Radar: Reliable Resource Scheduling for Composable/Disaggregated Data Centers

Chao Guo, Moshe Zukerman, *Life Fellow, IEEE*, and Tianjiao Wang

***Abstract*—Hardware disaggregation decouples resources (e.g., processors and memory) from monolithic servers, potentially improving service reliability. However, from another perspective, directly exposing resource modules to a shared network may adversely affect service reliability. In this paper, we study a reliable resource allocation problem in disaggregated DCs (DDCs), considering network impact and different disaggregation scales. We provide a mixed-integer linear programming formulation and a resource allocation framework named Radar for this problem. Numerical results demonstrate that the benefits of hardware disaggregation may be adversely affected by an imperfect network. It also shows that both the hardware backup and a proposed migration-based restoration can be applied to overcome this potential adverse effect.**

***Index Terms*—Composable/disaggregated infrastructure, hardware disaggregation, data center, reliability, network**

I. Introduction

WITH the rapid growth of internet technology like big data and cloud/fog/edge computing, vast amounts of data are poured into the data center (DC), imposing a significant burden on data centers [1], [2]. Considerable effort has been made to ease this burden through various aspects, including resource allocation [3-5], energy efficiency [6-9], and thermal issues [10], [11], which are for server-based architecture. This architecture has caused significant resource stranding, hindering efficiency improvement [12].

Composable/disaggregated infrastructure (CDI), which "uses an API to create physical systems from shared pools of resources" [13], is an emerging computing infrastructure for future data centers (DCs) to break through the boundary limits of traditional servers [14], [15]. In its current early stage, CDI has grown fast due to its high benefits, which is expected to grow to $13.5 billion by 2027 with an annual growth rate of 21 percent [16]. This paper refers to a CDI-based DC as a *disaggregated* DC (DDC), which represents a DC employing hardware disaggregation. Hardware disaggregation decouples resources (e.g., processors and memory) from integrated servers and reassembles them into resource pools interconnected through a fast network, converting a server-based DC (SDC) into a DDC. New techniques like non-volatile memory express [13], compute express link [13], computational storage [17], and the advancement in optical interconnection [18-20] provide diversified support for communication solutions to hardware disaggregation. DDCs achieve resource ef[1]ficiency and flexibility by reducing resource stranding in the SDC [12], [21-25]. Hardware upgrades and resource expansion become cost-efficient since they can be operated at the component level instead of the server level [22].

In this paper, we focus on service reliability in DDCs. Providing high service reliability is critical for DCs to provide continuous operations, ensuring high quality of services, while unreliable service may lead to severe economic loss [26]. Compared to SDCs, resource availability and reliability in DDCs are potentially improved for two main reasons. Firstly, improved flexibility expands the optimization regions [27], and secondly, resource decoupling leads to a less harmful failure pattern where failures among different resources may not implicate each other as in SDCs [28]. However, disaggregation may also adversely affect service reliability as resource modules are now directly exposed to a shared network. The failure of the shared network may lead to the unavailability of many resource modules.

In addition, due to the strict latency and bandwidth requirements of inter-resource communications, e.g., CPU-memory communication, the scale of disaggregation is limited [29]. Most efforts on resource disaggregation have considered rack-scale, where a resource, e.g., CPU, can use a different resource, e.g., memory, from the same rack but not from a different rack. Although several publications considered pod/DC-scale disaggregation [29], [30], their practical application is limited to only a few cases.

In this paper, we study the reliability performances in DDCs considering network impact and different disaggregation scales. We summarize the key novelty and main contributions of this paper as follows.

- We study the problem of reliable resource allocation for DDCs considering network effects and different disaggregation scales. In addressing this problem, we aim to achieve both high resource efficiency and high reliability. We consider both static and dynamic scenarios. For the static scenario, the resources are allocated to a batch of known requests. We aim to maximize the reliability of each accepted request and the acceptance ratio, defined as the ratio of the number of accepted requests to the total number of requests. For the dynamic scenario, where requests arrive and depart randomly,

This work was supported by a grant from the City University of Hong Kong, Hong Kong Special Administrative Region (CityU 9229020). (Corresponding author: *C. Guo*.)

