emptyset$ then
28             return $\{\mathcal{S}_{j,i} \mid \forall i\}$
29         end
30     end
31     return NULL
32 Procedure EDF($r, d, m$)
33     for $t \leftarrow r$ to $d - 1$ do
34         $t_t = t \mod H$
35         if $m[t_t] = 0$ then
36             $m[t_t] = 1$
37             return $t_t$
38         end
39     end
40     return NULL
```

The H is the hyper period of all periods. In line 16-17, a cyclic schedule for a partition is found if we already scheduled a slice in each request offset period among the hyper period length. However, the complexity of this approach is very high. Fortunately, in practice, the number of resource partitions to be scheduled on a physical resource is not large and it is bounded by the sum of all the availability factors.

Algorithm 4: Online Algorithm

```
Input: The availability factor $\alpha_{j,i}$ and request offset $O_{j,i}$ of each partition to be
       scheduled on physical resource $\Pi_j$.
Output: The schedules of each partition $\{\mathcal{S}_{j,i} \mid \forall i\}$. Otherwise, reject.
1  Procedure ConstructSchedule($\{(\alpha_{j,i}, O_{j,i}) \mid \forall i\}$)
2      for $t_t \leftarrow 0$ to $H$ do
3          $m[t_t] = 0$
4      end
5      Enqueue all partitions $P_{j,i}$ into a queue $Q$ following the period $\overline{p}_{j,i}$
6      while $Q \neq \emptyset$ do
7          Dequeue $P_{j,i}$ from Q
8          for Each pair of neighboring request offsets $(r_{j,i}, d_{j,i})$ do
9              $t = EDF(r_{j,i}, d_{j,i}, m)$
10             if $t \neq NULL$ then
11                 break
12             end
13         end
14         if $t = NULL$ then
15             return NULL
16         end
17         for $k \leftarrow 0$ to $H/\overline{p_{j,i}} - 1$ do
18             $t_t = t + k * \overline{p_{j,i}} \mod H$
19             $m[t_t] = 1$
20         end
21     end
22     return $\{\mathcal{S}_{j,i} \mid \forall i\}$
```

To accelerate the scheduling computation, we can limit the choice of availability factor of each partition to be power of $\frac{1}{2}$ and the size of resource slice to be power of 2. The algorithm will then become similar to the AAF algorithm, scheduling partitions based on the period, but we also take request offsets into consideration. Alg. 4 shows such online algorithm which has linear complexity because we only consider one combination of request offset pairs . The algorithm combines step A) and B) by picking the first pair of neighboring request offsets that allows a resource slice to be scheduled by EDF algorithm as in line 8-13. The periods of every partition are harmonic and we schedule partition with smallest period first. We will need to mark the resource slice assigned for each interval in the hyper period H to avoid partition with larger period to conflict with partition with smaller period as in line 17-20.

6 Resource Reconfigurability in RRP Model

In this section, we consider the dynamic partition reconfiguration problem for the composite resource partition. We first introduce the definition of reconfiguration request for acyclic composite resource partitions, the exact regularity semantics and its definition, the Dynamic Acyclic Composite Resource Partition Reconfiguration (DCRP) problem and then propose a solution with reasonable complexity for an online system.

As described in Section 3.3 and Fig. 5, applications can issue a request to reconfigure the resource partition. In multi-resource environment, we define applications can also issue a request to reconfigure composite resource partitions in Def. 6.1.

Definition 6.1: Reconfiguration Request of Acyclic Regular Composite Resource Partition is defined as $\lambda^c = \{\mathcal{C}^n, \mathcal{A}^n, \mathcal{R}^r, T\}$ where $\mathcal{C}^n$ is the target set of regular composite resource partitions after the request; each effective regular resource partition $P_{j,i}$ with the requested $\alpha_{j,i} \in A_i \in \mathcal{A}^n$ for the composite resource partition C_i has an associated effective reconfiguration supply regularity (see Def. 6.2) of $\overline{R^r_{j,i}} \in \mathcal{R}^r$; T is the maximum time allowed for the reconfiguration to complete.

The same as the definition of effective supply regularity was defined in Section 4.3. Next, we define the reconfiguration supply regularity of a resource partition in multi-resource environment in Def. 6.2.

Definition 6.2: Let e, x_1, x_2 be non-negative integers, $\mathcal{S} = \{s_1, s_2, \cdots, s_n\}$ be the slice offsets of P, p be the period of P and $O = (\{o_1, o_2, \cdots, o_{n'}\}, \overline{p})$ be the request offsets of P. The effective reconfiguration supply regularity $\overline{R^r}$ of a resource partition P is defined as the smallest integer k such that for any $s = s_i + x_1 \cdot p, o = o_j + x_2 \cdot \overline{p}$ and e, where $0 < i \leq n$, $0 < j \leq n'$, $e, x_1, x_2 \in \mathbb{N}$

$$\begin{cases} I(o+e) - I(o) - 1 > -k & \text{if } o \notin \mathbb{N} \text{ and } \exists s = \lfloor o \rfloor \\ I(o+e) - I(o) > -k & \text{otherwise} \end{cases}$$

We can define the effective reconfiguration regular partition as follows.

Definition 6.3: Resource partition P is effective reconfiguration regular if and only if its effective reconfiguration supply regularity $\overline{R^r} = 1$ and it is effective regular before and after the reconfiguration, respectively.

As described in Section 5.3.2, the run time complexity would be exponential without any assumptions and if we are running Alg. 3. For a system with resource reconfiguration capability, the run time complexity for reconfiguration needs to be low to make reconfiguration done in time. Therefore, before formalizing the problem, we first give the following assumptions.

general? specify but why

- Time zero of each physical resource is synchronized.
- No concurrent reconfiguration request is allowed in the system.
- The availability factor of P and the resource slice size of each resource is restricted to be power of $\frac{1}{2}$ and 2, respectively.
- For each resource partition P, the partition before and after the reconfiguration, which is P^o and P^n, are both effective regular but P can be effective reconfiguration irregular.
