Regular Composite Resource Partitioning and Reconfiguration in Open Systems

WEI-JU CHEN, The University of Texas at Austin, USA
PENG WU, University of Connecticut, USA
PEI-CHI HUANG, The University of Nebraska at Omaha, USA
ALOYSIUS K. MOK, The University of Texas at Austin, USA
SONG HAN, University of Connecticut, USA

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 resources are implicitly assumed to be synchronized and clocked at the same frequency. The challenge is that a task utilizing multiple resources may experience unexpected delays in non-uniform environments, where resources are clocked at different frequencies. 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 Scheduling (ARCRP-S)* algorithm constructs regular composite resource partitions and the *Acyclic Regular Composite Resource Partition Dynamic Reconfiguration (ARCRP-DR)* algorithm reconfigures the composite resource partitions in the run time upon requests of partition configuration changes. Our experimental results show that compared with state-of-the-art methods, ARCRP-S can prevent unexpected resource supply shortfall and improve the schedulability up to 50%. On the other hand, ARCRP-DR can guarantee the resource supply during the reconfiguration with moderate computational overhead.

CCS Concepts: • **Computer systems organization** → **Real-time systems**; **Embedded and cyber-physical systems;**

Additional Key Words and Phrases: Regularity-based resource partition, composite resource, dynamic reconfiguration, open systems

The work is supported in part by the National Science Foundation Grant CNS-1932480, CNS-2008463, CCF-2028875, Nebraska Research Initiative Grant NRI-4103080440, and Office of Naval Research under ONR Award N00014-17-1-2216.
Authors' addresses: W.-J. Chen (corresponding author) and A. K. Mok, The University of Texas at Austin, 2317 Speedway, Austin, TX 78712, USA; emails: {albertwj, mok}@cs.utexas.edu; P. Wu and S. Han, University of Connecticut, 371 Fairfield Way, Unit 4155, Storrs, CT 06269, USA; emails: {peng.wu, song.han}@uconn.edu; P.-C. Huang, The University of Nebraska at Omaha, 172 Peter Kiewit Institute, 1110 South 67th Street Omaha, NE 68182, USA; email: phuang@unomaha.edu.

1539-9087/2023/09-ART84 $15.00
https://doi.org/10.1145/3609424

ACM Reference format:
Wei-Ju Chen, Peng Wu, Pei-Chi Huang, Aloysius K. Mok, and Song Han. 2023. Regular Composite Resource Partitioning and Reconfiguration in Open Systems. *ACM Trans. Embedd. Comput. Syst.* 22, 5, Article 84 (September 2023), 29 pages.
https://doi.org/10.1145/3609424

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 [1, 2], there does not exist 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 its 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 [3–5]. In the RRP model, the resource supply is characterized in two dimensions where the *availability factor* defines the resource supply rate and the *supply regularity* defines the deviation of the allocated resource supply from the ideal resource supply. 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. Taking CPU resource as an example, Figure 1 gives an overview of the hierarchical resource scheduling model. In this example, individual applications utilize their resource partitions 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. In this way, 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 characterize the resource usage interface is the **Periodic Resource Model (PRM)** [6] (or its variant the **Explicit Deadline Periodic (EDP)** model [7]). These models characterize the resource interface as a periodic execution budget and its 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 RRP and EDP models is illustrated in Figure 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 bounds 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 application, but at a slower rate ($\frac{1}{4}$) as depicted in Figure 2(c). By considering the resource supply jitter, the resource supply specified by the resource interface under the RRP model can better approximate the ideal supply which is uniform over any time interval. Hence, changes to the task group can be more easily accommodated by the application's own task scheduler by rescheduling tasks within its allocated resource partition. This is possible as long as the application's task utilization remains below the assigned availability factor, thus avoiding the need to change the resource interface [3, 8]. 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 partition and give polynomial-time solutions. For example, the **Adjusted Availability Factor (AAF)**

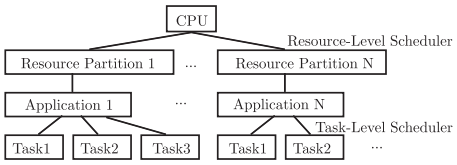


Fig. 1. Overview of the hierarchical scheduling model.

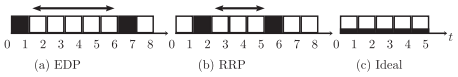


Fig. 2. (a) and (b): two possible schedules under the EDP and RRP models with a required bandwidth of $\frac{1}{4}$. (c): ideal resource supply from the application's point of view.

algorithm allocates resource partitions with availability factors (supply rate) of power of $\frac{1}{2}$ [3]. Li and Cheng [5, 9] proposed the use of a combination of Magic7 and PFair algorithms [9, 10] to improve the resource utilization overhead.

A notable limitation of previous work on the RRP model is that they implicitly assume that resources are clocked at the same frequency for both single-resource [3, 11, 12] and multi-resource environment [9], in which the minimal intervals (*resource slices*) assigned to each task for execution on each physical resource are the same. In multi-resource environments, however, the size of resource slices may be non-uniform across different types of physical resources. This difference in the clock frequencies may introduce unexpected delays for the end-to-end tasks, which may access multiple physical resources in a sequential fashion (see Section 4.1 for the formal definition of an end-to-end task).

In this paper, we first present the challenges in solving the above *resource misalignment problem*, and introduce a novel *composite resource partition* abstraction for non-uniform multi-resource environments. Based on this resource interface, the **Acyclic Regular Composite Resource Partition Scheduling algorithm (ARCRP-S)** and **Acyclic Regular Composite Resource Partition Dynamic Reconfiguration (ARCRP-DR)** algorithm are proposed to construct and reconfigure the composite resource partitions, respectively. The key idea behind the ARCRP-S algorithm is to consider the requesting time of each resource partition. The resource misalignment problem then can be mitigated by scheduling resource partitions in a way that resource will not be requested in the middle of any resource slice. The ARCRP-DR algorithm is built on top of the reconfigurable RRP model that we introduced in our recent work [12] by taking both the resource requesting time and the performance degradation into consideration during the reconfiguration of composite resource partitions. Finally, we evaluate the model with a real-world multi-resource system. Extensive simulation results are also presented to give a thorough evaluation on the proposed algorithms under more general settings.

2 RELATED WORK

The concept of regularity was first introduced by Shirero et al. [13] and was then extended to the RRP model by Mok and Feng [11], aiming to distribute resource evenly on each resource partition by specifying the regularity. Mok and Feng further introduced the irregular partition and presented the **Adjusted Availability Factor (AAF)**-based scheduling algorithm to schedule regularity-based resource partition in single-resource environments [3, 11]. Li and Cheng extended the AAF-based scheduling algorithm to uniform multi-resource environment and developed an optimized partitioning algorithm [5, 9]. More recently, the RRP model was further extended to support online reconfiguration of resource partitions [12] in single resource environments. Besides the RRP model, many other studies on hierarchical scheduling characterize resource interfaces using different models [4, 6, 7]. The most popular model among them is the **Explicit Deadline Periodic (EDP)** model [7] which introduces a relative deadline parameter based on the **Periodic Resource**

Model (PRM) model [6]. Most aforementioned work only discusses the resource allocation in uniform-resource environment (either single- or multi-resource). However, in multi-resource environments, the resource slice sizes may be non-uniform across different physical resources, and new resource interface and scheduling methods have to be designed.

Several works on resource scheduling under the multi-resource models have been proposed in recent years. Some of them apply resource reservation techniques based on processing capacity to achieve performance isolation on multicore architectures by a set of virtual processors with different speeds [14, 15]. For example, Buttazzo et al. [16, 17] 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. Some other research efforts were devoted on workload-partitioning techniques for heterogeneous computing systems which enable exploiting both CPU and GPU to improve resource utilization and increase high-performance computation. Two excellent review papers [18, 19] provided an overview and comparisons of the well-known techniques for such systems. Among these techniques, some of them focus on scheduling workloads on the appropriate device on the same die (*i.e.*, integrated CPU and GPU processors) which shares a total power budget and have strong thermal interactions [20–23]. Some work proposed device-contention-aware scheduling schemes that take the run-time conditions on CPU and GPU processors into consideration [24, 25]. There are also some work that focuses on cache-related research on multicore virtualization platforms [26, 27]. However, most aforementioned research studies did not consider non-uniform environment which may result in resource misalignment problem.

Chen et al. introduces the concept of composite resource partitions to address the resource misalignment problem in non-uniform multi-resource environment [28]. However, composite resource partitions can not be reconfigured with the single-resource online reconfiguration algorithm [12]. The resource misalignment problem will be severe if the resource partitions on different physical resources are reconfigured without any coordination. In this paper, we propose the online algorithm to reconfigure the composite resource partitions in non-uniform multi-resource environments.

To adapt a system to schedule tasks with varying timing requirements, a range of works have been developed. For example, Burns and Davis [29] presented a survey on mixed-criticality systems, in which the task specifications depend on the system state/criticality. Jiang et al. [30] proposed a mixed-criticality system for timely handling of I/O. It provides temporal and spatial isolation and prohibits fault propagation with small overhead based on hardware-assisted virtualisation to offer good timing predictability. Many multi-mode system designs are proposed to ensure that the mode switch is performed in a timely and safe manner in response to both internally and externally generated events [31–38]. 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. For example, Neukirchner et al. [31] introduced a packing solution and a scheduling algorithm based on both mode changes and criticalities for each processor; the dynamic budget management schemes were proposed to postpone criticality mode changes [34, 36]; the different execution-time servers under the control of a hypervisor were employed to bound the overheads when mode changes [33, 35]; Chen and Phan [37] provided a system to analyze and evaluate the mode-change protocols. However, the existing solutions did not consider the reconfiguration of resource abstraction in non-uniform multi-resource open system environment.

There are also some research works on the multi-mode resource interface and platform design [35, 39–42] where the resource interface may change in the run time for single-resource

environment. For instance, Evripidou and Burns [35] used a two-level scheduler or a hypervisor to handle the criticality mode change. Phan et al. [39] proposed a compositional analysis of the multi-mode resource interface. Li et al. [40] used **virtual machine (VM)** to support multi-mode virtualization where the VM parameters change with minimum transition latency. Paluri et al. [42] proposed a VM scheduler prototype based on the RRP model and implemented it on Xen's x-86 based hypervisor. In this paper, we focus on the scheduling and reconfiguration of resource interfaces in non-uniform multi-resource environment.

3 RRP MODEL IN UNIFORM ENVIRONMENT

This section revisits the RRP model in the uniform environment. We first define the time systems used in this paper, and review the concepts of RRP model in the uniform environment [3, 11, 12, 28].

3.1 Time Systems

In this paper, we use two time systems in the RRP model. The first one is the wall clock time defined as the *physical time* τ, which is synchronized among all physical resources (see Figure 3(a)). For the physical resource Π, a minimum non-preemptible physical time interval (2 in the example in Figure 3) is defined as a *resource slice* and allocated to an application exclusively. Physical resource is allocated to the application(s) in units of resource slices (see Figure 3(b)), where a *resource partition* P is a set of resource slices. The second time system, *physical resource time*, is defined as follows.

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

In 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. As shown in Figure 3, non-negative integer t denotes a time at the boundaries of resource slices. 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 revisit the formal definition of a regularity-based resource partition in the uniform environment [3].

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

Definition 3.3. 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 the resource partition P in $[0, t)$. As an example, the resource partition P in Figure 3 is $(\{s_1 = 0, s_2 = 2, s_3 = 4\}, 5)$. Its supply function has $S(1) = 1, S(2) = 1, S(3) = 2, S(4) = 2$, and so on.

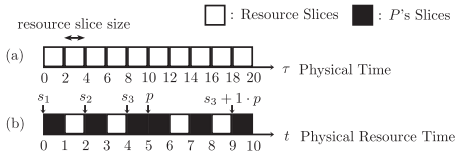


Fig. 3. (a) Physical resource Π is divided into units of resource slices with a minimum physical time interval of 2. Resource partition P is a set of resource slices allocated to an application. (b) Physical resource Π with physical resource time.