The authors are with the Department of Electrical Engineering, City University of Hong Kong, Hong Kong SAR. Email: chaoguo6-c@my.cityu.edu.hk; moshezu@gmail.com; tianjwang6-c@my.cityu.edu.hk;

resources are allocated to each request upon its arrival and are released at its departure. In addition, hardware failures occur over time, and each failure is fixed after a certain time to repair. Hardware failures interrupt their hosted requests, resulting in the requests failing to complete their service. The objectives in the dynamic scenario are to minimize the blocking probability (one minus the acceptance ratio) and the number of accepted requests failing to complete their service.

- We provide mixed-integer linear programming (MILP) formulations to solve the multi-objective problem in the static scenario by converting it into a single-objective problem by the weighted sum approach. We first provide a MILP formulation for a DC-scale DDC and then extend it to MILP formulations for an SDC and a rack/pod-scale DDC. We provide approximate Pareto fronts by solving the MILP with varied weights in the objective function.
- We propose *Radar*, a framework for **r**eli**a**ble resource allocation in **d**isaggregated d**a**ta cente**r**s, which considers both static and dynamic scenarios. We provide a heuristic algorithm for the static scenario that can solve the problem at a significantly lower complexity than the MILP. For the dynamic scenario, two heuristic algorithms are provided, with one applied for scheduling the arrival requests, and the other applied when a failure occurs to restore the interrupted requests by migrating them elsewhere.
- We provide extensive numerical results for the performance analyses. These numerical results are obtained by solving the MILP formulations and simulation studies. Numerical results demonstrate that the realistically imperfect network may significantly offset the reliability improvement brought by hardware disaggregation under the idealistic resilient assumption. Then, we also demonstrate that backup or a proposed migration-based restoration can overcome this weakness of disaggregation.

Next, we present related work in Section II and introduce the disaggregated architectures in Section III. Then, we describe the reliable resource allocation problem in DDCs in Section IV. Sections IV and V provide the MILP formulation and Radar framework, respectively. The performance evaluation is presented in Section VI, and Section VII concludes the paper.

II. Related Work

Some publications are dedicated to resource allocation for DDCs, though they do not consider reliability. Papaioannou *et al.* [25] proposed a heuristic algorithm for virtual machine (VM) placement for a rack-scale DDC to demonstrate that resource utilization in DCs can be significantly improved with resource disaggregation. They considered two specific resources, i.e., CPU and RAM. Similarly, Ali *et al.* [30] proposed a DC-scale VM placement method for a DC-scale DDC to minimize energy consumption. This paper considers CPU, memory, and IO resources. Zervas *et al.* [18] proposed a dedicated architecture for memory disaggregation, and they also provided VM placement algorithms based on different strategies: first-fit, best-fit, network-unaware locality-based, and network-aware locality-based. Ajibola *et al.* [31] provided MILP formulation for an energy-efficient workload scheduling, and they considered both rack-scale and pod-scale DDC architectures. They also considered specific resource types in their formulation. Pagès *et al.* [20] studied the virtual data center embedding problem for a rack-scale DDC to improve resource utilization. There are also several other works related to resource allocation or request scheduling for DDCs, e.g., [24], [32], and [33]. However, none of these studies considered reliability, and they considered a single objective mainly focusing on utilization. Moreover, their methods are limited to specific resources, e.g., CPU and memory.

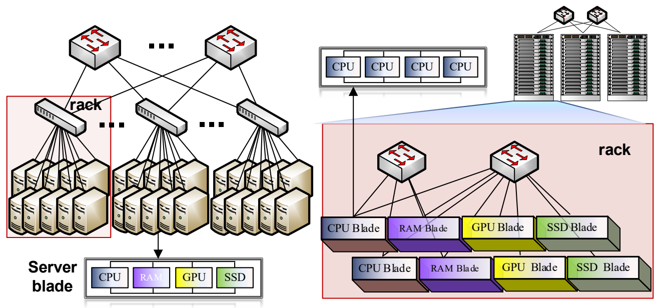
Several publications are dedicated to reliability issues in DDCs, though not to resource allocation. Shan *et al.* [12] incorporated the RAID-style memory replication as the fault tolerance mechanism in their proposed operating system (OS) for the DDC. Carbonari *et al.* [28] analyzed four failure-sharing models in the DDC and claimed that applications should be allowed to choose their preferred recovery models. Based on two attributes specific to the disaggregated architecture, namely, the ability to reassign memory and failure independence, Angel *et al.* [34] proposed several primitive operations for a potential OS dedicated to DDCs.