- We don't schedule resource slice on any request offsets

The composite resource partition may be irregular during reconfiguration as some resource partitions may be temporarily effective irregular but each resource partition will be effective regular after the reconfiguration is done. Also, it is possible to allow concurrent reconfiguration as long as the performance semantics can be specified for each partition. Imposing restriction on the choice of availability factors and resource sizes not only reduces the time complexity but also increases the overall schedulibility, as shown in Section 8. We shall discuss these assumptions in more details in Section 9.

We now formalize the problem as follows.

Problem 6.1: Dynamic Acyclic Composite Resource Partition Reconfiguration (DCRP) problem: Given a composite resource partition reconfiguration request $\lambda^c = \{\mathcal{C}^n, \mathcal{A}^n, \mathcal{R}^r, T\}$ and the tuples of all the resource partitions before the request $\{P^o_{j,i} \mid \exists P^n_{j,i} \in \mathcal{P}^n_i\}$, compute the schedule of $P^t_{j,i}, P^n_{j,i}$ for each composite resource partition C_i such that the following three conditions are satisfied:

C-1: $P^n_{j,i}$ is effective regular with availability factor $\alpha_{j,i} \in A_i \in \mathcal{A}^n$;

C-2: the effective reconfiguration regularity of $P_{j,i}$ is less than $R^r_{j,i} \in \mathcal{R}^r$;

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

6.1 DCRP Algorithm

We present the solution to the DCRP problem in this section by first giving a base line algorithm based on the DPR algorithm (Chen et al. 2019) which trades some performance degradation for simplicity. To further improve the schedulability and mitigate the performance degradation, we revisit the concept of dynamic RRP scheduler and then propose the DCRP algorithm.

6.1.1 Baseline DPR-Based Algorithm

A DCRP problem can be solved by breaking the reconfiguration request down to a set of independent R^{3}Ps. Each R^{3}P then can be handled by the DPR algorithm independently on each physical resource. However, each reconfigured partition P may have its effective supply regularity and effective reconfiguration supply regularity bounded by the following lemma.

Lemma 6.1: Breaking reconfiguration request down into a set of R^{3}P and performing DPR algorithm on each physical resource independently, the effective reconfiguration supply regularity $\overline{R^r} \leq R^r + 1$ and effective supply regularity $\overline{R} \leq R + 1$ where R^r and R are the reconfiguration supply regularity and supply regularity of each partition in the uniform resource RRP model.

The regularity may increase by one due to the resource misalignment problem. To avoid such extra performance degradation, we need to consider the request offsets of each resource partition. However, the request offsets of a resource partition can only be deduced after the schedules of resource partitions with lower

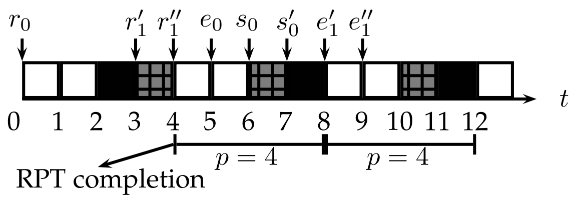


Fig. 10: An example of a dynamic RRP-based partition scheduler where resource slices in black and gray color are two possible schedules. The release time and deadline of the next instance will be based on the previous instance.

rank in the resource access order. Furthermore, each resource partition should supply enough resources (defined by its availability factor, supply regularity and reconfiguration regularity) both during and after the reconfiguration.

6.1.2 Online Dynamic RRP Scheduler

In this section, we revisit the concept of dynamic RRP scheduler and extend the related concept, *maximum supply shortfall*, to non-uniform environment which are first introduced in (Chen et al. 2019).

Maximum Supply Shortfall

Traditional RRP schedulers are mostly static schedulers (Feng 2004; Li and Cheng 2017). The concept of maximum supply shortfall was introduced in (Chen et al. 2019) which enables dynamic RRP scheduler by tracking the supply shortfall of each resource partition. This makes it possible to progressively compute the schedule by limiting the supply shortfall. In this paper, we extend this concept to take the request offsets into consideration.

Definition 6.4: Let $\mathcal{S} = \{s_1, s_2, \cdots, s_n\}$ be the slice offsets, p be the period, $O = (\{o_1, o_2, \cdots, o_{n'}\}, \overline{p})$ be the request offsets of a resource partition P and t be the time where there exists at least one request offset o_j s.t. $t \geq o_j$. Given that no resource slice is scheduled on any candidate of request time, the maximum supply shortfall of P at time t is defined as $d(t)$ such that for any $o = o_j + x_2 \cdot \overline{p}$, where $0 < j \leq n'$ and $x_2 \in \mathbb{N}$

$$d(t) = \min_{\forall o \leq t} (I(t) - I(o))$$

The $d(t)$ denotes the maximum supply shortfall among all the time intervals ending at time t and it takes request offsets into consideration in non-uniform environment. The effective reconfiguration regularity will be the smallest positive integer k such that $d(t) > -k$ $\forall t$.

Dynamic RRP Scheduler

Given the **C-1** and **C-2** requirements and the maximum supply shortfall of a partition at time t, we can compute the deadline to schedule the next slice based on Def. 6.4 and Def. 4.3 in order to preserve the regularity of the partition (Chen et al. 2019).

Let's first review the case where the request offsets include all the integer time points as in uniform environment. The problem of scheduling resource partitions is akin to schedule a set of tasks such that each partition is considered as a task and each task instance indicates a deadline for the partition to schedule the next slice. The following requirements 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 maximum supply shortfall, availability factor and regularity of its associated partition; and (3) each partition follows a cyclic schedule after the RPT stage. Fig. 10 depicts an example of such 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. 10. Notice that each scheduled task instance will map to the resource slice offset of the resource partition. The resource partition in Fig. 10 may have resource slice offset s_0 or s_0' after the reconfiguration. To further take request offsets into consideration, we will also need to take request offsets into consideration by avoiding scheduling resource slices on any request offsets to preserve regularity as discovered in Section 5.