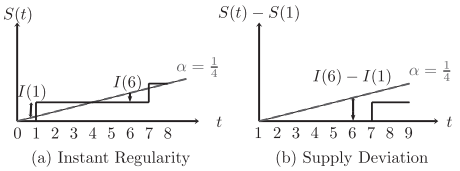


Fig. 4. Illustration of the concepts of availability factor, instant regularity and supply regularity in RRP model.

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 $I(t) = S(t) - \alpha \cdot t$.

Instant regularity $I(t)$ quantifies the gap between the ideal supply and actual supply at time t. The difference in instant regularity at two time instants represents the gap between the ideal supply and actual supply for that time interval.

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 $|I(b) - I(a)| < k, \forall b \geq a$.

The regularity defines the maximum supply deviation from the ideal resource supply. Regularity of one means that the resource supply will never undersupply or oversupply more than one unit of resource.

Figure 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 shown using 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. Figure 4(b) illustrates the actual resource supply for time interval $[1, t)$ as $S(t) - S(1)$, and $I(t) - I(1)$ is the supply deviation in this time interval. For example, $I(6) - I(1)$ is the supply deviation in time interval $[1, 6)$. The supply regularity defines the maximum supply deviation for **all time intervals**.

Definition 3.7. 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.

Recall that the actual resource supply function $S(t)$ is a staircase function while the ideal resource supply is a linear function as illustrated in Figure 4(a). The supply regularity bounds the supply deviation between the two functions and is thus not possible to be zero in a practical system. Moreover, a regular partition, which has regularity of 1, has the following nice property. The utilization bounds for both fixed-priority scheduling and dynamic-priority scheduling of tasks running on a regular partition remain the same as if the task group were running on a dedicated resource

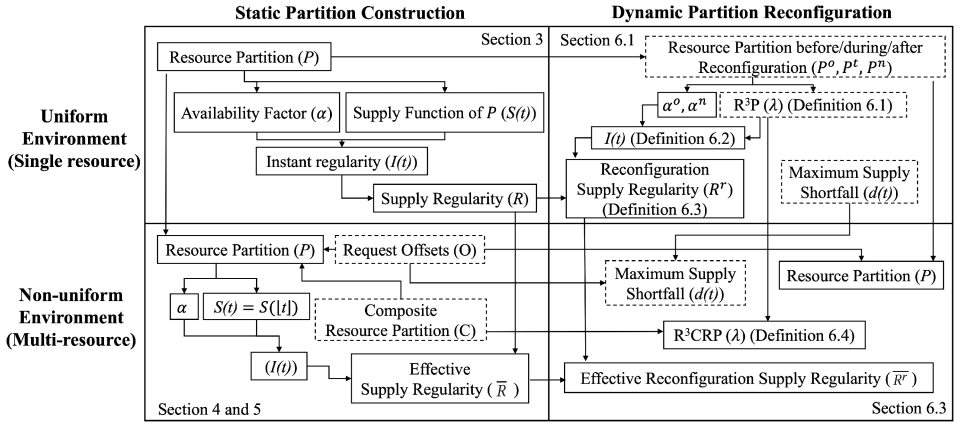


Fig. 5. The relationship among important concepts in partition construction and reconfiguration under different environments.

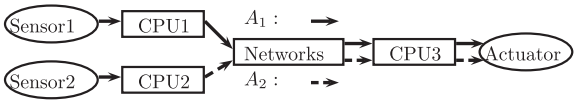


Fig. 6. Solid and dash lines represent two end-to-end wireless networked control applications A_1 and A_2 sharing CPU and wireless network channels, respectively. A_1 has an end-to-end task K_1 and A_2 has an end-to-end task K_2.

whose supply rate is the same as that of the regular partition [3, 8]. Thus, a task group scheduled on a regular partition with either scheduling policy cannot distinguish whether it is scheduled on a resource partition or a dedicated resource with the same rate. This is, however, not true for a task group scheduled on an irregular partition which has a regularity larger than 1. Irregular partitions will introduce larger jitter although this might be acceptable for some applications. In this paper, we focus on regular partitions. Table 1 summarizes the frequently used symbols in this paper. Figure 5 illustrates the relationship among important definitions and categorizes them into four quadrants. For example, the top-left quadrant denotes the definitions used for static partition construction in the uniform environment.

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

4 COMPOSITE RESOURCE PARTITION: CHALLENGES AND MODEL EXTENSION

4.1 Challenges in Non-uniform Multi-resource Environment

In uniform environments, tasks are assumed to only access resource at the resource slice boundaries. However, this assumption may not hold in a practical system, especially in multi-resource environments where resources have different sizes of resource slices. In such environments, each application may have tasks utilizing multiple resources in a periodic and sequential fashion. We define such periodic end-to-end task as follows:

Definition 4.1. A periodic end-to-end task K is defined as $(\mathcal{I}, \mathcal{X}, p, d)$ where $\mathcal{I} = \{\Pi_1; \Pi_2; \cdots; \Pi_n\}$ is a sequence of n physical resources that K will access in sequence, $\mathcal{X} =$

Table 1. Summary of Important Notations and Definitions

Notation	Definition
τ, Π, t	Physical time; physical resource; physical resource time
P, Q	The resource partition; the resource slice size of Π
$(\mathcal{S}, p)$	The start time of the resource slices (offsets) allocated to the partition; the partition period
$S(t)$, α	The supply function of P in interval $[0, t)$; the availability factor
$I(t)$, R	The instant regularity; the supply regularity
$K = (\mathcal{I}, \mathcal{X}, p, d)$	A periodic end-to-end task K; a sequence of n physical resources $\mathcal{I}$; the corresponding requested amount of resource slices $\mathcal{X}$; the period of K; the relative deadline of K
$O = (\mathcal{O}, \overline{p})$	$\mathcal{O}$ is a set of n' time points when the resource may be requested (called the request offsets); $\overline{p}$ is their period
$\overline{R}$	The effective supply regularity
$C = (\mathcal{P}, \Pi^c, E)$	A composite resource partition C; a set of partitions $\mathcal{P}$; a set of physical resource Π^c; a binary relation E on Π^c (the resource access order of C)
Π_j, $O_{j,i}$, $\alpha_{j,i}$, A_i	Each physical resource Π_j; the requested offset $O_{j,i}$ of $P_{j,i}$; the requested resource supply rate $\alpha_{j,i}$; each application A_i
P^o, P^t, P^n	Resource Partition before reconfiguration; during RPT stage; after reconfiguration
$\lambda = \{C, \mathcal{F}, \mathcal{R}, T\}$	Reconfiguration request of regular composite resource partition λ; the set of all regular composite resource partitions C; each effective regular resource partition $P_{j,i}$ has the requested resource $\alpha_{j,i} \in \mathcal{F}$; an associated effective reconfiguration supply regularity $\overline{R^r_{j,i}} \in \mathcal{R}$; the maximum complete time T for reconfiguration
$\overline{R^r}$	The effective reconfiguration supply regularity $\overline{R^r}$ of partition P
$d(t)$	The maximum supply shortfall among all the time intervals ending at time t
s, o, $\mathcal{T}_t$	The resource slice starting time and requesting time; the state of the partition system at time t
q, b, r, e, d	The time of reconfiguration request; the budget; release time; deadline time; the maximum supply shortfall

$\{x_1; x_2; \ldots; x_n\}$ is the corresponding requested amount of resource slices, p and d are the period and the relative deadline of K, respectively, in units of physical time.

Figure 6 gives an example of two end-to-end wireless networked control applications in multi-resource environment. Let physical resource CPU1, CPU2, CPU3 and Networks denoted as Π_1, Π_2, Π_3 and Π_4, respectively. Application A_1 has an end-to-end task K_1 with $\mathcal{I}_1 = \{\Pi_1; \Pi_4; \Pi_3\}$. Application A_2 has an end-to-end task K_2 with $\mathcal{I}_2 = \{\Pi_2; \Pi_4; \Pi_3\}$. In this paper, we focus on the resource-level scheduler and the resource supply of the constructed partitions instead of the task-level scheduler. Thus, in the following, we will assume that there is only one task in each application so there is no task-level scheduler. The response time of this single task running on the resource partition can represent the actual resource supply it received. Given a regular resource partition with availability factor α, we can derive the maximum response time t for the task requesting x resource as follows. $t - 1$ is the time that the task is about to receive the last one unit of resource slice. Assuming this time interval is $[a, b)$, we can have $S(b) - S(a) = x - 1$ by Definition 3.3. Further by Definition 3.6, Definition 3.5, and the fact that the partition is regular, we have $|(x - 1) - \alpha(t - 1)| < 1$. This leads to the conclusion that the maximum response time is $\lceil x/\alpha \rceil$ because both x and t are integers. In this paper, we use the upper bound of the response time $\lceil x/\alpha \rceil$ of the single task running on the partition to measure whether the partitions can provide enough resource supply to highlight the resource misalignment problem. For a task utilizing multiple resource, the response time is the sum of the time spent on each resource partition.

Figure 7 shows a partition schedule for application A_1 where its task K_1 experiences the resource misalignment problem. 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. The black 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 assigned to A_1, respectively. Note that for ease of presentation, we use $P_{j,i}$ to denote the partition on physical resource Π_j assigned to A_i. These three regular partitions have availability factor $\alpha_{1,1} = \frac{1}{8}, \alpha_{4,1} = \frac{1}{4}, \alpha_{3,1} = \frac{1}{2}$,

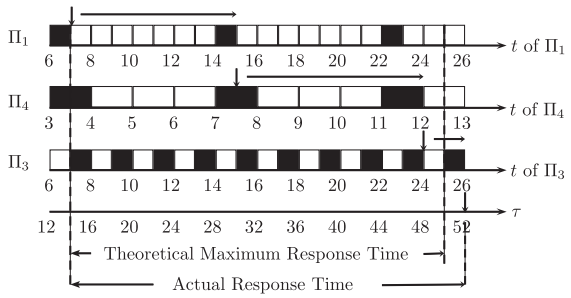


Fig. 7. The vertical and horizontal arrow denote the resource request time and the response time, respectively, of a task instance running on each partition. This instance may miss the deadline because of the unexpected delay for time interval $[7.5, 8)$ on Π_4.

respectively. Let's consider an instance of task $K_1 = (\{\Pi_1; \Pi_4; \Pi_3\}, \{x_1 = 1; x_4 = 1; x_3 = 1\}, 36, 36)$ of A_1 where the task has a period and relative deadline of 36 in physical time. The vertical and horizontal arrow denote the resource request time and the response time, respectively, of a task instance running on each partition. Given the resource supply of these partitions, the theoretical maximum response time of such task instance is 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 execute in the middle of an resource slice of Π_4. This prolongs the expected finishing time on Π_4 from physical time 46 to 48 and prolongs the end-to-end response time to 38 in physical time. This causes the instance to miss the deadline.

Compensating the loss of resource supply due to the 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 needs to schedule one more resource slice for each allocated resource slice. In the worst case, the system needs to compensate a partition with α of $\frac{1}{2}$ with another $\frac{1}{2}$ of the resource to make up for the resource misalignment problem. For the latter approach, the deadline of each task needs to be decreased for each misaligned partition and this may cause the system unschedulable. In this work, we mitigate the resource misalignment problem by judiciously scheduling the resource partitions in a way that no task will request resource at the middle of a resource slice of its resource partition.

4.2 RRP Model Extension

To deal with the above challenge, we introduce the concept of composite resource partition by extending the RRP model with the new concept of effective supply regularity to address the resource misalignment problem. We first extend the definition of a resource partition with the *request offsets* to take the resource requesting time into consideration.