In the literature, we found only two papers that consider reliability or availability in resource allocation for DDCs, and the first one is the work done by Ferreira *et al.* [35]. This work aims to maximize the availability of the application and minimize the total cost when allocating resources. However, this work is different from our work. They considered that a user requires a specific number of resource components of each type, e.g., the requirement of three CPU chips and two memory modules. By comparison, we allow one component to be used by multiple requests, and each request comes with a given resource demand, e.g., 10 GB of memory demand. The second work was done by Guo *et al.* [27], who considered the resource allocation form similar to ours. They provided a resource allocation method for request scheduling to maximize the acceptance ratio subject to meeting requests' requirements. However, both [27] and [35] assumed a resilient network and a DC-scale disaggregation, which may be unrealistic, and a DC-scale DDC may not be able to support some applications because of latency and bandwidth requirements of inter-resource communications. Different from them, we consider different disaggregation scales and study the impact of a network that is not resilient. We also consider the latency and bandwidth requirement of inter-resource communications, which they did not consider.

III. Overview of SDC vs. DDC Architectures

In this section, we introduce the DDC architectures we consider for the reliable resource allocation problem (which is defined in the next section). We also discuss the reliability challenges of these DDC architectures.

Fig. 1 (a) shows an SDC architecture, where computing and storage resources are packaged in server blades, each containing different resources, and the onboard circuit provides communications among different resources. These servers are interconnected through a DC network (DCN). Traditional DCNs are interconnected through wired links and are organized in a tree-like topology, e.g., leaf-spine [36] topology, like the one shown in Fig. 1(a). Recently, wireless communications techniques have also been introduced to reduce the reconfiguration complexity [37].



(a) Server-based DC architecture (b) Rack-scale disaggregated DC architecture

Fig. 1. Server-based and rack-scale disaggregated DC architectures.

Fig. 1(b) shows a rack-scale DDC architecture, where each blade contains homogeneous resources while a rack also contains heterogeneous resources [31], [32], [38]. A resource, e.g., CPU, can use a different resource, e.g., memory, from the same rack but not from a different rack. In DDCs, communications among different resources are completed through a network. For a DDC with a larger disaggregation scale, e.g., pod- or DC-scale, a rack contains homogeneous resources, and the usage among different types of resources is no longer restricted within a rack but can be across racks and restricted within a pod or DC [31].

Hardware disaggregation has reliability benefits but also challenges. In SDCs, resources in a server blade are interconnected through the motherboard, whose failure affects the blade itself but not other blades. While in a DDC, the failure of the shared network directly affects all the connected components. Also, resource pooling may become a challenge. Fig. 2 shows how the four workloads, W1-W4, are placed in two server blades (Fig. 2(a)) or two disaggregated resource blades (Fig. 2(b)). Each workload's CPU and memory demands can be satisfied by one server blade or two disaggregated resource blades. Therefore, each blade in the SDC hosts two workloads but four workloads in the DDC. When one blade fails, all workloads are interrupted in the DDC, but only two are interrupted in the SDC.

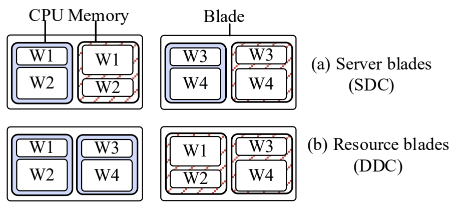


Fig. 2. Placement of four workloads in SDC vs. DDC.

To address the problem caused by resource pooling, we may disaggregate the resources thoroughly and make each hardware component a distinct node. However, this method shifts the reliability issue to the network and increases the number of nodes and network scale [39]. A more practical approach is needed through either hardware organization or software scheduling. Next, we provide detailed analyses and possible solutions through a novel resource allocation approach.

IV. Problem Statement and Request Reliability

A. Problem Description

We define the problem of resource allocation in DDCs as follows. We are given a DDC as an undirected graph consisting of nodes and links, and the nodes include switches and blades (See Fig. 1). We assume that DDCs employ optical circuit switches because they provide ultra-low and deterministic latency and guaranteed network bandwidth [18], [39], [40]. Each switch is characterized by a switching delay and reliability. Each blade contains a set of components, e.g., CPU components (See Fig. 1), and each component is characterized by its resource type, available capacity, and reliability. Like Carbonari *et al.* [28], we assume that different components in a blade fail independently. Each blade is also associated with delay and reliability, which are the delay between receiving and transmitting data and the reliability of the peripheral elements, such as interfaces and control circuits. Each link is characterized by available bandwidth, propagation delay, and reliability.