Fast Deadline Computation in Non-uniform Environment Naïvely computing the resource supply shortfall and deadline of each partition based on every request offset and time point will impose great run time complexity. However, we want this online algorithm to be computed fast. In the following, we present the method to update the deadline for each partition P_i in constant time for online reconfiguration by only tracking 1) the maximum supply shortfall $d_i(t)$ at scheduling decision time t and 2) the nearest future request offset $o^t > t$.

Theorem 6.1: Given that no resource slice is scheduled on any candidate of request time, a resource partition P has effective reconfiguration regularity of $\overline{R^r} \leq k$ if for any time t,

$$s^t \leq \min(t + d(t)/\alpha^n, o^t) + k/\alpha^n - 1$$

where o^t is the smallest candidate of request time no less than t and s^t is the smallest resource slice no less than t.

Proof We prove this by showing that for any time t, $d(t) > -k$ by Mathematical Induction and this leads to $I(o + e) - I(o) > -k$ for any candidate of request time o, $e \in \mathbb{N}$ by Def. 6.4. Further by Def. 6.2, P will be effective reconfiguration regular.

For $t < o^0$, both the regularity and $d(t)$ is undefined. The resource slice scheduled before o^0 doesn't affect them. Without loss of generality, we assume

$s^0 \geq o^0$ and start with $t \in [o^0, s^0]$ and $s^0 \leq o^0 + k/\alpha^n - 1$ by the theorem. By Def. 3.5, Def. 6.4 and the fact that there is no slice scheduled between o^0 and s^0, we have $d(t) \geq I(s^0) - I(o^0) \geq -k + \alpha^n > -k$ for $t \in [o^0, s^0 + 1]$.

Assume that $d(t') > -k$ for any time $t' \leq s^t + 1, t < t'$ is true where s^t is the first resource slice no less than t. Let $s^x > s^t$ be the next resource slice after the one at s^t, $o^x > s^t$ be the first candidate of request time after s^t and $t'' \leq s^x + 1$. We proceed to prove that that $d(t'') > -k$ for any time $t'' \leq s^x + 1$.
Case 1:
If $I(o^x) < \max\limits_{\forall o \leq o^x}(I(o))$, by Def. 6.4 and the fact that there is no resource slice scheduled between $[s^t + 1, s^x)$, we have

$$t_1 + d(t_1)/\alpha^n = X \tag{11}$$

and

$$d(t_1) \geq d(s^x) \text{ for } t_1 \in [s^t + 1, s^x] \tag{12}$$

where X is a fixed value. By Def. 6.4, Def. 3.5 and the fact that there is no resource slice scheduled between $[s^t + 1, s^x)$, we have

$$d(s^x) = d(s^t + 1) - \alpha^n(s^x - (s^t + 1))$$

By the theorem that $s^x \leq s^t + 1 + d(s^t + 1)/\alpha^n + k/\alpha^n - 1$, Equ. 11 and $d(s^t + 1) > -k$, we have $d(s^x) >= -k + \alpha^n$. Further by Equ. 12, we have $d(t_1) > -k$ for $t_1 \in [s^t + 1, s^x + 1]$.
Case 2:
If $I(o^x) = \max\limits_{\forall o \leq o^x}(I(o))$, for $t_1 \in [s^t + 1, o^x]$, it is the same as **Case 1**. For $t_1 \in (o^x, s^x]$, $d(t_1) = I(t_1) - I(o^x)$. By Def. 3.5 and the fact that there is no resource slice scheduled between $[o^x, s^x)$, we have

$$d(t_1) \geq d(s^x) \text{ for } t_1 \in (o^x, s^x] \tag{13}$$

From the theorem, we have $s^x \leq o^x + k/\alpha^n - 1$ and thus $d(s^x) = -k + \alpha^n$. By Equ. 13, we have $d(t_1) > -k$ for t_1 in $[o^x, s^x + 1]$.

Combined both cases and Equ. 11, we have $d(t'') > -k$ for any time $t'' \leq s^t + 1$ for any $t < t''$. Notice that, if $d(s^x) > -k$ and there is a resource at s^x, $d(s^x + 1)$ must be greater than $-k$ by Def. 6.4 and Def. 3.5.

By Mathematical Induction, for any t we have $d(t') > -k$ for $t' \leq s^t + 1$ where s^t is the next resource slice after t. □

This theorem states a sufficient deadline for the next resource slice such that the resource partition P is guaranteed to have effective reconfiguration supply regularity less than or equal to K. Intuitively, we can consider the term k/α^n to be the budget that can be used to delay the next resource slice. The term $d(t)/\alpha^n$ is the used budget comes from the supply shortfall until t.

Based on this theorem, we can progressively construct the schedule with effective reconfiguration regularity of $k <= \overline{R^r}$ by 1) scheduling a slice by the deadline and not on any request offset, 2) updating the maximum supply shortfall, the next deadline and repeat these two above steps until we can find a cyclic schedule for each partition for the time after RPT stage.

6.1.3 DCRP Algorithm

In the following, we use $\mathcal{T}_t$ to denote the state of the partition scheduling system at time t, which includes the timestamp of last scheduling decision r_i, the maximum supply shortfall at r_i as d_i, and the deadline e_i to schedule next slice for each partition P_i.

We first present the algorithm overview of the DCRP algorithm in Alg. 5 and then present the required three steps to compute the schedules.

Algorithm 5: Algorithm Overview for the DCRP Problem

```
Input: The reconfiguration request $\lambda^c$, the time $t_r$ of the request and the total
       access order $E^t = \{(\Pi_j, \Pi_k) \mid (\Pi_j, \Pi_k) \in E_i, \forall i\}$.
Output: Transition schedule $\mathcal{S}^t_{j,i}$ and cyclic schedule $\mathcal{S}^n_{j,i}$ for all $P_{j,i}$ on each
        physical resource $\Pi_j$. Reject if no feasible schedule.