Definition 4.2. A resource partition P on a physical resource Π in non-uniform environment is a tuple $(\mathcal{S}, O, p)$, where $\mathcal{S} = \{s_1, s_2, \ldots, s_n : 0 \leq s_1 < s_2 < \cdots < s_n < p\}$ is a set of n time points that denote the start time of the resource slices (called the *slice offsets*) allocated to the partition, and $O = (O = \{o_1, o_2, \ldots, o_{n'}, : 0 \leq o_1 < o_2 < \cdots < o_{n'} < \overline{p}\}, \overline{p})$ 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 $[s_k + x_1 \cdot p,\ s_k + 1 + x_1 \cdot p)$, $x_1 \in \mathbb{N}, 1 \leq k \leq n$ and resource may only be requested at any *requesting time* $o = o_k + x_2 \cdot \overline{p}$, $x_2 \in \mathbb{N}, 1 \leq k \leq n'$.

In order to take the resource misalignment problem into consideration, the partition in non-uniform environment now is defined with the consideration of the time points (request offsets) at which the task group utilizing the partition can request resource. These request offsets will be used to compute the supply regularity and will be defined in Definition 4.4. For example, considering the slice schedules of the physical resources accessed earlier, the three request offsets of the resource partitions $P_{1,1}, P_{4,1}, P_{3,1}$ in Figure 7 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 request offset extension above is compatible with the RRP model in uniform environment where the request offsets include all the integer time points. In the rest of the paper, we still assume that tasks can only have integer execution time but can request resource in the middle of a resource slice such that the requesting time is non-integer. With this extension, we further extend the definition of the supply function as follows.

Definition 4.3. For any non-negative 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 4.3, 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 a non-integer requesting time and there is a resource slice with a starting time of $\lfloor a \rfloor$. Otherwise, it is $S(b) - S(a)$.*

Based on the above extensions, the effective supply regularity can be defined as follows.

Definition 4.4. Given a resource partition $P = (\mathcal{S}, O, p)$, where $\mathcal{S} = \{s_1, s_2, \ldots, s_n : 0 \le s_1 < s_2 < \cdots < s_n < p\}$ and $O = (\mathcal{O} = \{o_1, o_2, \ldots, o_{n'}, : 0 \le o_1 < o_2 < \cdots < o_{n'} < \overline{p}\}, \overline{p})$. Let e, x_1, x_2 be non-negative integers. The effective supply regularity $\overline{R}$ of partition P is defined as the smallest integer k such that for any slice starting time $s = s_i + x_1 \cdot p$, requesting time $o = o_j + x_2 \cdot \overline{p}$ and e where $0 < i < n, 0 < j < n', e, x_1, x_2 \in \mathbb{N}$

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

Note that tasks in the non-uniform environment have their execution times and deadlines all in integers as they are in the uniform environment. Hence, the resource supply deviation for a task requesting resource at o should be computed for the time intervals $[o, o + e)$ for all integer e. The effective supply regularity defines the maximum supply deviation from the ideal resource supply with regards to the application's requesting time. Effective supply regularity of 1 indicates that the resource supply will never undersupply or oversupply more than one unit of resource considering the application's requesting time. This definition of effective supply regularity is backward compatible with the original definition of supply regularity in Definition 3.6 where the requesting times include every integer time point in uniform environment.

Definition 4.5. 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 a non-integer requesting time $o = o_j + x_2 \cdot \overline{p}$ and an integer slice starting time $s = s_i + x_1 \cdot p$ such that $s = \lfloor o \rfloor$ where $0 < i < n, 0 < j < n', e, x_1, x_2 \in \mathbb{N}$.*

Proof. If such starting time s and requesting time o exist for a regular partition P, by Definition 4.4 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 Definition 3.4, P has $|\mathcal{S}|$ resource slices over the time interval of p; and from Definition 4.2 and 4.3, 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. □

We now define composite resource partition and its regularity.

Definition 4.6. A composite resource partition C is defined as a tuple $(\mathcal{P}, \Pi^c, E)$ where $\mathcal{P}$ is a set of resource partitions, Π^c is a set of physical resource and E is a binary relation on Π^c such that the partition $P_j \in \mathcal{P}$ is on physical resource $\Pi_j \in \Pi^c$ and E represents the resource access order of C.

The composite resource partition can be considered as a collection of resource partitions on a set of physical resources with a fixed resource access order. For example in Figure 6, let Π_1, Π_4 and Π_3 denote 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 may 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 as illustrated in Figure 7.

Definition 4.7. 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: PROBLEM FORMULATION AND ALGORITHM DESIGN

In this section, we study how to construct composite resource partitions in multi-resource environment. We first formulate the ARCRP-S problem, and then present the necessary and sufficient condition for constructing composite resource partition. Building upon this condition, we then propose the ARCRP-S algorithm.

5.1 Problem Formulation and the Necessary and Sufficient Condition for Partition Construction

Problem 5.1. **Acyclic Regular Composite Resource Partition Scheduling (ARCRP-S) Problem:** *Given the resource demands $\{\alpha_{j,i} \mid \forall i\}$ and the resource access order E_i from each application A_i where $\alpha_{j,i}$ represents the requested resource supply rate on physical resource Π_j from A_i, the ARCRP-S problem is to construct a regular composite resource partition $C_i = (\mathcal{P}_i, \Pi_i^c, E_i)$ for each A_i with the assumption that the total resource access order $E = \{(\Pi_m, \Pi_n) \mid (\Pi_m, \Pi_n) \in E_i, \forall i\}$ does not have cycles.*

The ARCRP-S problem is proved to be NP-hard [28]. However, we can transform the availability factor and request offsets of each partition into simpler forms which makes the problem easier. The availability factor of each partition can be adjusted and transformed into the form of $1/m$ for some integer m by allocating more resource. The request offsets can also be transformed into a periodic pattern with a period of $m = \frac{1}{\alpha}$ which covers the original offsets. These transformations will lead to a necessary and sufficient condition for constructing an effective regular partition, although more resources will be allocated for each partition. In this paper, we make the following assumptions: (1) the availability factor α is limited to be $1/m$ for some integer m; and (2) the period of request offsets $\overline{p}$ is $m = \frac{1}{\alpha}$.

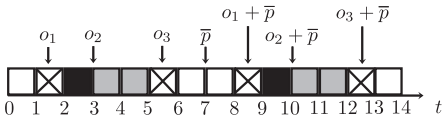


Fig. 8. Examples of resource slice allocation for an effective regular partition P.

Theorem 5.1. *Let a resource partition P has resource slice offsets $\{s_1, s_2, \ldots, s_n\}$, period p and request offsets $O = (\{o_1, o_2, \ldots, o_{n'}\}, \overline{p})$. P is effective regular if and only if $\exists k \in \mathbb{N}, 0 \le k \le n'$*

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

where $o_0 = o_{n'} - \overline{p}$, $o_{n'+1} = o_1 + \overline{p}$ and no resource slice is scheduled on any non-integer requesting time.

Theorem 5.1 gives the necessary and sufficient condition for the slice allocation for an effective regular partition. The resource slices should be scheduled in a periodic time interval enclosed by a pair of neighboring request offsets with a period of $\overline{p}$. The time interval in each period should be scheduled exactly one resource slice. Each pair of neighboring request offsets, o_k and o_{k+1}, is a candidate for such periodic time interval and we denote each choice with a color in Figure 8. For example, resource partition P has request offset $O = (\{o_1 = 1.5, o_2 = 3, o_3 = 5.5\}, \overline{p} = 7)$. o_1, o_2, o_2, o_3 and o_3, o_1 are three candidates and denoted with black, gray and white, respectively. An effective regular resource partition P should have all its resource slices scheduled in the periodic time interval with the same color. For example, $P = (\{0, 6\}, 14)$ is effective regular because all its slices are scheduled in the white periodic interval.

Proof. We first prove the sufficient condition by showing that the effective supply regularity is 1 if we have such schedule. We first construct intervals $[b, a)$ as follows.

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

where b represents a requesting time and a represents some time after b. 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 $\lfloor b \rfloor$ for all non-integer b. So we have

$$y \le S(a) - S(b) \le y + 1 \tag{3}$$

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

$$-\alpha \cdot m \le S(a) - S(b) - \alpha(a-b) \le 1 - \alpha \cdot m \tag{4}$$

From Definition 3.5,

$$-\alpha \cdot m \le I(a) - I(b) = S(a) - S(b) - \alpha(a-b) \le 1 - \alpha \cdot m \tag{5}$$

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

$$-1 < I(a) - I(b) < 1 \tag{6}$$

Equation (6) holds for all time intervals $[b, a)$ where b is the requesting time and $a = b + e$ for $e \in \mathbb{N}$. P is effective regular by Definition 4.4. Next, we prove the necessary condition by contradiction.

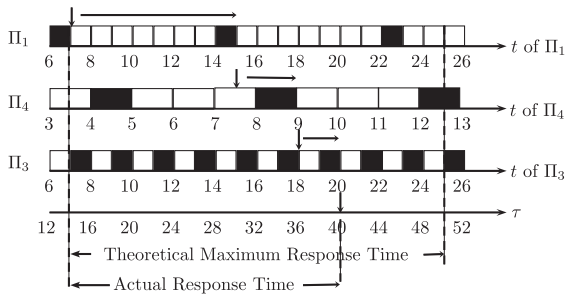


Fig. 9. The result of running Algorithm 1 for application A_1 depicted in Figure 6. The slices scheduled for A_1 are colored in black.

Assume such k does not exist then one of the following cases must be true

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

where s_d and s_{d+1} denote the starting time of two neighboring slices; $d, g, x \in \mathbb{N}$. For **Case 1**, a resource slice is scheduled on a non-integer requesting time and hence P is not effective regular by Lemma 4.2. For **Case 2** and **Case 3**, resource slices are not scheduled in each time interval enclosed by the same pair of neighboring request offsets.

Case 2: Let a requesting time $b = o_g + x \cdot \overline{p}$ and a time $a = b + \overline{p}$. Because there are two resource slices s_d and s_{d+1} in time interval $[b,\ a)$ and $a - b = \overline{p} = \frac{1}{\alpha}$, we have

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

By Definition 3.5, $|I(a) - I(b)| < \overline{R}$ and thus $\overline{R} > 1$ by Definition 4.4. P is not effective regular.

Case 3: Let a requesting time $b = o_{g+1} + x \cdot \overline{p}$ and a time $a = b + \overline{p}$. Because there is no resource slice scheduled in time interval $[b,\ a)$ and $a - b = \overline{p} = \frac{1}{\alpha}$, we have

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

For the same reason as in Case 2, P is not effective regular. □

5.2 ARCRP-S Algorithm

We first give an overview of the ARCRP-S algorithm (Algorithm 1) and then describe each step in detail. In the following, we use the applications depicted in Figure 6 to illustrate each step in the ARCRP-S algorithm. The final schedules for application A_1 is illustrated in Figure 9. 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, respectively. A_1 and A_2 will be assigned a regular composite resource partition $C_1 = (\{P_{1,1}, P_{4,1}, P_{3,1}\}, E_1)$ and $C_2 = (\{P_{2,2}, P_{4,2}, P_{3,2}\}, E_2)$, respectively, and $P_{j,i}$ denotes A_i's partition on Π_j. The desired availability factors are $\alpha_{1,1} = \alpha_{2,2} = \frac{1}{8}, \alpha_{4,1} = \alpha_{4,2} = \frac{1}{4}, \alpha_{3,1} = \alpha_{3,2} = \frac{1}{2}$, respectively.

ARCRP-S determines a linear order of all the physical resources and then constructs the resource partition on each physical resource following this order (Line 1–3). Using the applications as an example, the linear order $\{\Pi_1; \Pi_2; \Pi_4; \Pi_3\}$ is constructed by a topology sort on the total resource

ALGORITHM 1: Overview of ARCRP-S Algorithm

Input: The requested $\alpha_{j,i}$ of all application A_i and the total access order $E = \{(\Pi_m, \Pi_n) \mid (\Pi_m, \Pi_n) \in E_i, \forall i\}$
Output: The partition schedules $\{S_{j,i} \mid \forall j, i\}$.