We use the term *request* to represent a possible form of resource allocation requests, including requests for jobs/tasks [4], virtual machines [41], [42], and virtual containers [43]. We consider the static and dynamic scenarios of problems regarding whether the requests arrive at once or randomly. Each request arrives with a given service time and a resource demand for each resource type. The request's inter-resource traffic demand and latency requirements are also given. As in [25], [31], [32], we assume that a request can only use one blade for one resource, and the disaggregation scale determines whether it can obtain different resources from different blades. Consider the rack-scale DDC as an example, where a request can only use memory from one blade but obtains different resources from multiple blades within one rack. Existing work such as [25], [31], [32] regards a blade in a DDC as a single node but does not consider components in the blade. We also consider how the resources from these components are allocated because we need to consider the failure independence among different components. Nevertheless, a request can obtain one resource from multiple components in a blade in a DDC.

The problem objectives and methodologies are different in the static and dynamic scenarios.

1) Static Scenario

In this scenario, all requests arrive at once. Our aim in this scenario is to maximize the acceptance ratio and each request's reliability. We provide a MILP formulation and a scalable heuristic algorithm to address the problem. The MILP formulation translates the problem into precise mathematical language, and it can be solved using commercial solvers to provide optimal solutions. The optimal solutions can be further used to validate the efficiency of the proposed algorithm.

2) Dynamic Scenario

In this scenario, requests arrive and leave randomly and sequentially. A request is accepted if the available resources are sufficient upon arrival and blocked otherwise. We assume no waiting room, and a blocked request is rejected and leaves the system without re-attempting. One of our objectives is to minimize the blocking probability (or maximize the acceptance ratio), defined as the ratio of the number of blocked requests to the total number of arrivals during a specified period. An accepted request may be interrupted by a hardware failure, failing to complete its service. Our other objective is to minimize the number of accepted requests failing to complete service. The blocking probability does not include the accepted requests interrupted by hardware failures. We provide a Radar framework to achieve the two objectives.

B. The Reliability of a Service Request

Assume that a request (denoted i) arrives, and denote the arrival and departure times of request i as t_i^a and t_i^d, respectively. Also, assume that an element (denoted e) was last repaired at the time t_e^{LR}. Let the random variable Δt be the time between failures (TBF) of element e. As commonly used, TBF is the time from the moment the element is repaired until it fails again. The probability that e does not fail during the service time of request i (denoted P_e^i) can be obtained by:

$$P_e^i = P\left(\Delta t > t_i^d - t_e^{LR} \middle| \Delta t > t_i^a - t_e^{LR}\right) = \frac{\mathcal{R}_e\left(t_i^d - t_e^{LR}\right)}{\mathcal{R}_e\left(t_i^a - t_e^{LR}\right)}, \tag{1}$$

where $\mathcal{R}_e(t)$ is the reliability of element e, i.e., the probability that the TBF of e is not shorter than t.

We consider the reliability of request i, denoted by $\mathcal{R}_i$, as the probability that request i encounters no hardware failure during its service time. This is equal to the probability that no element that serves it fails during its service time. Assuming independence of failures among different elements, we obtain,

$$\mathcal{R}_i = \prod_{e\in\boldsymbol{\mathcal{E}_i}} P_e^i, \tag{2}$$

where $\boldsymbol{\mathcal{E}_i}$ denotes the set of elements used by request i.

V. The MILP Formulation

A. MILP Formulation for a DC-Scale DDC

We first introduce the MILP for a DDC of DC-scale and later extend it to other scales. Table I provides our used notations.