1  Enqueue all physical resource $\Pi_j$ into a queue $Q$ by following the topological
    sorted order of $E^t$
2  for $t_b \leftarrow 0$ to $T$ do
3      $Q_t = Q$
4      while $Q_t \neq \emptyset$ do
5          Dequeue $\Pi_j$ from $Q_t$
6          $\mathcal{T}_{t_r}$ = Initialization($\Pi_j, t_r$) // Stage-1
7          $(\{\mathcal{S}^t_{j,i} \mid \forall i\}, \mathcal{T}_{t_r+t_b})$ = TransitionSchedule($\mathcal{T}_{t_r}, t_r, t_r + t_b$) // Stage-2
8          if $\mathcal{T}_{t_r+t_b} = Null$ then
9              break
10         end
11         $\{\mathcal{S}^n_{j,i} \mid \forall i\}$ = CyclicSchedule($\mathcal{T}_{t_r+t_b}$)// Stage-3
12         if $\{\mathcal{S}^n_{j,i} \mid \forall i\} = Null$ then
13             break
14         end
15     end
16     if $Q_t = \emptyset$ then
17         return $(\{\mathcal{S}^t_{j,i} \mid \forall j,i\}, \{\mathcal{S}^n_{j,i} \mid \forall j,i\})$
18     end
19 end
20 return NULL
```

In Alg. 5, we first perform a topology sort based on the resource access order E^t of all the new composite resource partitions to generate a linear order $\mathcal{T}' = \{\Pi'_1; \Pi'_2; \cdots; \Pi'_{n_1}\}$ of all the physical resources. The algorithm then computes the schedules of the partitions on each physical resource by following this linear order. Note that, the algorithm adopts the same budget t_b for all physical resource and tests for schedulability given t_b. For each physical resource, the algorithm has three stages to compute the partition schedules. In stage-1, the **Initialization** procedure will compute the $\mathcal{T}_{t_r}$, which includes the maximum supply shortfall and the deadline of each partition. In stage-2, **TransitionSchedule** will compute the transition schedule based on $\mathcal{T}_{t_r}$ and then compute the $\mathcal{T}_{t_r+t_b}$ for next stage. In stage-3, **CyclicSchedule** for each partition will compute their cyclic schedule based on $\mathcal{T}_{t_r+t_b}$. We will explain each stage of the algorithm as follows:

Initialization In the initialization stage of the partition system, the algorithm aims to compute the $d_i(t_r)$ and the deadline of each partition to adopt Thm. 6.1.

The algorithm first computes the request offset of each partition as $O_{j,i}$ in line 4 by computing the request offsets $O^o_{j,i}, O^t_{j,i}, O^n_{j,i}$ of $P_{j,i}$ for the time before, during and after the reconfiguration based the Alg. 2 and combining them into single $O_{j,i}$. Also, the procedure $\textbf{Next}(O,t)$ computes the next nearest request offset $o \in O$ s.t. $t \leq o$ while the **ComputeMSS** procedure computes the maximum supply shortfall based on Def. 6.4. Note that, the procedure can be improved to be computed in $O(1)$ (Chen et al. 2021). In line 7 and 10, we compute the maximum supply shortfall $d_i(t_r)$ based on two conditions and update the deadline based on Thm. 6.1 in line 8 and 11 accordingly.

Algorithm 6: Partition System Initialization

```
    Input: Physical resource $\Pi_j$ and the requesting time $t_r$
    Output: $\mathcal{T}_{t_r}$, the state of the partition system.
 1  Procedure Initialization($\Pi_i, t_r$)
 2      for $P_{j,i}$ on $\Pi_j$ do
 3          $r_{j,i} = t_r$
 4          $O_{j,i} = \textbf{ComputeAllOffsets}(P_{j,i})$
 5          $n_{j,i} = \textbf{Next}(O_{j,i}, t_r)$
 6          if $P_{j,i}$ is a new partition then
 7              $d_{j,i} = \text{MAX}$
 8              $e_{j,i} = n_{j,i} + R^r_{j,i}/\alpha^n_{j,i} - 1$
 9          else
10              $d_{j,i} = \textbf{ComputeMSS}(P_{j,i}, t_r, O_{j,i})$
11              $e_{j,i} = \min(t_r + d_{j,i}/\alpha^n_{j,i}, n_{j,i}) + R^r_{j,i}/\alpha^n_{j,i} - 1$
12          end
13      end
14      return $\mathcal{T}_{t_r}$
15  Procedure ComputeMSS($P_{j,i}, t_r, O_{j,i}$)
16      $d = \text{MAX}$
17      for All $o \in O_{j,i}$ and $o < t_r$ do
18          $d = \min(d, S_{j,i}(t_r) - S_{j,i}(o) - \alpha^o_{j,i}(t_r - o))$
19      end
20      return $d$
```

Transition Schedule Computation Given a partition system computed in Stage-1 with a time budget t_b to complete the reconfiguration on physical resource ϕ_j, this stage computes the transition schedule for $P^t_{j,i}$ by following three heuristic principles: (1) We avoid scheduling resource slices on any request offset; (2) 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 partitions during the RPT stage; and (3) if the deadline of a partition calculated

Algorithm 7: Transition Schedule Computation

Input: The time of reconfiguration t_r, the budget t_b, and the state of the partition system $\mathcal{T}_{t_r}$.
Output: Transition schedule $\{\mathcal{S}^t_{j,i} \mid \forall j\}$ and $\mathcal{T}_{t_r+t_b}$. Reject if no feasible schedule.