1 Enqueue all physical resource Π_j into a queue Q following the topological sorted order of E
2 **while** $Q \neq \emptyset$ **do**
3 Dequeue Π_j from Q
4 **for** *each* $P_{j,i}$ *on* Π_j **do**
5 $O_{j,i}$ = **RequestOffset**($\alpha_{j,i}$, $\{P_{m,i} \mid (\Pi_m, \Pi_j) \in E_i\}$)
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** $\{S_{j,i} \mid \forall j, i\}$

ALGORITHM 2: Computation of the Request Offsets

Input: The requested α, a set of resource partitions $\mathcal{P}$ and all the resource slice sizes Q_j and O.
Output: The request offset $O = (O, \overline{p})$ of the partition P.

1 **Procedure RequestOffset**(α, $\mathcal{P}$)
2 $O = \{\}$
3 $\overline{p} = \frac{1}{\alpha}$
4 **for** $P_j \in \mathcal{P}$ **do**
5 $p'_j = p_j \cdot Q_j / Q$
6 H = **LCM**(p'_j, $\overline{p}$)
7 **for** $s \in \mathcal{S}_j$ **do**
8 $t' = (s+1) \cdot Q_j / Q$
9 **for** $x \leftarrow 0$ *to* $H/p'_j - 1$ **do**
10 **Insert**(O, $t' + x \cdot p'_j \ (mod\ \overline{p})$)
11 **end**
12 **end**
13 **end**
14 **return** $O = (O, \overline{p})$

access order E and the construction of schedule will follow this order. To compute the schedules on each physical resource, the algorithm will compute the request offsets in the **RequestOffset** subroutine (Algorithm 2) and construct the partitions in the **ConstructSchedule** subroutine (Algorithm 3). Using the applications as an example, to compute the schedules of $P_{4,1}$ and $P_{4,2}$ on Π_4, the algorithm first computes the request offsets $O_{4,1}, O_{4,2}$ using subroutine **RequestOffset** based on the schedules of $P_{1,1}$ and $P_{2,2}$. In the second step, the schedules of $P_{4,1}$ and $P_{4,2}$ are computed based on $O_{4,1}$ and $O_{4,2}$ using subroutine **ConstructSchedule**. The same procedure will be repeated for each physical resource.

As described above, ARCRP-S has the following two key steps: (1) the computation of the request offsets and (2) the computation of the schedules given the request offsets of each partition. They will be elaborated below.

5.2.1 Computation of the Request Offsets. Algorithm 2 shows the procedure to compute the request offsets of each resource partition P on a physical resource Π. Assuming the resource slice size Q_j of resource Π_j is given, Algorithm 2 computes the request offsets based on the requested α and a set of resource partitions $\mathcal{P}$, that may be accessed right before accessing P according to the access order of application A. For each such partition, the algorithm converts the end time $s+1$ of each resource slice to the time system of P as t' and add such requesting time to the request offsets O as the loops in Line 4 and 7. The requesting time is assumed to be the end time of a resource slice because tasks are assumed to have integer execution time and hence it is also the time the next partition may request for resource. Using the applications as an example, the request offsets can simply include all integer domain as $O_{1,1} = O_{2,2} = (\{0, 1, 2, 3, 4, 5, 6, 7\}, 8)$ if there is no partition accessed right before accessing $P_{1,2}$ and $P_{2,2}$. Moreover, assuming the schedule of $P_{1,1} = (\{6\}, O_{1,1}, 8)$ is computed in the previous phase, to compute the $O_{4,1}$, we need the schedule of $P_{1,1}$ because $P_{1,1}$ is the partition accessed right before $P_{4,1}$. Following the steps in Algorithm 2, $O_{4,1}$ is computed as $(\{3.5\}, 4)$ which can be seen in Figure 9.

5.2.2 Computation of the Partition Schedules. Given the request offset of each partition, the problem to compute the schedule is NP-hard. This can be proved by reducing PMP_1 [43] to this problem and setting the request offsets to be the set of all integer time. We present two algorithms below. ARCRP-S (Algorithm 3) computes the schedules by exploring a large search space with exponential time complexity, while ARCRP-S-Fast (Algorithm 4) computes the schedule by only exploring an essential search space with polynomial-time complexity but it is only applicable to some settings.

ALGORITHM 3: ConstructSchedule for ARCRP-S

Input: The availability factor $\alpha_{j,i}$ and request offset $O_{j,i}$ of each partition on physical resource Π_j.
Output: The schedules $(\{\mathcal{S}_{j,i} \mid \forall i\}$. Otherwise, reject.

1 **Procedure ConstructSchedule**$(\{(\alpha_{j,i}, O_{j,i}) \mid \forall i\})$
2 **for** *Each combination of pairs of request offsets* **do**
3 Initialize $m[t] = 0, \forall t \in [0, H)$ and $\mathcal{S}_{j,i} = \{\}, \forall i$
4 **for** *Each chosen request offset pair* (o'_i, o''_i) **do**
5 $r_{j,i} = \lceil o'_i \rceil; d_{j,i} = \lfloor o''_i \rfloor$
6 **if** $r_{j,i} \geq d_{j,i}$ **then**
7 $d_{j,i} += \overline{p}_{j,i}$
8 **end**
9 **end**
10 Enqueue all partitions $P_{j,i}$ into a new queue Q following the ascending order of deadline $d_{j,i}$
11 **while** $Q \neq \emptyset$ **do**
12 Dequeue $P_{j,i}$ from Q
13 **if** $(\textbf{Next}(\mathcal{S}_{j,i}, r_{j,i}) < d_{j,i})$ *or* $(d_{j,i} > H$ *and* $\textbf{Next}(\mathcal{S}_{j,i}, 0) < (d_{j,i} \mod H))$ **then**
14 **continue**
15 **end**
16 $t = \textbf{EDF}(r_{j,i}, d_{j,i}, m)$
17 **if** $t = NULL$ **then**
18 **break**
19 **end**
20 $\textbf{Insert}(\mathcal{S}_{j,i}, t)$
21 $r_{j,i} += \overline{p_{j,i}}; d_{j,i} += \overline{p_{j,i}}$
22 Enqueue $P_{j,i}$ into Q with deadline $d_{j,i}$
23 **end**
24 **if** $Q = \emptyset$ **then**
25 **return** $\{\mathcal{S}_{j,i} \mid \forall i\}$
26 **end**
27 **end**
28 **return** NULL

ALGORITHM 4: ConstructSchedule for ARCRP-S-Fast

Input: The availability factor $\alpha_{j,i}$, request offset $O_{j,i}$ of each partition on physical resource Π_j and H is the hyperperiod of all the request offsets.

Output: The schedules $\{\mathcal{S}_{j,i} \mid \forall i\}$. Otherwise, reject.

1 **Procedure ConstructSchedule**$(\{(\alpha_{j,i}, O_{j,i}) \mid \forall i\})$
2 Initialize $m[t] = 0, \forall t \in [0, H)$
3 Enqueue all partitions $P_{j,i}$ into a queue Q following the ascending order of period $\overline{p_{j,i}}$
4 **while** $Q \neq \emptyset$ **do**
5 Dequeue $P_{j,i}$ from Q
6 **for** *Each pair of request offset* (o'_i, o''_i) *of* $P_{j,i}$ **do**
7 $r_{j,i} = \lceil o'_i \rceil$
8 $d_{j,i} = \lfloor o''_i \rfloor$
9 $t = \textbf{EDF}(r_{j,i}, d_{j,i}, m)$
10 **if** $t \neq NULL$ **then**
11 **break**
12 **end**
13 **end**
14 **if** $t = NULL$ **then**
15 **return** NULL
16 **end**
17 **for** $x \leftarrow 0$ *to* $H/\overline{p_{j,i}} - 1$ **do**
18 $m[t + x \cdot \overline{p_{j,i}} \mod H] = 1$
19 **end**
20 **end**
21 **return** $\{\mathcal{S}_{j,i} \mid \forall i\}$

Algorithm 3 shows the **ConstructSchedule** subroutine of the ARCRP-S algorithm. The algorithm tries to construct cyclic schedules for every partition by testing each combination of pairs of neighboring request offsets of each partition (see Line 2). Each partition has a release time $r_{j,i}$ and a deadline $e_{j,i}$ based on the chosen request offsets in Line 4–9. The algorithm ends when it finds a cyclic schedule for each partition in Line 13–15 and Line 24–26. The algorithm uses an EDF algorithm to compute the slice schedule in each periodic interval in Line 11, 12, and 16 where the release time and deadline are updated in Line 21. The EDF subroutine takes a release time, a deadline and marks the slot $m[t]$ as occupied in Line 16. H is the hyperperiod of all the periods of request offsets. The **Next** subroutine in Line 13 returns the first element in $S_{j,i}$ no less than $r_{j,i}$. The **Insert** subroutine in Line 20 inserts the slice offset t to $S_{j,i}$.

For example, to compute the schedule of $P_{4,1}$ and $P_{4,2}$, we need the request offset $O_{4,1}$ and $O_{4,2}$. Assuming $O_{4,1} = (\{3.5\}, 4)$ and $O_{4,2} = (\{2.5\}, 4)$, there is only one possible combination of request offset pairs which is $(3.5, 3.5 + 4)$ for $P_{4,1}$ and $(2.5, 2.5 + 4)$ for $P_{4,2}$. $P_{4,2}$ is then picked from the queue to be scheduled because of its smallest deadline. **EDF** subroutine will assign resource slice at 3 to $P_{4,2}$ because $2.5 \leq 3 \leq 6.5$. The release time and deadline will be updated as $(6.5, 10.5)$ and $P_{4,2}$ is put back to the queue. $P_{4,1}$ is the next one to be scheduled based on its deadline. The **EDF** subroutine will assign resource slice at 0 $(4 \mod 4)$ to $P_{4,1}$ because $3.5 \leq 4 \leq 7.5$ and $H = 4$ which can be seen in Figure 9. The release time and deadline will be updated as $(7.5, 11.5)$ and $P_{4,1}$ is put back to the queue. In the next steps, $P_{4,2}$ and $P_{4,1}$ will be dequeued and the algorithm will halt because both of them have cyclic schedules as checked in Line 13–15.

The ARCRP-S algorithm can work for partitions with the availability factors and the periods of the request offsets transformed into the form of $\frac{1}{m}$ and m, respectively. However, it has exponential time complexity. Assuming that the number of physical resources, the size of total resource access

order, the periods of the slice offsets, the periods of the request offsets and the hyper-period of all the periods are bounded by a constant C, the time complexity of the ARCRP-S algorithm is $O(N \cdot C^N)$ if the number of applications is N. To further reduce the time complexity and potentially make it an online algorithm, we can further adjust and transform the availability factors into the form of power of $\frac{1}{2}$ and set the size of resource slice to be power of 2. In the following, we denote such setting as geometric sequence with common ratio of 2 setting (called *GS-2 setting*). In this way, we can employ a similar idea as the AAF algorithm where partitions are linearly scheduled by following the ascending order of periods. Algorithm 4 summarizes the ARCRP-S-Fast algorithm. The algorithm picks the first pair of neighboring request offsets that allows a resource slice to be scheduled by EDF algorithm (Line 6–12). We then mark the resource slice assigned for each interval in the hyperperiod H to avoid partition with larger period to conflict with partition with smaller period (Line 17–19). Using the same example as mentioned above, $O_{4,1} = (\{3.5\}, 4)$ and $O_{4,2} = (\{2.5\}, 4)$. $P_{4,2}$ will be first assigned resource slice at 3 for having request offset pair of $(2.5, 6.5)$. $P_{4,1}$ will then be assigned resource slice at 0 for having request offset pair of $(3.5, 7.5)$. With the above mentioned assumptions, the time complexity of the ARCRP-S-Fast algorithm is $O(C^4 \cdot N + C \cdot N \cdot \log N)$ if the number of applications is N.

6 DYNAMIC RECONFIGURATION OF RESOURCE PARTITION IN NON-UNIFORM ENVIRONMENT

In open system environments under the RRP model, applications may request to reconfigure their resource partitions on demand. This section first revisits the dynamic reconfiguration of resource partitions in uniform environments and then extend the study to non-uniform environments.