TABLE I
List of Notations

Notation	Explanation
$\boldsymbol{R}$	Set of resource types, e.g., CPU and memory
$\boldsymbol{N}$	Set of blades
$\boldsymbol{X}$	Set of switches
$\boldsymbol{NE_n}$	Set of nodes neighboring to node $n \in \boldsymbol{N} \cup \boldsymbol{X}$
$\boldsymbol{\mathcal{C}_n}$	Set of (resource) components in blade n
$\boldsymbol{I}$	Set of requests
θ_{nc}^r	Binary parameter indicating whether component c in blade n is of resource type $r \in \boldsymbol{R}$
A_{nc}	Available capacity of component c in blade n
B_{mn}	Available bandwidth of the link (m,n)
τ_n^{bld}	Delay of blade n
τ_n^{sw}	Switching delay of switch n
τ_{mn}^{pro}	Propagation delay of the link (m,n)
$\xi_{mn}^{r_1r_2}$	Binary parameter denoting whether the traffic of (r_1,r_2), $r_1,r_2 \in \boldsymbol{R}$ is allowed to traverse the link (m,n)
P_{nc}^i	The probability that component c in blade $n \in \boldsymbol{N}$ does not fail during the service time of request i
P_n^i	The probability that the blade or switch n does not fail during the service time of request i
P_{mn}^i	The probability that link (m,n) does not fail during the service time of request i
D_{ir}	Resource demand of request i for resource r
$\lambda_{r_1r_2}^i$	Traffic demand of resource pair (r_1,r_2) in request i
$\Delta\tau_{r_1r_2}^i$	Latency requirement of traffic (r_1,r_2) in request i Note that the requirement is for the extra latency traversing the network relative to the onboard latency in a server
∇	A large value
α	A weight factor
Decision Variables	
π_i	(Binary) Equal one if request i is accepted; zero otherwise
δ_n^{ir}	(Binary) Equal one if request i gets resource r from the blade n; zero otherwise
$\zeta_n^{ir_1r_2}$	(Binary) Equal one if request i gets both resources r_1 and r_2 from blade n; zero otherwise
μ_{nc}^{ir}	(Real) The amount of resource r that request i gets from component c in blade n
$\gamma_{mn}^{ir_1r_2}$	(Binary) Equal one if the traffic of (r_1,r_2) in request i traverses link (m,n); zero otherwise
$\omega_n^{ir_1r_2}$	(Binary) Equal one if the traffic of (r_1,r_2) in request i traverses switch n; zero otherwise
ρ_n^i	(Binary) Equal one if request i uses switch or blade node n ($n \in \boldsymbol{N} \cup \boldsymbol{X}$); zero otherwise
ϱ_{nc}^i	(Binary) Equal one if request i uses component c in blade n ($n \in \boldsymbol{N}$); zero otherwise
χ_{mn}^i	(Binary) Equal one if request i uses link (m,n); zero, otherwise
Λ_i	(Real) The logarithm of the reliability of request i, i.e., $\Lambda_i = \log(\mathcal{R}_i)$, where $\mathcal{R}_i$ is the reliability of the request
Λ^{min}	(Real) The minimum value of all $\Lambda_i, i \in \boldsymbol{I}$

Our objective is to maximize the weighted sum of the minimum request reliability and acceptance ratio, formulated as:

$$\textbf{Maximize}: (1-\alpha)\cdot\frac{\sum_{i\in\boldsymbol{I}}\pi_i}{|\boldsymbol{I}|} + \alpha\cdot\Lambda^{min}. \tag{3}$$

The decision variables in (3) are defined in Table I. Note that some decision variables in the table are not included in the objective function but are included in the constraints.

- <u>*Blade and component allocation constraints*</u>

$$\pi_i = \sum_{n\in\boldsymbol{N}} \delta_n^{ir}\ \forall i \in \boldsymbol{I}, r \in \boldsymbol{R} \tag{4}$$

$$\nabla\cdot\delta_n^{ir} \geq \mu_{nc}^{ir}\ \forall i \in \boldsymbol{I}, r \in \boldsymbol{R}, n \in \boldsymbol{N}, c \in \boldsymbol{\mathcal{C}_n} \tag{5}$$

$$\delta_n^{ir} \leq \nabla\cdot\sum_{c\in\boldsymbol{\mathcal{C}_n}} \mu_{nc}^{ir}\ \forall i \in \boldsymbol{I}, r \in \boldsymbol{R}, n \in \boldsymbol{N} \tag{6}$$

$$\sum_{n\in\boldsymbol{N}, c\in\boldsymbol{\mathcal{C}_n}} \mu_{nc}^{ir} = D_{ir}\cdot\pi_i\ \forall i \in \boldsymbol{I}, r \in \boldsymbol{R} \tag{7}$$

$$\sum_{i\in\boldsymbol{I}} \mu_{nc}^{ir} \leq A_{nc}\cdot\theta_{nc}^r\ \forall n \in \boldsymbol{N}, c \in \boldsymbol{\mathcal{C}_n}, r \in \boldsymbol{R} \tag{8}$$