```
1  Procedure TransitionSchedule($\mathcal{T}, t_r, t_b$)
2      for $t_t \leftarrow 0$ to $t_b$ do
3          $m[t_t] = 0$ //initialize the data structure for schedules
4      end
5      Enqueue all partitions $P_{j,i}$ into a queue $Q$ in the ascending order following (1)
         deadline $e_{j,i}$ and (2) period $p_{j,i}$
6      while $Q \neq \emptyset$ do
7          Dequeue $P_{j,i}$ from $Q$
8          $l =$ DS-EDF$(r_{j,i} - t_r, e_{j,i} - t_r, m, t_b, O_{j,i})$
9          if $l = NULL$ then
10             if $e_{j,i} < t_r + t_b$ then
11                 return NULL // Deadline will miss
12             end
13             continue
14         end
15         Add $l - t_r$ to $\mathcal{S}^t_{j,i}$
16         UpdateStates$(\mathcal{T}, t_r + l + 1, P_{j,i})$
17         Enqueue $P_{j,i}$ to $Q$
18     end
19     return $(\{\mathcal{S}^t_{j,i} \mid \forall i\}, \mathcal{T})$
```

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 schedulability of the cyclic schedule construction in Stage-3. Note that the strategy used is a modified version from the reconfigurable RRP model (Chen et al. 2021).

In Alg. 7, the algorithm picks the partition with earliest deadline and schedules it as late as possible using procedure **DS-EDF** as in line 7-8. If no partition is not be able to be scheduled before the deadline, the algorithm will reject in line 10-12. Any time if a partition is scheduled a slice, the algorithm will update its d_i, e_i, r_i using procedure **UpdateStates**.

In the **DS-EDF** procedure in Alg. 8, the resource slice is not only scheduled as late as possilbe but also prohibited from scheduled on any of the request offset. This will avoid the resource slice to skip problem. The procedure **UpdateStates** based on the Thm. 6.1 first updates the d_i from $d_i(r_{j,i})$ to $d_i(r')$ in line 13-16. Based on def. 6.4 and def. 3.9, if there is no request offset between $r_{j,i}$ and r', $d_i(r') = d_i(r_{j,i}) + 1 - \alpha^t_{t,j}(r' - r_{j,i})$ as in line 13. If there exist request offsets between $r_{j,i}$ and r', we only need to consider whether the request offset $n_{j,i}$ recorded from last scheduling decision can give a lower $d_{j,i}(r')$ as in line 15. In line 17-19, it updates the states based on Thm. 6.1.

Cyclic Schedule Computation The Alg. 9 computes the cyclic schedule of each partition to satisfy the following: **C-1**: each partition $P^n_{j,i}$ is effective regular by scheduling resource slices between a pair of neighboring request

Algorithm 8: DS-EDF and update supply shortfall procedures

```
Procedure DS-EDF($r, e, m, t_b, O$)
    $e = \lfloor e \rfloor$
    if $e >= t_b$ then
        $e = t_b - 1$
    end
    for $t_t \leftarrow e$ to $r$ do
        if $m[t_t] = 0$ and ($t_t \mod \overline{p}$) not on any $o \in O$ then
            $m[t_t] = 1$
            return $t_t$
        end
    end
    return NULL
Procedure UpdateStates($\mathcal{T}, r', P_{j,i}$)
    $d_{j,i} = d_{j,i} + 1 - \alpha^t_{j,i}(r' - r_{j,i})$
    if $r' \geq n_{j,i}$ then
        $d_{j,i} = \min(d_{j,i}, 1 - \alpha^t_{j,i}(r' - n_{j,i})))$
    end
    $n_{j,i} = \mathbf{N}ext(O_{j,i}, r')$
    $e_{j,i} = \min(r' + d_{j,i}/\alpha^t_{j,i}, n_{j,i}) + R^r_{j,i}/\alpha^t_{j,i} - 1$
    $r_{j,i} = r'$
    return
```

Algorithm 9: Cyclic Schedule Computation

```
Input: $\mathcal{T}^{t_b}$, the state of the partition system after the RPT.
Output: Cyclic schedule $\{\mathcal{S}^n_{j,i} \mid \forall i\}$. Otherwise, reject.
Procedure CyclicSchedule($\mathcal{T}^{t_b}$)
    Enqueue all $P_{j,i}$ into a queue $Q$ in the ascending order following the (1) period
      $p_{j,i}$ and (2) deadline $e_{j,i}$
    for $t_t \leftarrow 0$ to $p_{max}$ do
        $m[t_t] = 0$
    end
    while $Q \neq \emptyset$ do
        Dequeue $P_{j,i}$
        $l = $ DS-EDF$(0, e_{j,i} - t_r - t_b, m, p_{j,i}, O_{j,i})$
        if $l = NULL$ then
            return NULL
        end
        Add $l$ to $\mathcal{S}^n_{j,i}$
        for $t_t \leftarrow 0$ to $p_{max}/p_{j,i} - 1$ do
            $m[l + t_t \times p_{j,i}] = 1$
        end
    end
    return $\{\mathcal{S}^n_{j,i} \mid \forall i\}$
```

offsets with the corresponding availability factor according to the Theorem 5.2; **C-2**: the effective reconfiguration supply regularity of each partition $P^n_{j,i}$ is guaranteed by Theorem 6.1 where resource slices are scheduled by the deadline computed from the maximum supply shortfall and the fact that $P^n_{j,i}$ has a cyclic schedule with exact one offset. Note that, this algorithm can only be used to

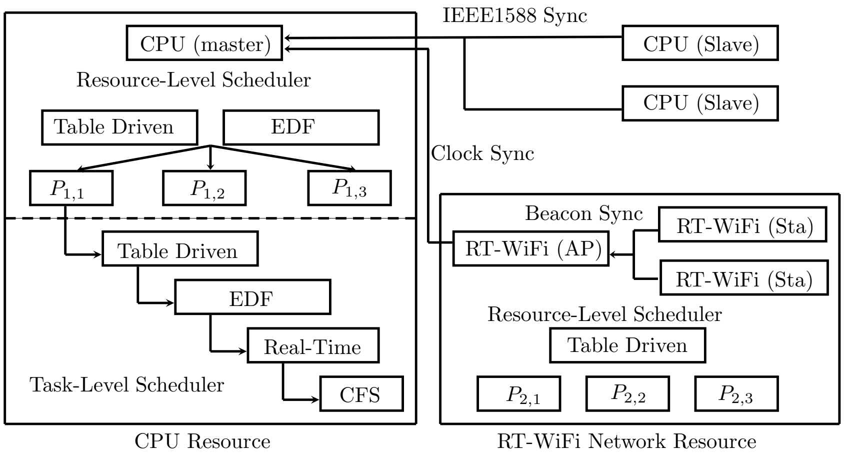


Fig. 11: System architecture of the multi-resource scheduling framework.