6.1 Reconfigurable RRP Model in Uniform Environments

In uniform environments, an application can issue a *Reconfiguration Request of Resource Partition* (R^3P) to request new resource partitions or reconfigure the existing ones. The application can request to reconfigure its resource supply curve by issuing an R^3P (see Figure 10). 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 Figure 10(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 Figure 10(a) (lower dash supply curve) and (b) (upper dash supply curve).

Recall that a resource partition P in uniform environments is a tuple $(\mathcal{S}, p)$ which describes its cyclic schedule and period. In each stage, the cyclic schedule of the resource partition can be described using this tuple counting from time zero at the start of the stage. The resource partition P at different stages can be described using different superscripts. The resource partition before, during and after the reconfiguration is denoted as P^o, P^t and P^n, respectively. Based on these notations, the reconfiguration request of resource partition (R^3P) can be formally defined as follows.

Definition 6.1. Reconfiguration Request of Resource Partition (R^3P) is defined as a tuple $\lambda = \{\mathcal{P}, \mathcal{F}, \mathcal{R}, T\}$ where $\mathcal{P}$ is the set of resource partitions;[1] each resource partition $P_i \in \mathcal{P}$ will have associated availability factor of $\alpha_i \in \mathcal{F}$ and P_i will have reconfiguration supply regularity (see Definition 6.3) of $R_i^r \in \mathcal{R}$. T is the maximum time allowed for the reconfiguration to complete.

[1] $\mathcal{P}$ denotes the set of all the partitions. However, these partitions are neither specified with the resource slice offsets nor the performance semantics. Their desired availability factors and reconfiguration supply are defined in $\mathcal{F}$ and $\mathcal{R}$, respectively.

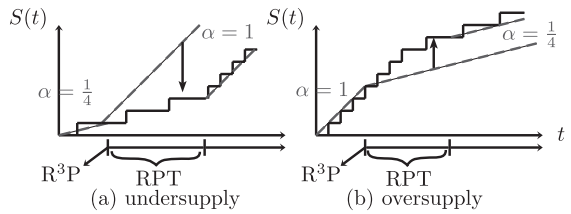


Fig. 10. The dash line shows that the requested availability factor changes from $\frac{1}{4}$ and 1 to 1 and $\frac{1}{4}$ in (a), (b), at the time of R^{3}P. The arrow shows the supply deviation during the reconfiguration. There is resource undersupply in (a) but resource oversupply in (b).

The resource supply of a resource partition P after the R^3P is guaranteed by enforcing the availability factor and regularity of each partition while the performance semantics of the reconfiguration can be defined as the maximum deviation between the actual resource supply and the desired supply 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, α^o and α^n are used to denote the availability factor of P before and after the reconfiguration, respectively. The definition of instant regularity is thus extended as follows.

Definition 6.2. The instant regularity $I(t)$ of a resource partition P at time $t \geq q$ is $I(t) = S(t) - (\alpha^o \cdot q + \alpha^n(t-q))$ where q is the time of an R^3P;

Based on the above extension, the *reconfiguration supply regularity* for uniform environments is defined as follows.

Definition 6.3. Let a, b, k be non-negative integers. The reconfiguration supply regularity of resource partition P is defined as R^r which equals to the smallest $k \geq 1$ such that $I(b) - I(a) > -k, \forall b \geq a$.

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 on the definition gives the scheduler flexibility to schedule the resource slices to earlier time compared to the case where the maximum oversupply is also bounded. For example, without this relaxation, a regular partition with an availability factor of $\frac{1}{2}$ has a periodic schedule which cannot be edited on the fly. With this relaxation, the partition schedule can be changed on the fly by shifting the schedule one time slot earlier. This relaxation, however, is only applicable during the partition reconfiguration.

6.2 Extending Partition Reconfiguration to Multi-resource Environments

We now extend dynamic reconfiguration of resource partitions to multi-resource environments. We first extend the definitions of reconfiguration request and reconfiguration supply regularity as follows.

Definition 6.4. Reconfiguration Request of Regular Composite Resource Partition (R^{3}CRP) is defined as $\lambda = \{C, \mathcal{F}, \mathcal{R}, T\}$ where C is the set of all regular composite resource partitions; each effective regular resource partition $P_{j,i}$ has the requested $\alpha_{j,i} \in \mathcal{F}$ for the composite resource partition C_i with an associated effective reconfiguration supply regularity (see Definition 6.5) of $\overline{R^r_{j,i}} \in \mathcal{R}$; T is the maximum time allowed for the reconfiguration to complete.

Definition 6.5. Let e, x_1, x_2 be non-negative integers, $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$ be the slice offsets of P, p be the period of P and $O = (\{o_1, o_2, \ldots, o_{n'}\}, \overline{p})$ be the request offsets of P. The effective reconfiguration supply regularity $\overline{R^r}$ of partition P is defined as the smallest integer k such that for any slice starting time $s = s_i + x_1 \cdot p$, requesting time $o = o_j + x_2 \cdot \overline{p}$ and e, where $0 < i \leq n$, $0 < j \leq n'$, $e, x_1, x_2 \in \mathbb{N}$,

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

Based on Definition 6.5, we can define the effective reconfiguration regular partition as follows.

Definition 6.6. A 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.

The problem of reconfiguring composite resource partitions can be reduced to the problem of composite resource partition construction described in Section 5. The construction problem is proved to be NP-hard [28]. In order to develop an online algorithm to reconfigure composite resource partitions, we have the following assumptions:

AS-1: time zero of each physical resource is synchronized;

AS-2: no concurrent reconfiguration request is allowed in the system;

AS-3: the availability factor of P and the resource slice size of each resource are adjusted and set to be the power of $\frac{1}{2}$ and 2, respectively. Also, the sum of the availability factors on each physical resource is no larger than 1;

AS-4: the composite resource partitions are acyclic and regular.

AS-1 assumes time synchronization among the physical resources. This allows us to study the resource misalignment problem only caused by the non-uniform sizes of resource slices but not the time drifts among the physical resources. AS-2 assumes that only one reconfiguration request can happen at any time in the system to simplify the performance semantics during the reconfiguration. By limiting the form of availability factor and slice size, AS-3 allows us to adopt the ARCRP-S-Fast algorithm to achieve a good balance between schedulability and polynomial-time complexity. AS-4 is a consistent assumption as made for composite resource partition construction (see Section 5) to construct/reconfigure regular composite partitions with acyclic resource access order. We now define the ARCRP-DR problem as follows.

Problem 6.1. **Acyclic Regular Composite Resource Partition Dynamic Reconfiguration (ARCRP-DR) Problem:** *Given a composite resource partition reconfiguration request $\lambda = \{C, \mathcal{F}, \mathcal{R}, T\}$ and the state of all the resource partitions before the request ($\{P^o_{j,i}\}$), the problem is to compute the partition schedules during the reconfiguration $P^t_{j,i}$ and after the reconfiguration $P^n_{j,i}$ on each physical resource π_j 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 $\alpha_{j,i} \in \mathcal{F}$;*

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

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

Please note the composite resource partition may be irregular during reconfiguration as some resource partitions may be effective irregular but each resource partition will be effective regular after the reconfiguration is completed.

6.3 ARCRP-DR Algorithm

The ARCRP-DR problem can be solved by breaking the reconfiguration request down to a set of independent R^3Ps and each R^3P can be handled by the DPR algorithm independently on each physical resource as if reconfiguring only a single resource [12]. However, the effective supply regularity and effective reconfiguration supply regularity will increase if there is the resource misalignment problem. To further improve the schedulability and mitigate the problem, we present the ARCRP-DR algorithm by first introducing the important concepts and presenting a running example of a simplified algorithm, and then describing the ARCRP-DR algorithm details.

6.3.1 Dynamic RRP Scheduler. In the following, we revisit the concept of Dynamic RRP Scheduler by introducing the *maximum supply shortfall*, give an example of a *dynamic RRP scheduler* [44] in uniform environments, and then extend the concept to non-uniform environments.

Maximum Supply Shortfall: The concept of maximum supply shortfall was introduced in [44]. We extend this concept below by taking the request offsets into consideration.

Definition 6.7. Let $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$ be the slice offsets, p be the period, $O = (\{o_1, o_2, \ldots, o_{n'}\}, \overline{p})$ be the request offsets of a resource partition P and t be the time when there exists at least one request offset o_j s.t. $t \geq o_j$. Given that no resource slice is scheduled on any requesting time, the maximum supply shortfall of P at time t is defined as $d(t)$ such that for any requesting time $o = o_j + x_2 \cdot \overline{p}$, where $0 < j \leq n'$ and $x_2 \in \mathbb{N}$

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

$d(t)$ denotes the maximum supply shortfall among all the time intervals ending at time t and it takes request offsets into consideration. Based on Definition 6.5, the effective reconfiguration regularity will be the smallest positive integer k such that $d(t) > -k \ \forall t$.

Dynamic RRP Scheduler: Traditional RRP schedulers are mostly static schedulers [3, 5]. The concept of maximum supply shortfall 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. Given the **C-1** and **C-2** requirements in Problem 6.1 and the maximum supply shortfall of a partition at time t, we can compute the deadline to schedule the next slice based on Definition 6.7 and Definition 4.4 in order to preserve the regularity of the partition [12].

In the following, we give a simplified example of the dynamic partition reconfiguration algorithm scheduling a single partition in Figure 11. 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. To simplify the model, we omit the computation of the maximum supply shortfall, assume that the relative deadline computed is always derived as 5 and the desired regular partition shall have period of 4 after the reconfiguration. As illustrated in Figure 11, time 0 is the time of the reconfiguration request and time 0-4 is the RPT stage. Starting from time 4, which is the end of the reconfiguration, the partition shall have a cyclic schedule with period of 4. 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

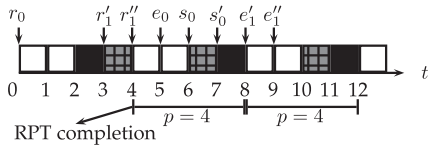


Fig. 11. An example of a dynamic RRP-based partition scheduler where resource slices in black and gray color are two possible schedules. The release time and deadline of the next instance will be based on the previous instance.

a cyclic schedule with a period of 4. The partition with either resource slice offset $s_0 = 6$ or $s'_0 = 7$ with a period of 4 has a valid cyclic schedule.

Fast Deadline Computation in Non-Uniform Environment: In the example, we use a simplified example to illustrate the algorithm by omitting the computation of the deadline. Naïvely computing the resource supply shortfall and deadline of each partition for all time t will impose significant complexity. In the following, we present the Theorem to update the deadline of each partition P_i in constant time for dynamic reconfiguration by only tracking (1) the maximum supply shortfall $d_i(t)$ at each scheduling decision time t and (2) the nearest future requesting time $o^t > t$.

Theorem 6.1. *A resource partition P has effective reconfiguration regularity of $\overline{R^r} \leq k$ if no resource slice is scheduled on any requesting time and for any time t,*

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

where s^t and o^t are the smallest resource slice starting time and requesting time which are no less than t, respectively.

Proof. We prove this theorem by showing that for $\forall t, d(t) > -k$ by Mathematical Induction. This leads to $I(o + e) - I(o) > -k$ for any requesting time o and $e \in \mathbb{N}$ by Definition 6.7. P is effective reconfiguration regular by Definition 6.5.

For $t < o^0$, both the regularity and $d(t)$ are undefined. 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 Definition 6.2, Definition 6.7 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 requesting 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 $I(o^x) < \max_{\forall o \leq o^x}(I(o))$, by Definition 6.7 and the fact that there is no resource slice scheduled between $[s^t + 1, s^x)$, we have

$$t_1 + d(t_1)/\alpha^n = X \tag{7}$$

and

$$d(t_1) \geq d(s^x) \text{ for } t_1 \in [s^t + 1, s^x] \tag{8}$$

where X is a fixed value. By Definition 6.7, Definition 6.2 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 $s^x \leq s^t + 1 + d(s^t + 1)/\alpha^n + k/\alpha^n - 1$, Equation (7) and $d(s^t + 1) > -k$, we have $d(s^x) >= -k + \alpha^n$. Further by Equation (8), we have $d(t_1) > -k$ for $t_1 \in [s^t + 1, s^x + 1]$.