- <u>*Traffic scheduling constraints*</u>

$$\sum_{i\in\boldsymbol{I}, r_1,r_2\in\boldsymbol{R}: r_1\neq r_2} \gamma_{mn}^{ir_1r_2}\cdot\lambda_{r_1r_2}^i \leq B_{mn}$$
$$\forall m \in \boldsymbol{N}\cup\boldsymbol{X}, n \in \boldsymbol{NE_m} \tag{9}$$

$$\sum_{n\in\boldsymbol{NE_m}} \gamma_{nm}^{ir_1r_2}\cdot\lambda_{r_1r_2}^i - \sum_{n\in\boldsymbol{NE_m}} \gamma_{mn}^{ir_1r_2}\cdot\lambda_{r_1r_2}^i = \begin{cases}\lambda_{r_1r_2}^i\cdot\left(\delta_m^{ir_2} - \delta_m^{ir_1}\right), & m \in \boldsymbol{N} \\ 0, & m \in \boldsymbol{X}\end{cases}\ \forall\, i \in \boldsymbol{I}, r_1, r_2 \in \boldsymbol{R}: r_1 \neq r_2 \tag{10}$$

$$\gamma_{mn}^{ir_1r_2} \leq \xi_{mn}^{r_1r_2}\ \forall m \in \boldsymbol{X}\cup\boldsymbol{N}, n \in \boldsymbol{NE_m}, i \in \boldsymbol{I}, r_1, r_2 \in \boldsymbol{R} \tag{11}$$

$$\gamma_{mn}^{ir_1r_2} = \gamma_{nm}^{ir_2r_1}\ \forall\, i \in \boldsymbol{I}, r_1, r_2 \in \boldsymbol{R}, m \in \boldsymbol{N}\cup\boldsymbol{X}, n \in \boldsymbol{NE_m} \tag{12}$$

$$\gamma_{nm}^{ir_1r_2} + \gamma_{mn}^{ir_1r_2} \leq 1 \forall i \in \boldsymbol{I}, r_1, r_2 \in \boldsymbol{R}, n \in \boldsymbol{N}\cup\boldsymbol{X}, m \in \boldsymbol{NE_n} \tag{13}$$

$$\omega_n^{ir_1r_2} \geq \gamma_{mn}^{ir_1r_2}\ \forall n \in \boldsymbol{X}, m \in \boldsymbol{NE_n}, i \in \boldsymbol{I}, r_1, r_2 \in \boldsymbol{R} \tag{14}$$

$$\omega_n^{ir_1r_2} \geq \gamma_{nm}^{ir_1r_2}\ \forall n \in \boldsymbol{X}, m \in \boldsymbol{NE_n}\ i \in \boldsymbol{I}, r_1, r_2 \in \boldsymbol{R} \tag{15}$$

$$\omega_n^{ir_1r_2} \le \sum\nolimits_{m\in \boldsymbol{NE_n}} \left(\gamma_{nm}^{ir_1r_2} + \gamma_{mn}^{ir_1r_2}\right) \forall n \in \boldsymbol{X}, i \in \boldsymbol{I}, r_1, r_2 \in \boldsymbol{R} \quad (16)$$

$$\lambda_{r_1r_2}^i \cdot \left(\sum_{m\in \boldsymbol{N}\cup\boldsymbol{X}, n\in\boldsymbol{NE_m}} \gamma_{mn}^{ir_1r_2}\cdot\tau_{mn}^{pro} + \sum_{n\in\boldsymbol{X}} \omega_n^{ir_1r_2}\cdot\tau_n^{sw} + \sum\nolimits_{n\in\boldsymbol{N}} \left(\delta_n^{ir_1} + \delta_n^{ir_2} - 2\cdot\zeta_n^{ir_1r_2}\right)\cdot\tau_n^{bld} \right) \le \lambda_{r_1r_2}^i \cdot \Delta\tau_{r_1r_2}^i, \forall\, i\in\boldsymbol{I}, r_1, r_2 \in \boldsymbol{R}: r_1 \ne r_2 \quad (17)$$

$$\begin{cases} \zeta_n^{ir_1r_2} \le \delta_n^{ir_1} \\ \zeta_n^{ir_1r_2} \le \delta_n^{ir_2} \\ \zeta_n^{ir_1r_2} \ge \delta_n^{ir_1} + \delta_n^{ir_2} - 1 \end{cases} \forall n \in \boldsymbol{N}, i\in\boldsymbol{I}, r_1, r_2 \in \boldsymbol{R} \quad (18)$$