compute the schedules when the availability factors and the resource slice sizes are power of $\frac{1}{2}$ and 2, respectively. In Alg. 9 line 8, it finds an available slot for this each partition and marks the cyclic schedule as used in line 13-15. Note that p_{max} is the largest period among all the partitions.

7 Multi-Resource Scheduling Framework Design

To demonstrate the applicability of the concept of regular composite resource partition in practice, we implemented a multi-resource scheduling framework including both CPU and network resources. It is based on a Linux kernel 3.18.12 without multi-core support and combined with a modified version of RT-WiFi (Wei et al. 2013). Fig. 11 illustrates the system architecture. There are one master CPU resource and multiple slave CPU resources. For the network resource, a single RT-WiFi access point (AP) and a cluster of RT-WiFi stations operating on the same channel constitute the WiFi network resources. In the following, we first describe the design and implementation of each component, and then discuss the synchronization mechanism among the components.

CPU Resource: Linux kernel 3.18.12 has EDF (highest priority), real-time and complete fair scheduler (CFS, lowest priority). Each physical CPU has a run queue data structure to manage tasks inside the task scheduler hierarchy. We implemented the hierarchical CPU scheduler by introducing a layer of resource-level scheduler to schedule multiple run queues either by EDF or table driven scheduler. Each run queue represents a resource partition assigned to a CPS application and each application should be able to run its task-level scheduler with its run queue. Moreover, we added a table driven scheduler in the task-level scheduler hierarchy to provide the capability of recursive resource partitioning (Chen et al. 2017). The new hierarchy is illustrated in the master CPU resource diagram in Fig. 11. Note that we do acknowledge that this framework lacks the isolated task group management system for each individual CPS application and also does not take several OS non-real-time characteristics into consideration.

Network Resource: For the network resource, we use RT-WiFi (Wei et al. 2013) to schedule the network channel resource. RT-WiFi is a TDMA-based real-time high-speed wireless data link layer protocol. It provides deterministic timing guarantee on packet delivery with adjustable sampling rates up to $6kHz$. A single-cluster RT-WiFi network consists of one RT-WiFi access point (AP) and multiple RT-WiFi stations.

Synchronization: To schedule regular composite resource partitions, we need to synchronize the clocks and schedules on different physical resources to avoid the resource misalignment problem. In our framework, every slave CPU component synchronizes its clock with the master CPU's clock using IEEE 1588 software implementation (Eidson 2010; ptp 2017). For RT-WiFi network resources, the RT-WiFi AP synchronizes its clock with master CPU component's clock and each RT-WiFi station synchronizes its clock with the associated AP via RT-WiFi beacon frame. The synchronization mechanism and the relation between each component are illustrated in Fig. 11.

8 Performance Evaluation

In this section, we will show three sets of experiments. We first demonstrate the impact of the resource misalignment problem with real cases and evaluation our multi-resource scheduling framework. Two sets of simulation results are then presented for comparing CRP and AAF algorithms (Mok and Alex 2001); and reconfiguration of composite resource partition in multi-resource environment.

8.1 Real System Evaluation

The test-bed has two machines connected with RT-WiFi, providing CPU1, CPU2 and RT-WiFi resources. Machine 1 has an Intel Core i7-4790 CPU and an AR9XX series WiFi card configured as a RT-WiFi access point. Machine 2 has an Intel Core i5-3337U CPU and an AR9XX series WiFi card configured as a station. We emulate a periodic end-to-end task T for an application A_1 and measure its end-to-end response time as our evaluation metric. The applications A_1 will access CPU1, WiFi, CPU2 resources in a sequence and the system constructs a regular composite resource partition C_1 for it. The task T periodically processes the sensor data on CPU1, passes the data to CPU2 and finally process the data to pass the decision to some actuator on CPU2. The task consumes one resource slice on each resource.

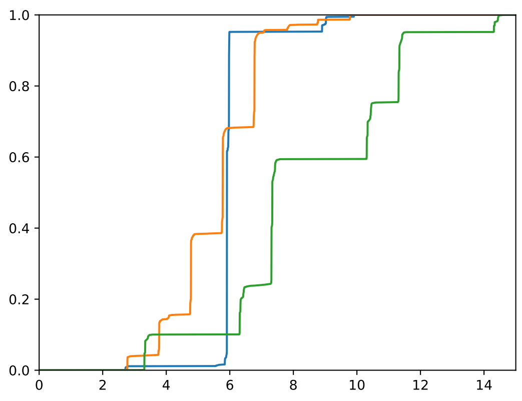


Fig. 12: TODO:Format figure and legend. The Uniform and Misaligned cases highlight the misaligned problem. The Non-Uniform case demonstrates the applicability of the framework and the CRP algorithm.

8.1.1 Resource Misalignment Problem

To create the resource misalignment problem, the resource slice size of CPU1 and CPU2 are set to $1ms$ and the resource slice size of the RT-WiFi is set to $1024\mu s$. The result is denoted as the Misaligned in Fig. 12. To compare the Misaligned result with the aligned result, another experiment in which all the resource slice size are synchronized and set to $1ms$ is denoted as Uniform in Fig. 12. The task execution time taken on CPU1, RT-WiFi and CPU2 is calibrated such that it takes slightly less than one resource slice on each resource. The constructed regular resource partitions $P_{1,1}, P_{2,1}, P_{3,1}$ on CPU1, RT-WiFi and CPU2 have availability factor of $0.25, 0.25, 1$, respectively. The theoretical maximum response time of each task instance would be $1ms$ on CPU1 as the time it get the reading and pass the data to RT-WiFi, $1ms/0.25$ (or $1024\mu s/0.25$ for the misaligned case) on RT-WiFi to pass the data including the wait time; and $1ms/1$ on CPU2. The total would be roughly $6ms$. The Misaligned case shows that 20% task instances will have response time larger than the theoretical one while the Uniform case shows only 2.5% task instances. This simple experiment shows the importance of synchronizing the resource partitions even in the uniform environment.