Case 2: If $I(o^x) = \max_{\forall o \le o^x}(I(o))$, 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 Definition 6.2 and the fact that there is no resource slice scheduled between $[o^x, s^x)$, we have

$$d(t_1) \ge d(s^x) \text{ for } t_1 \in (o^x, s^x] \tag{9}$$

From the theorem, we have $s^x \le o^x + k/\alpha^n - 1$ and thus $d(s^x) = -k + \alpha^n$. By Equation (9), we have $d(t_1) > -k$ for t_1 in $[o^x, s^x + 1]$.

Combining both cases and Equation (7), we have $d(t'') > -k$ for any time $t'' \le s^t + 1$ for any $t < t''$. Please 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 Definition 6.7 and Definition 6.2.

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

Intuitively, we can consider the term k/α^n to be the budget that can be used to delay the next resource slice and the term $d(t)/\alpha^n$ to be the already consumed budget. This theorem states a sufficient deadline for each resource slice such that the resource partition P is guaranteed to have effective reconfiguration supply regularity less than or equal to k. Based on this theorem, we can progressively construct the schedule by (1) scheduling slices by the deadline and not on any requesting time; (2) updating the maximum supply shortfall and deadline; and (3) repeating the above two steps until a cyclic schedule for each partition is found for the RPT stages.

6.3.2 ARCRP-DR Algorithm. In the following, we use $\mathcal{T}_t$ to denote the state of the partition system at time t, which includes the time r_i of the last scheduling decision, the maximum supply shortfall d_i at r_i, and the deadline e_i to schedule the next slice for each partition P_i. We first give an overview of the ARCRP-DR algorithm and then present the detailed steps to compute the schedules.

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

Initialization. Algorithm 6 aims to compute the maximum supply shortfall $d_i(q)$ and the deadline of each partition based on Theorem 6.1. It first computes the request offset of each partition as $O_{j,i}$ (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 on Algorithm 2. $O_{j,i}$ denotes the union of $O^o_{j,i}, O^t_{j,i}, O^n_{j,i}$ while considering the present time of each symbol. The procedure **Next**$(O_{j,i}, q)$ computes the next nearest requesting time $n_{j,i}$ no less than q based on the request offsets of $P_{j,i}$ (Line 5). The **ComputeMSS** procedure computes the maximum supply shortfall based on Definition 6.7. Please note that the procedure can be improved to be computed in constant time [12]. In Line 6–12, we compute the maximum supply shortfall $d_i(q)$ based on two conditions and update the deadline $e_{j,i}$ based on Theorem 6.1 (Line 8 and 11).

Transition Schedule Computation. Algorithm 7 computes the transition schedule $P^t_{j,i}$ for each partition given a partition system computed in Stage-1 by following three heuristic principles: (1) we avoid scheduling resource slices on any requesting time; (2) we employ the ***deferrable***

ALGORITHM 5: Overview of ARCRP-DR

Input: Reconfiguration request λ, request time q and resource access order $E = \{(\Pi_m, \Pi_n) \mid (\Pi_m, \Pi_n) \in E_i, \forall i\}$.
Output: Transition schedule $\mathcal{S}^t_{j,i}$ and cyclic schedule $\mathcal{S}^n_{j,i}$ for all $P_{j,i}$. Reject if no feasible schedule.

1 Enqueue all physical resources Π_j into a queue Q by the topological sorted order of E
2 **for** $b \leftarrow 0$ *to* T **do**
3 $Q_t = Q$
4 **while** $Q_t \neq \emptyset$ **do**
5 Dequeue Π_j from Q_t
6 $\mathcal{T}_q$ = **Initialization**(Π_j, q) // **Stage-1**
7 $(\{\mathcal{S}^t_{j,i} \mid \forall i\}, \mathcal{T}_{q+b})$ = **TransitionSchedule**($\mathcal{T}_q$, q, $q+b$) // **Stage-2**
8 **if** $\mathcal{T}_{q+b}$ = *Null* **then**
9 **break**
10 **end**
11 $\{\mathcal{S}^n_{j,i} \mid \forall i\}$ = **CyclicSchedule**($\mathcal{T}_{q+b}$) // **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

ALGORITHM 6: Partition System Initialization

Input: Physical resource Π_j and the requesting time q.
Output: $\mathcal{T}_q$, the state of the partition system.

1 **Procedure Initialization**(Π_j, q)
2 **for** *Each* $P_{j,i}$ *on* Π_j **do**
3 $r_{j,i} = q$
4 $O_{j,i}$ = **ComputeAllOffsets**($P_{j,i}$)
5 $n_{j,i}$ = **Next**($O_{j,i}$, q)
6 **if** $P_{j,i}$ *is a new* partition **then**
7 $d_{j,i} = \infty$
8 $e_{j,i} = n_{j,i} + \overline{R^r_{j,i}}/\alpha^n_{j,i}$
9 **else**
10 $d_{j,i}$ = **ComputeMSS**($P_{j,i}$, q, $O_{j,i}$)
11 $e_{j,i} = \min(q + d_{j,i}/\alpha^n_{j,i}, n_{j,i}) + \overline{R^r_{j,i}}/\alpha^n_{j,i}$
12 **end**
13 **end**
14 **return** $\mathcal{T}_q$

15 **Procedure ComputeMSS**($P_{j,i}$, q, $O_{j,i}$)
16 $d_{j,i} = \infty$
17 **for** *Each* $o \in O_{j,i}$ *and* $o \leq q$ **do**
18 $d_{j,i} = \min(d_{j,i}, S_{j,i}(q) - S_{j,i}(o) - \alpha^o_{j,i}(q - o))$
19 **end**
20 **return** $d_{j,i}$

scheduling (DS)*-*EDF algorithm [45] 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 through DS-EDF is larger than the time budget b, the algorithm will try to schedule it in an idle slice before 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.

Algorithm 7 selects the partition with the earliest deadline and schedules it as late as possible using the **DS-EDF** procedure (Line 6). If a partition is not able to be scheduled before the deadline, the algorithm will reject (Line 8–10). Any time when a partition is scheduled a slice, the algorithm will update its $d_{j,i}, e_{j,i}, r_{j,i}$ using the **UpdateStates** procedure.

In the **DS-EDF** procedure in Algorithm 8, the resource slice is not only scheduled as late as possible as shown in Line 6 but also prohibited from being scheduled on any requesting time based on O. This will avoid the resource misalignment problem. The **UpdateStates** procedure is based on Theorem 6.1. It first updates $d_{j,i}$ from $d_i(r_{j,i})$ to $d_i(r)$ in Line 14–17. Based on Definition 6.7 and Definition 6.2, if there is no requesting time between $r_{j,i}$ and r, $d_{j,i}(r) = d_{j,i}(r_{j,i}) + 1 - \alpha^n_{j,i}(r - r_{j,i})$ as in Line 14. If there exists a request time between $r_{j,i}$ and r, we only need to consider if $n_{j,i}$ would give a lower supply shortfall as in Line 16. In Line 18–20, we update the states based on Theorem 6.1.

Cyclic Schedule Computation. Algorithm 9 computes the cyclic schedule of each partition. This algorithm combines the idea from the AAF algorithm where partitions are scheduled according to their periods; the idea of supply shortfalls and requesting time. Line 6 in Algorithm 9 finds an available slot for each partition and marks the slots as used for the cyclic schedule in Line 11–13. p_{max} is the largest period among all the partitions.

In the following, we check whether the problem requirements are satisfied.

Problem6.1 C-1: Each partition $P^n_{j,i}$ is effective regular by scheduling resource slices in periodic time intervals based on a pair of neighboring request offsets by Theorem 5.1.

ALGORITHM 7: Transition Schedule Computation

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

Procedure TransitionSchedule($\mathcal{T}$, q, b)
  Initialize $m[t] = 0$, $\forall t \in [0, b)$
  Enqueue all partitions $P_{j,i}$ into a queue Q in the ascending order of (1) deadline $e_{j,i}$ and (2) period $p_{j,i}$ for tie-breaker
  while $Q \neq \emptyset$ **do**
    Dequeue $P_{j,i}$ from Q
    t = **DS-EDF**($r_{j,i} - q$, $e_{j,i} - q$, m, b, $O_{j,i}$)
    if t = *NULL* **then**
      if $e_{j,i} \leq q + b$ **then**
        return NULL // Deadline will miss
      end
      continue // We are done with this partition
    end
    Add $t - q$ to $\mathcal{S}^t_{j,i}$
    UpdateStates($\mathcal{T}$, $q + t + 1$, $P_{j,i}$)
    Enqueue $P_{j,i}$ to Q
  end
  return ($\{\mathcal{S}^t_{j,i} \mid \forall i\}$, $\mathcal{T}$)

ALGORITHM 8: **DS-EDF** and **UpdateStates**

Procedure DS-EDF(r, e, m, b, O)
  $e = \lfloor e \rfloor$
  if $e >= b$ **then**
    $e = b$
  end
  for $t \leftarrow e - 1$ *to* r **do**
    if $m[t] = 0$ *and* t *is not on any requesting time* **then**
      $m[t] = 1$
      return t
    end
  end
  return NULL

Procedure UpdateStates($\mathcal{T}$, r, $P_{j,i}$)
  $d_{j,i} = d_{j,i} + 1 - \alpha^n_{j,i}(r - r_{j,i})$
  if $r \geq n_{j,i}$ **then**
    $d_{j,i} = \min(d_{j,i}, 1 - \alpha^n_{j,i}(r - n_{j,i}))$
  end
  $n_{j,i}$ = **Next**($O_{j,i}$, r)
  $e_{j,i} = \min(r + d_{j,i}/\alpha^n_{j,i}, n_{j,i}) + \overline{R^r_{j,i}}/\alpha^n_{j,i}$
  $r_{j,i} = r$
  return

Problem6.1 C-2: The effective reconfiguration supply regularity of each partition $P_{j,i}$ is guaranteed based on Theorem 6.1 by scheduling source slices by the deadline and the fact that $P_{j,i}$ has a cyclic schedule with exact one slice offset. Note that, this algorithm can only be used to compute the schedules when the availability factors and the resource slice sizes are the power of $\frac{1}{2}$ and 2, respectively.

Problem6.1 C-3: The reconfiguration is done by the end of the RPT stage which is no longer than T as in Algorithm 7.

Assuming that the number of physical resources, the size of total resource access order, the periods of the slice offsets, the periods of the request offsets, the hyper-period of all the periods and the budget are bounded by a constant C, the time complexity of ARCRP-DR algorithms is $O(C^5 \cdot N + C^2 \cdot N \cdot \log N)$ if the number of applications is N.

7 PERFORMANCE EVALUATION

In this section, we first compare the application response times in a real system with and without considering the resource misalignment problem. In the second set of experiments, we compare the performance of the ARCRP-S and AAF algorithms [3] in both uniform and non-uniform environments. In the final set of experiments, we evaluate the performance of the ARCRP-DR algorithm in a non-uniform environment.

7.1 Real System Evaluation

To demonstrate the applicability of the concept of regular composite resource partition in practice, we implemented a multi-resource scheduling system 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 [46]. Figure 12 illustrates the system architecture. There is one master CPU resource and multiple slave CPU resources. For the network resource, it consists of a single RT-WiFi **access point (AP)** and a cluster of RT-WiFi stations operating on the same channel.

CPU Resource: We add a layer of resource-level scheduler on top of the original scheduler to schedule each partition. A table driven scheduler in the task-level scheduler hierarchy is added to provide the capability of recursive resource partitioning [28]. The new hierarchy is illustrated in the master CPU resource diagram in Figure 12.