- <u>*Reliability-related constraints*</u>

$$\Lambda_i = \sum\nolimits_{n\in\boldsymbol{N}\cup\boldsymbol{X}} \rho_n^i \cdot \log P_n^i + \sum\nolimits_{n\in\boldsymbol{N}, c\in\boldsymbol{C_n}} \varrho_{nc}^i \cdot \log P_{nc}^i + \sum\nolimits_{m\in\boldsymbol{N}\cup\boldsymbol{X}, n\in\boldsymbol{NE_m}} \chi_{mn}^i \cdot \log P_{mn}^i + (\pi_i - 1)\cdot\nabla \;\forall i \in \boldsymbol{I} \quad (19)$$

$$\Lambda^{min} \le \pi_i \cdot \Lambda_i + (1-\pi_i)\cdot\nabla \;\forall i\in\boldsymbol{I} \quad (20)$$

$$\chi_{mn}^i \ge \gamma_{mn}^{ir_1r_2} \;\forall n \in \boldsymbol{N}\cup\boldsymbol{X}, m\in\boldsymbol{NE_n}, i\in\boldsymbol{I}, r_1, r_2\in\boldsymbol{R} \quad (21)$$

$$\chi_{mn}^i \le \sum_{r_1,r_2\in\boldsymbol{R}: r_1\ne r_2} \gamma_{mn}^{ir_1r_2} \;\forall n\in\boldsymbol{N}\cup\boldsymbol{X}, m\in\boldsymbol{NE_n}, i\in\boldsymbol{I} \quad (22)$$

$$\rho_n^i \ge \omega_n^{ir_1r_2} \;\forall n\in\boldsymbol{X}, i\in\boldsymbol{I}, r_1, r_2\in\boldsymbol{R} \quad (23)$$

$$\rho_n^i \le \sum\nolimits_{r_1,r_2\in\boldsymbol{R}: r_1\ne r_2} \omega_n^{ir_1r_2} \;\forall n\in\boldsymbol{X}, i\in\boldsymbol{I} \quad (24)$$

$$\rho_n^i \ge \delta_n^{ir} \;\forall n\in\boldsymbol{N}, i\in\boldsymbol{I}, r\in\boldsymbol{R} \quad (25)$$

$$\rho_n^i \le \sum\nolimits_{r\in\boldsymbol{R}} \delta_n^{ir} \;\forall n\in\boldsymbol{N}, i\in\boldsymbol{I} \quad (26)$$

$$\varrho_{nc}^i \cdot \nabla \ge \mu_{nc}^{ir} \;\forall i\in\boldsymbol{I}, r\in\boldsymbol{R}, n\in\boldsymbol{N}, c\in\boldsymbol{C_n} \quad (27)$$

$$\varrho_{nc}^i \le \nabla \cdot \sum\nolimits_{r\in\boldsymbol{R}} \mu_{nc}^{ir} \;\forall i\in\boldsymbol{I}, n\in\boldsymbol{N}, c\in\boldsymbol{C_n}. \quad (28)$$

<u>Explanations:</u>

Blade and component allocation constraints: Constraint (4) ensures that every resource that serves a request is from only one blade. Constraints (5) and (6) ensure that a blade is used by request i as long as a component hosted by this blade is used by request i. Constraint (7) ensures that the amount of resources required by an accepted request is equal to the amount of resources allocated to it. Constraint (8) ensures no violation of the component capacity restriction.

Traffic scheduling constraints: Constraint (9) ensures that the link capacity restriction is not violated. Constraint (10) is the flow-conservation constraint for routing the traffic between each resource pair for each request. Constraint (11) ensures that traffic cannot traverse an unpermitted link. This constraint is used for the case when some links are for dedicated communications as required by some architectures [25]. Constraint (12) ensures that the links are bidirectional. Constraint (13) ensures that each traffic stream can only use one direction but cannot use both directions of a link to avoid wastage of communication resources (e.g., creating an unnecessary cycle). Constraints (14) - (16) ensure that a switch is used by a resource pair if any link connected to this switch carries the traffic. Constraint (17) ensures no violation of the traffic latency requirement, where the left-hand side is the traffic latency which is the summation of propagation, switching, and blade delays. The blade delay is $\sum_{n\in\boldsymbol{N}} \tau_n^{bld} \cdot \left(\delta_n^{ir_1} + \delta_n^{ir_2} - 2\cdot\zeta_n^{ir_1r_2}\right)$. Notice that if the source and destination share the same blade, the blade latency equals zero. Constraint (18) ensures that when a blade allocates r_1 and r_2 to a request, the two resources in the request share the blade.