8.1.2 Non-uniform Environment

We next demonstrate a case how CRP can be applied to real-world systems in non-uniform environment. The resource slice sizes of CPU1 and CPU2 are $1ms$ while the resource slice size of RT-WiFi is $2ms$. Two regular composite resource partitions C_1, C_2 are constructed. C_1, which is assigned to the appli-

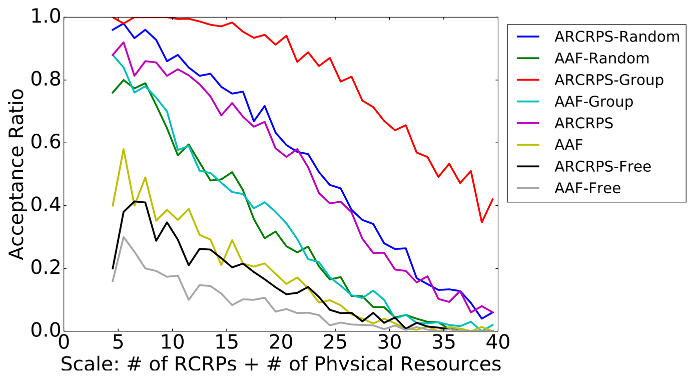


Fig. 13: TODO redo the graph for consistency [WJ] Comparison of acceptance ratio: ARCRPS vs. AAF. The ARCRPS, AAF, ARCRPS-Free and AAF-Free are non-uniform multi-resource environment. Others are uniform environment.

cation A, has effective regular resource partitions $P_{1,1}, P_{2,1}, P_{3,1}$ all with availability factor of 0.25. C_2 has effective regular resource partitions $P_{1,2}, P_{2,2}, P_{3,2}$ with availability factor of $0.25, \frac{1}{3}, 0.25$, respectively. The theoretical maximum response time of each task instance would be $1ms$ on CPU1, $(1 \cdot 2ms)/0.25$ on RT-WiFi to and $1ms/0.25$ on CPU2 with a total of $13ms$. As shown in Fig. 12 as the Non-Uniform case, 95% of the task instance has response time less than 13 ms.

8.2 CRP Algorithm

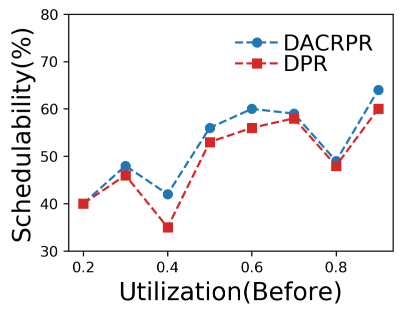
In the followings, we presents three sets of simulation result for comparing CRP-offline, CRP-online and AAF algorithms in different settings.

Figures:

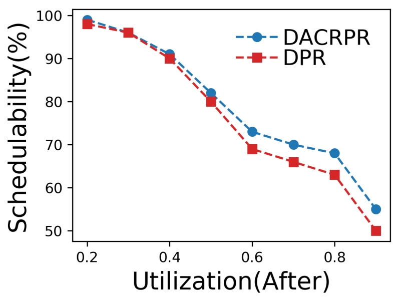
– Uniform(AAF, ARCRP): linear alpha, harmonic alpha for each application
– Nonuniform(AAF, ARCRP): linear alpha/phy, harmonic all,
– online vs offline: harmonic

8.3 DCRPAlgorithm

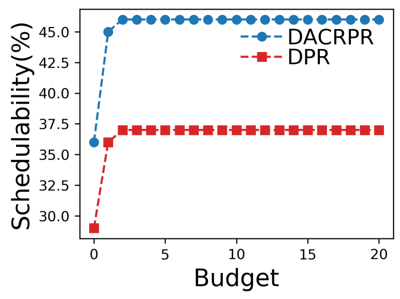
In this section, we compares the results of reconfigure composite resource partition using DCRP algorithm with the DPR algorithm in non-uniform environment. The experiment is conducted by generating X testing samples in each setting and evaluating the acceptance by checking the regularity of all the partitions. If the reconfigured partition have the required regularity, the sample is accepted in terms of schedulability. In each sample, there are X regular composite resource partitions and Y physical resource. Also, each composite resource partition is assumed to be able to be constructed by Alg. 4 without considering



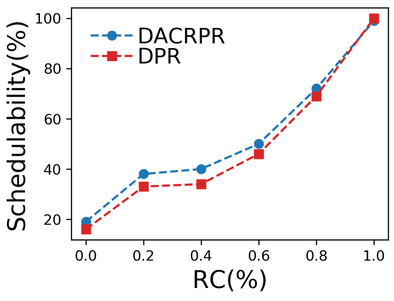
(a) Varying utilization (Before)



(b) Varying utilization (after)



(c) Varying budget



(d) Varying RC

Fig. 14: Schedulability of DCRP and DCRP-DIR

the reconfiguration before the reconfiguration and after the reconfiguration by the CRP-online algorithm because we are evaluating the reconfiguration algorithm. The availability factors of each individual partition before and after reconfiguration are randomly sampled in the set of $\frac{1}{2^i}$ $(1 \leq i \leq 7)$ with total of 80% resource utilization. The effective 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 $[10, 15]$.

Fig. 14a, Fig. 14b?