ALGORITHM 9: Cyclic Schedule Computation

Input: $\mathcal{T}_b$, the state of the partition system after the RPT.
Output: Cyclic schedule $\{\mathcal{S}^n_{j,i} \mid \forall i\}$. Otherwise, reject.

Procedure CyclicSchedule($\mathcal{T}_b$)
  Enqueue all $P_{j,i}$ into a queue Q in the ascending order of (1) period $p_{j,i}$ and (2) deadline $e_{j,i}$ for tie-breaker
  Initialize $m[t] = 0, \forall t \in [0, p_{max})$
  while $Q \neq \emptyset$ **do**
    Dequeue $P_{j,i}$
    t = **DS-EDF**$(0, e_{j,i} - q - b, m, p_{j,i}, O_{j,i})$
    if $t = NULL$ **then**
      return NULL
    end
    Add t to $\mathcal{S}^n_{j,i}$
    for $x \leftarrow 0$ *to* $p_{max}/p_{j,i} - 1$ **do**
      $m[l + x \cdot p_{j,i}] = 1$
    end
  end
  return $\{\mathcal{S}^n_{j,i} \mid \forall i\}$

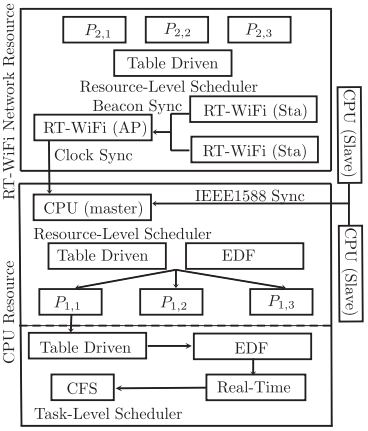


Fig. 12. Overview of the multi-resource scheduling framework.

Network Resource: We use RT-WiFi [46] to schedule the network resource. RT-WiFi is a real-time high-speed wireless data link layer protocol that can provide deterministic packet delivery with adjustable sampling rates up to $6kHz$.

Synchronization: To schedule regular composite resource partitions, we need to synchronize the clocks and schedules on different physical resources (see Figure 12). In our system, every slave CPU synchronizes its clock with the master CPU's clock using IEEE 1588 software implementation [47]. For the network resources, the RT-WiFi AP synchronizes its clock with the master CPU and each RT-WiFi station synchronizes its clock with the AP via RT-WiFi beacon frame.

The testbed 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 an RT-WiFi AP. 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. A_1 has resource accessing order as CPU1, RT-WiFi and CPU2 resources. The task T periodically processes the sensor data on CPU1, passes the data to CPU2 and finally processes the data and passes the decision to the actuator on machine 2. 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.

Figure 13 shows the cumulative distribution function of the response time for task T in each setting. The line denoted as Misaligned shows the results with AAF and the resources are not aligned. The resource slice sizes of CPU1 and CPU2 are set to $1ms$ and the resource slice size of the RT-WiFi is set to $1024\mu s$. On the other hand, the line denoted as Synchronized shows the results with ARCRP-S and resource slices are all synchronized with the size of $1ms$. The constructed regular resource partitions for A_1 are $P_{1,1}, P_{2,1}, P_{3,1}$ on CPU1, RT-WiFi and CPU2 with availability factors of $0.25, 0.25$ and 1, respectively. The theoretical maximum response time of each task instance would be $1ms$ on CPU1 as it starts on CPU1, $1ms/0.25$ (or $1024\mu s/0.25$ for the Misaligned run) on RT-WiFi for the queuing and processing time; and $1ms/1$ on CPU2 with a total of $6ms$. In Figure 13, the Misaligned case shows that 30% task instances have response times larger than $6ms$ while the Synchronized case shows only 5%. This shows that the resource misalignment problem may cause serious deadline miss, while synchronizing the resource slice size and the time can mitigate it.

We next demonstrate a case how ARCRP-S Algorithm can be applied to real-world systems in a non-uniform environment. With the same testbed, the resource slice sizes of CPU1 and CPU2

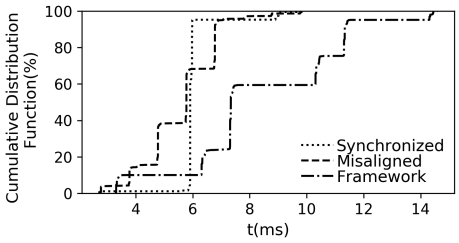


Fig. 13. The cumulative distribution function of the response time for task T under three different settings.

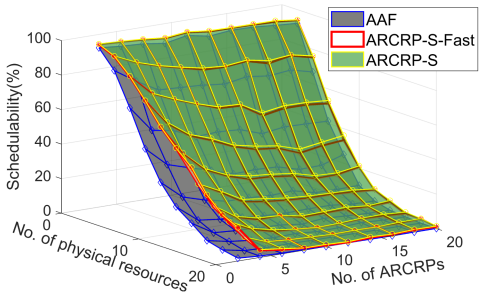


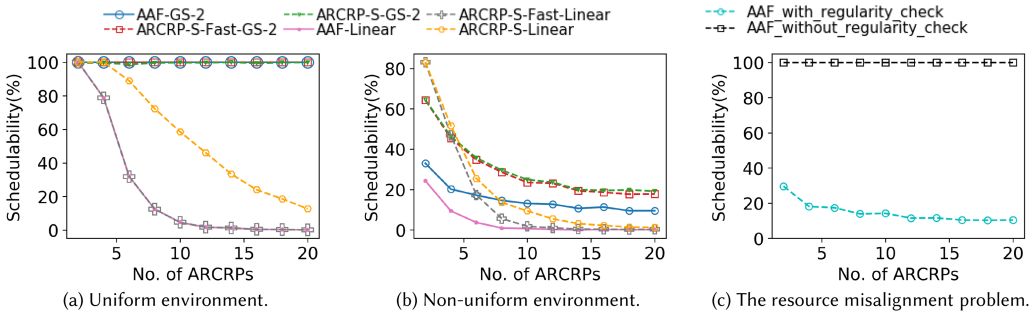
Fig. 14. Schedulability of ARCRP-S, ARCRP-S-Fast and AAF algorithm in non-uniform environment with the GS-2 setting.

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, assigned to A_1, 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 max response time of each task instance would be $1ms$ on CPU1, $(1 \cdot 2ms)/0.25$ on RT-WiFi and $1ms/0.25$ on CPU2 with a total of $13ms$. The results denoted as Framework in Figure 13 show that 95% of the task instances have response times less than 13 ms.

7.2 Performance Comparison Among ARCRP-S, ARCRP-S-Fast and AAF

In the following, we present the simulation results to compare three algorithms, ARCRP-S, ARCRP-S-Fast and AAF [3], under different settings. In uniform environments, the resource slice size of each physical resource is set to be 1. In non-uniform environments, the resource slice size is set to 2^i $(0 \leq i \leq 7)$ in the GS-2 (geometric sequence with common ratio of 2) setting and i $(2 \leq i \leq 7)$ in the linear setting. The requested availability factor of each partition is restricted to $\frac{1}{2^i}$ $(1 \leq i \leq 7)$ in the GS-2 setting and $\frac{1}{i}$ $(2 \leq i \leq 7)$ in the linear setting. Furthermore, all the composite resource partitions requested by the applications are regular. To compute the schedulability, 1,000 samples are generated for each combined setting and the three algorithms are used to construct composite resource partitions. Each sample is marked as schedulable by each algorithm if a schedule can be constructed except for the AAF algorithm. The schedule constructed by the AAF algorithm is further validated by checking the effective supply regularity of each partition because of the resource misalignment problem.

Our simulation results in different settings generally show similar trends that more physical resources and composite resource partitions lead to lower schedulability. This is because the probability that a task requesting resources at some non-integer time point will increase and this leads to resource misalignment problem. For example, Figure 14 shows the schedulability trend of the three algorithms in non-uniform environment under the GS-2 setting. AAF performs the worst because it doesn't consider the resource misalignment problem. ARCRP-S-Fast and ARCRP-S algorithms perform similarly under the GS-2 setting. The ARCRP-S-Fast algorithm employs a similar strategy as the AAF algorithm which schedules partitions following the ascending order of their periods. This strategy searches a small but large enough search space under the GS-2 setting with polynomial time complexity. On the other hand, the ARCRP-S algorithm searches a much larger search space for general settings with exponential time complexity. Thus, ARCRP-S algorithm performs similar to the ARCRP-S-Fast algorithm. To make the figures easy to read, in the following, we



(a) Uniform environment. (b) Non-uniform environment. (c) The resource misalignment problem.

Fig. 15. Performance comparison among ARCRP-S, ARCRP-S-Fast and AAF algorithms.

only present the results where the number of physical resources is 10 and the number of resource partitions varies between 2 and 20.

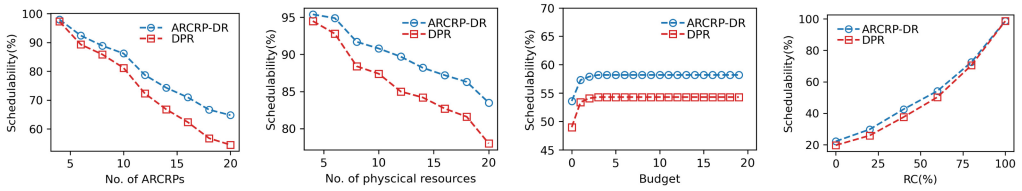
7.2.1 Uniform Environment. Figure 15(a) shows the schedulability results of the three algorithms in uniform environment under GS-2 and linear settings. ARCRP-S-Fast is equivalent to AAF in terms of schedulability under both settings because there is no resource misalignment problem. In fact, AAF is optimal under the GS-2 setting where it can construct a feasible schedule if there exists one [3]. ARCRP-S-Fast is also optimal under the GS-2 setting because it is exactly the same as AAF if there is no resource misalignment problem. On the other hand, ARCRP-S performs slightly worse (1%) than ARCRP-S-Fast under the GS-2 setting but it performs significantly better than the other two algorithms under the linear setting. This is because ARCRP-S explores a very large decision space which results in much better schedulability with settings other than GS-2 setting.

7.2.2 Non-Uniform Environment. Figure 15(b) shows the schedulability of the three algorithms in non-uniform environments. With linear setting, they all perform poorly when the number of composite resource partitions is large. This is because it is almost impossible to avoid scheduling resource slices on any of the requesting times when there is no restriction on the resource slice sizes and availability factors. However, under the GS-2 setting, ARCRP-S-Fast and ARCRP-S perform closely and much better than AAF, which doesn't consider the resource misalignment problem.

Figure 15(c) shows the schedulability comparison of the AAF algorithm in the GS-2 setting with and without the regularity check. Without the regularity check, the AAF algorithm will schedule the partitions as if they are in the uniform environment under the GS-2 setting and the schedulability will be incorrectly considered as 100%. However, when the resource misalignment problem is taken into consideration by checking the regularity, the actual schedulability may drop more than 70% because of the existence of non-integer request offsets.

7.3 Performance Evaluation on the ARCRP-DR Algorithm

We now compare the results of reconfiguring composite resource partitions using ARCRP-DR and DPR in non-uniform environments. The experiment is conducted by generating 1,000 samples for each combined setting and using the two algorithms to construct the schedules. For each sample, if a schedule can be computed by ARCRP-DR, this sample is schedulable. For DPR, a schedule needs to be computed and the regularity of all the partitions need to be checked to have the required regularity. For the parameters, the numbers of partitions and physical resources are both set to 10. The resource slice size is restricted to be 2^i $(0 \leq i \leq 7)$. The availability factors of each partition before and after reconfiguration are randomly sampled in the set of $\frac{1}{2^i}$ $(1 \leq i \leq 7)$ and each physical resource will have the same resource utilization at 80%. The effective reconfiguration supply



(a) Varying RCPRs (b) Varying resources (c) Varying budget (d) Varying RC

Fig. 16. Schedulability comparison: ARCRP-DR vs. DPR-based algorithm.

regularity $\overline{R_i^r}$ of each resource partition is sampled from $[1, 5]$ and 60% of partitions have effective reconfiguration regularity over 1. The transition time budget T is randomly sampled from $[5, 20]$. To focus on evaluating the reconfiguration algorithm, the requested composite resource partitions are assumed to be able to be constructed by ARCRP-S-Fast (Algorithm 4) without considering the reconfiguration regularity.