Increasing the budget for reconfiguration leads to the increase of the schedulability as shown in Fig. 14c (e). However, the schedulability stops increasing past 4 and this indicates that setting the budget to a very high value will only waste computation time. Fig. 14d shows the percentage of partitions with effective reconfiguration regularity over 1 affects the schedulability. Intuitively, the effective reconfiguration regularity denotes the tolerance of performance degradation during reconfiguration. When all the partitions have effective reconfiguration regularity equals or more than 1, the schedulability will achieve 100%. This is because the partitions are assumed to be able to be constructed by the Alg. 4 without considering the reconfiguration.

Clearly, the DCRP algorithm has better schedulability than DPR algorithm because DPR algorithm doesn't take request offset into consideration.

9 Conclusion

In this paper, we study the resource misalignment problem where resource partition may undersupply when resource partitions are not aligned. The concept of composite resource partition is introduced to address this problem and we further propose several algorithms to schedule and reconfigure composite resource partitions. The key ideas are to avoid scheduling resource slice on any candidate of request time and to progressively construct the schedules on each physical resource. These ideas are also applied to reconfigurable RRP model and extend it to non-uniform multi-resource environment. Extensive experiments are conducted including a framework featuring CPU and RT-WiFi network. The framework shows that aligning the resource partitions while considering the request offsets, the system can avoid the unexpected prolonged response time which may leads to deadline misses. The simulation also gives several insights regarding the CRP, DCRP algorithms in different settings.

10 table

Notation	Definition
τ	physical time
Π	the physical resource
P	a set of resource slices
t	the physical resource time
τ	a function of the physical time
Q	the resource slice size of Π
$\mathcal{S}$	*(the sliced offsets)* allocated to the partition (check Def. 3.6)
p	the partition period
$S(t)$	the supply function of P in interval $[0, t)$
α	the availability factor
$I(t)$	the instant regularity
R	the supply regularity
P^o	the old stage (before reconfiguration)
P^t	the transition stage as (during RPT stage)
P^n	the new stage (after reconfiguration)
$\mathcal{P}^n$	the target set of resource partitions after the request
P_i^n	each resource partition $\in \mathcal{P}^n$ has an associated α_i^n
P_i	Why this (Check Def.3.10)
T	the maximum time allowed for the reconfiguration to complete
R^r	the reconfiguration supply regularity of resource partition P
T	A periodic end-to-end task (Check again!)
$\mathcal{PI}$	a sequence of physical resources
$\mathcal{X}$	the corresponding requested amount of resource slices
p	the period (Check again!)
d	the relative deadline
A	applications
Q	the resource slice sizes
$P_{1,1}$	regular resource partitions (Check again!)
$\alpha_{1,1}$	the availability factor (Check again!)
$O = (\mathcal{O})$	a set of n' time points when the resource may be requested (the *request offsets*)
$p(\overline{p})$	the offsets period
$\overline{R}$	the effective supply regularity
e	Definition 4.3 (Check again!)
C	a composite resource partition
$\mathcal{P}$	a set of resource partitions
Π^c	a set of physical resource (Check again!)
E	a binary relation on Π^c; a collection of composite resource partition on a set of physical resource with a predetermined resource access order
$\alpha_{j,i}$	the fraction of resource from Π_j and the resource access order E_i of each application A_i
Π_j	the resource access order
$\mathbb{S}_i$	valid schedules
$\overline{p}$	the request offsets period
E^t	the total resource access order E^t; the union of the access order of all the application A_i
$\mathcal{T}'$	a linear order for all the physical resources.
$\mathcal{A}^n$	(Check Again!)
$\mathcal{C}^n$	the target set of regular composite resource partitions after the request
C_i	the composite resource partition
$\mathcal{T}$	(Check Again!)
λ^c	a composite resource partition reconfiguration request
O	the request offsets of P (Check Again!)
$d(t)$	the maximum supply shortfall
n_i	the nearest future request offset

References

(2017) Precision Time Protocol Daemon. `http://sourceforge.net/projects/ptpd/`

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

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

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 Computing Surveys (CSUR) 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 Transactions on Industrial Informatics 7(2):302–315

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

Chen WJ, Wu P, Huang PC, Mok AK, Han S (2019) Online reconfiguration of regularity-based resource partitions in cyber-physical systems. In: 2019 IEEE Real-Time Systems Symposium (RTSS), IEEE, pp 495–507

Chen WJ, Wu P, Huang PC, Mok AK, Han S (2021) Online reconfiguration of regularity-based resource partitions in cyber-physical systems. Real-Time Systems pp 1–44

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

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

Eidson JC (2010) Measurement, control, and communication using IEEE 1588. Springer

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

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

Han S, Chen D, Xiong M, Lam KY, Mok AK, Ramamritham K (2012) Schedulability analysis of deferrablescheduling algorithms for maintainingreal-time data freshness. IEEE Transactions on Computers 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 Transactions on Embedded Computing Systems (TECS) 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

Jiang Z, Audsley N, Dong P, Guan N, Dai X, Wei L (2019) Mcs-iov: Real-time i/o virtualization for mixed-criticality systems. In: 2019 IEEE Real-Time Systems Symposium (RTSS), IEEE, pp 326–338

Lee J, Chwa HS, Phan LT, Shin I, Lee I (2017) Mc-adapt: Adaptive task dropping in mixed-criticality scheduling. ACM Transactions on Embedded Computing Systems (TECS) 16(5s):163

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

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 Transactions on Embedded Computing Systems (TECS) 16(4):111

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

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

Mok AK, Rosier L, Tulchinsky I, Varvel D (1989) Algorithms and complexity of the periodic maintenance problem. Microprocessing and Microprogramming 27(1):657–664

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-realtime system. In: 29th Euromicro Conference on Real-Time Systems (ECRTS), Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik

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

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 systems 26(2):161–197

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)

Wei YH, Leng Q, Han S, Mok AK, Zhang W, Tomizuka M (2013) Rt-wifi: Real-time high-speed communication protocol for wireless cyber-physical control applications. In: Real-Time Systems Symposium, IEEE, pp 140–149

Xu H, Burns A (2019) A semi-partitioned model for mixed criticality systems. Journal of Systems and Software 150:51–63