Figure 16(a) and 16(b) show how the number of partitions and physical resources affects the schedulability. Increasing the number of resource partitions lowers the schedulability as shown in Figure 16(a). This is because there are more resource partitions on each physical resource and this will result in lower schedulability. Note that, we set the resource utilization of each physical resource to be the product of 4.5% and the number of composite resource partitions. On the other hand, increasing the number of physical resources also decreases the schedulability as shown in Figure 16(b). With an increasing number of physical resources, the reconfiguration process needs to construct more resource partitions for each composite resource partition, leading to lower schedulability. Figure 16(c) shows that increasing the reconfiguration budget improves the schedulability. However, the improvement stops when the budget is larger than 4. This indicates that setting a higher budget may only waste computation time. Figure 16(d) shows that the schedulability increases when the percentage of partitions with effective reconfiguration regularity over 1 increases. Intuitively, the effective reconfiguration regularity denotes the tolerance of performance degradation during reconfiguration. Note that when all the partitions have effective reconfiguration regularity larger than 1, the schedulability will reach 100%. This is due to the assumption that the partitions can be constructed by ARCRP-S-Fast independently without considering the reconfiguration.

8 CONCLUSION

In this paper, we study the resource misalignment problem where a resource partition may undersupply when resource partitions are not aligned in non-uniform environments. We first introduce the the concept of composite resource partition, and then propose ARCRP-S and ARCRP-DR algorithms to construct and reconfigure the composite resource partitions to mitigate the problem. The key ideas of the algorithms are to avoid scheduling resource slice on any requesting time and to progressively construct or reconfigure the partition schedules on each physical resource. Extensive experiments are conducted to demonstrate the applicability of our proposed multi-resource scheduling framework through both real-world system implementation and simulation studies.

REFERENCES

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

[2] Matthias M. Herterich, Falk Uebernickel, and Walter Brenner. 2015. The impact of cyber-physical systems on industrial services in manufacturing. *Procedia Cirp* 30 (2015), 323–328.

[3] Alex Xiang Feng. 2004. *Design of Real-time Virtual Resource Architecture for Largescale Embedded Systems*. Ph.D. Dissertation. Department of Computer Science, The University of Texas at Austin.
[4] Jalil Boudjadar, Jin Hyun Kim, Linh Thi Xuan Phan, Insup Lee, Kim G. Larsen, and Ulrik Nyman. 2018. Generic formal framework for compositional analysis of hierarchical scheduling systems. In *21st IEEE International Symposium on Real-Time Distributed Computing (ISORC)*. IEEE, 51–58.
[5] Yu Li and Albert M. K. Cheng. 2017. Toward a practical regularity-based model: The impact of evenly distributed temporal resource partitions. *ACM Transactions on Embedded Computing Systems (TECS)* 16, 4 (2017), 111.
[6] Insik Shin and Insup Lee. 2003. Periodic resource model for compositional real-time guarantees. In *24th IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2–13.
[7] Arvind Easwaran, Madhukar Anand, and Insup Lee. 2007. Compositional analysis framework using EDP resource models. In *28th IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 129–138.
[8] Yu Li and Albert Mo Kim Cheng. 2015. Transparent real-time task scheduling on temporal resource partitions. *IEEE Trans. Comput.* 65, 5 (2015), 1646–1655.
[9] Yu Li and Albert M. K. Cheng. 2012. Static approximation algorithms for regularity-based resource partitioning. In *33rd IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 137–148.
[10] Sanjoy K. Baruah, Neil K. Cohen, C. Greg Plaxton, and Donald A. Varvel. 1996. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica* 15, 6 (1996), 600–625.
[11] Aloysius K. Mok and Xiang Feng. 2001. Towards compositionality in real-time resource partitioning based on regularity bounds. In *22nd IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 129–138.
[12] Wei-Ju Chen, Peng Wu, Pei-Chi Huang, Aloysius K. Mok, and Song Han. 2021. Online reconfiguration of regularity-based resource partitions in cyber-physical systems. *Real-Time Systems* (2021), 1–44.
[13] S. Shirero, Matsumoto Takashi, and Hiraki Kei. 1999. On the schedulability conditions on partial time slots. In *International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*.
[14] Luca Abeni and Giorgio Buttazzo. 2004. Resource reservation in dynamic real-time systems. *Real-Time Systems* 27, 2 (2004), 123–167.
[15] Luca Abeni, Luigi Palopoli, Claudio Scordino, and Giuseppe Lipari. 2009. Resource reservations for general purpose applications. *IEEE Transactions on Industrial Informatics* 5, 1 (2009), 12–21.
[16] Giorgio Buttazzo, Enrico Bini, and Yifan Wu. 2010. Partitioning parallel applications on multiprocessor reservations. In *22nd Euromicro Conference on Real-Time Systems (ECRTS)*.
[17] Giorgio Buttazzo, Enrico Bini, and Yifan Wu. 2011. Partitioning real-time applications over multicore reservations. *IEEE Trans. on Industrial Informatics* 7, 2 (2011), 302–315.
[18] Sparsh Mittal and Jeffrey S. Vetter. 2015. A survey of CPU-GPU heterogeneous computing techniques. *Comput. Surveys* 47, 4 (2015), 1–35.
[19] Cristinel Ababei and Milad Ghorbani Moghaddam. 2018. A survey of prediction and classification techniques in multicore processor systems. *IEEE Trans. on Parallel and Distributed Systems* 30, 5 (2018), 1184–1200.
[20] Gregory F. Diamos and Sudhakar Yalamanchili. 2008. Harmony: An execution model and runtime for heterogeneous many core systems. In *Proceedings of the 17th International Symposium on High Performance Distributed Computing*. 197–200.
[21] Cédric Augonnet, Samuel Thibault, Raymond Namyst, and Pierre-André Wacrenier. 2011. StarPU: A unified platform for task scheduling on heterogeneous multicore architectures. *Concurrency and Computation: Practice and Experience* 23, 2 (2011), 187–198.
[22] Janghaeng Lee, Mehrzad Samadi, Yongjun Park, and Scott Mahlke. 2015. SKMD: Single kernel on multiple devices for transparent CPU-GPU collaboration. *ACM Transactions on Computer Systems (TOCS)* 33, 3 (2015), 1–27.
[23] Yasir Noman Khalid, Muhammad Aleem, Usman Ahmed, Muhammad Arshad Islam, and Muhammad Azhar Iqbal. 2019. Troodon: A machine-learning based load-balancing application scheduler for CPU–GPU system. *J. Parallel and Distrib. Comput.* 132 (2019), 79–94.
[24] Yuan Wen, Zheng Wang, and Michael F. P. O'Boyle. 2014. Smart multi-task scheduling for OpenCL programs on CPU/GPU heterogeneous platforms. In *2014 21st International Conference on High Performance Computing (HiPC)*. IEEE, 1–10.
[25] Kapil Dev and Sherief Reda. 2016. Scheduling challenges and opportunities in integrated CPU+ GPU processors. In *the 14th ACM/IEEE Symposium on Embedded Systems for Real-Time Multimedia*. 78–83.
[26] Meng Xu, Linh Thi Xuan Phan, Oleg Sokolsky, Sisu Xi, Chenyang Lu, Christopher Gill, and Insup Lee. 2015. Cache-aware compositional analysis of real-time multicore virtualization platforms. *Real-Time Systems* 51, 6 (2015), 675–723.
[27] Meng Xu, Linh Thi, Xuan Phan, Hyon-Young Choi, and Insup Lee. 2017. vCAT: Dynamic cache management using CAT virtualization. In *2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 211–222.

[28] Wei-Ju Chen, Pei-Chi Huang, Quan Leng, Aloysius K. Mok, and Song Han. 2017. Regular composite resource partition in open systems. In *38th IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 34–44.
[29] Alan Burns and Robert I. Davis. 2018. A survey of research into mixed criticality systems. *Comput. Surveys* 50, 6 (2018), 82.
[30] Zhe Jiang, Neil Audsley, Pan Dong, Nan Guan, Xiaotian Dai, and Lifeng Wei. 2019. MCS-IOV: Real-time i/o virtualization for mixed-criticality systems. In *2019 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 326–338.
[31] Moritz Neukirchner, Kai Lampka, Sophie Quinton, and Rolf Ernst. 2013. Multi-mode monitoring for mixed-criticality real-time systems. In *2013 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*. IEEE, 1–10.
[32] Dionisio de Niz and Linh T. X. Phan. 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, 111–122.
[33] Alan Burns. 2014. System mode changes-general and criticality-based. In *Proc. of 2nd Workshop on Mixed Criticality Systems (WMC)*. 3–8.
[34] Xiaozhe Gu and Arvind Easwaran. 2016. Dynamic budget management with service guarantees for mixed-criticality systems. In *2016 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 47–56.
[35] Chiristos Evripidou and A. Burns. 2016. Scheduling for mixed-criticality hypervisor systems in the automotive domain. In *WMC 2016 4th International Workshop on Mixed Criticality Systems*.
[36] Biao Hu, Lothar Thiele, Pengcheng Huang, Kai Huang, Christoph Griesbeck, and Alois Knoll. 2018. FFOB: Efficient online mode-switch procrastination in mixed-criticality systems. *Real-Time Systems* (2018), 1–43.
[37] Tianyang Chen and Linh Thi Xuan Phan. 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, 105–116.
[38] Hao Xu and Alan Burns. 2019. A semi-partitioned model for mixed criticality systems. *Journal of Systems and Software* 150 (2019), 51–63.
[39] Linh T. X. Phan, Insup Lee, and Oleg Sokolsky. 2010. Compositional analysis of multi-mode systems. In *22nd Euromicro Conference on Real-Time Systems (ECRTS)*. IEEE, 197–206.
[40] Haoran Li, Meng Xu, Chong Li, Chenyang Lu, Christopher Gill, Linh Phan, Insup Lee, and Oleg Sokolsky. 2018. Multi-mode virtualization for soft real-time systems. In *24th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 117–128.
[41] Vladimir Nikolov, Stefan Wesner, Eugen Frasch, and Franz J. Hauck. 2017. A hierarchical scheduling model for dynamic soft-realtime system. In *29th Euromicro Conference on Real-Time Systems (ECRTS)*.
[42] Pavan Kumar Paluri, Guangli Dai, and Albert Mo Kim Cheng. 2021. ARINC 653-inspired regularity-based resource partitioning on Xen. In *Proceedings of the 22nd ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems*. 134–145.
[43] A. K. Mok, L. Rosier, I. Tulchinsky, and D. Varvel. 1989. Algorithms and complexity of the periodic maintenance problem. *Microprocessing and Microprogramming* 27, 1 (1989), 657–664.
[44] Wei-Ju Chen, Peng Wu, Pei-Chi Huang, Aloysius K. Mok, and Song Han. 2019. Online reconfiguration of regularity-based resource partitions in cyber-physical systems. In *2019 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 495–507.
[45] Song Han, Deji Chen, Ming Xiong, Kam-Yiu Lam, Aloysius K. Mok, and Krithi Ramamritham. 2012. Schedulability analysis of deferrable scheduling algorithms for maintaining real-time data freshness. *IEEE Trans. Comput.* 63, 4 (2012), 979–994.
[46] Yi-Hung Wei, Quan Leng, Song Han, Aloysius K. Mok, Wenlong Zhang, and Masayoshi Tomizuka. 2013. RT-WiFi: Real-time high-speed communication protocol for wireless cyber-physical control applications. In *Real-Time Systems Symposium*. IEEE, 140–149.
[47] J. C. Eidson. 2010. *Measurement, Control, and Communication using IEEE 1588*. Springer.

Received 30 August 2022; revised 18 May 2023; accepted 14 June 2023