Reliability-related constraints: Constraint (19) ensures that the reliability of each accepted request is correctly calculated. The last term on the right-hand side of (19) is to avoid the reliability of a rejected request being 1, which may distort output information. Constraint (20) ensures that the minimum reliability (log form) is no larger than that of each accepted request. Constraints (21) – (22) ensure that a link is used by a request when it carries the traffic of the request. Constraints (23) – (24) ensure that a switch node is used by a request when it switches the requested traffic. Constraints (25) – (26) ensure that a blade is used by a request when it provisions resources to this request. Constraints (27) – (28) ensure that a component is used by a request when it provisions resources to this request.

The complexity of both the number of dominant variables ($\gamma_{mn}^{ir_1r_2}$) and the dominant number of constraints (11) is given by $O(|\boldsymbol{I}|\cdot|\boldsymbol{R}|^2\cdot|\boldsymbol{N}\cup\boldsymbol{X}|^2)$.

B. MILP Formulation for a Blade-Scale DDC (SDC)

The following constraints are added to extend the DC-scale DDC to a blade-scale DDC.

$$\sum\nolimits_{n\in\boldsymbol{N}} \rho_n^i = \pi_i \;\forall i\in\boldsymbol{I} \quad (29)$$

$$\zeta_n^{ir_1r_2} = \rho_n^i \;\forall i\in\boldsymbol{I}, r_1, r_2\in\boldsymbol{R}, n\in\boldsymbol{N}: r_1\ne r_2 \quad (30)$$

$$\gamma_{mn}^{ir_1r_2} = 0 \;\forall\, i\in\boldsymbol{I}, r_1, r_2\in\boldsymbol{R}, m\in\boldsymbol{N}\cup\boldsymbol{X}, n\in\boldsymbol{NE_m} \quad (31)$$

$$\chi_{mn}^i = 0 \;\forall\, i\in\boldsymbol{I}, m\in\boldsymbol{N}\cup\boldsymbol{X}, n\in\boldsymbol{NE_m} \quad (32)$$

$$\omega_n^{ir_1r_2} = 0 \;\forall i\in\boldsymbol{I}, r_1, r_2\in\boldsymbol{R}, n\in\boldsymbol{X}: r_1\ne r_2. \quad (33)$$

Constraint (29) ensures that each accepted request can only use resources from a single blade, and constraint (30) ensures that all the resources required by a request share one blade. Constraint (31) ensures that, in an SDC, each traffic stream does not use network links as it is done locally. Constraints (32) – (33) ensure that an entire request does not use links or switches in an SDC.

C. MILP Formulation for a Rack- or Pod-Scale DDC

Here, a request can only use resources from a single rack (or pod, same as below) in a rack-scale DDC but cannot use resources from different racks. Let Γ be the set of racks and ς_{nk} be a binary parameter indicating whether the blade n is in rack k. Define a binary variable β_k^i that equals one if request i uses blade in rack k; otherwise, zero. The following constraints should be added to extend the MILP for DC-scale DDC to the rack-scale DDC.

$$\sum\nolimits_{k\in\Gamma} \beta_k^i = \pi_i \;\forall i\in\boldsymbol{I} \quad (34)$$

$$\beta_k^i \ge \rho_n^i \cdot \varsigma_{nk} \;\forall i\in\boldsymbol{I}, n\in\boldsymbol{N}, k\in\Gamma \quad (35)$$

$$\beta_k^i \le \sum\nolimits_{n\in\boldsymbol{N}} \rho_n^i \cdot \varsigma_{nk} \;\forall i\in\boldsymbol{I}, k\in\Gamma. \quad (36)$$

Constraint (34) ensures that each accepted request can only use resources from a single rack. Constraints (35) – (36) ensure that a rack is used by request i as long as request i uses blades in the rack.

VI. Framework Overview and Heuristic Algorithms

A. Framework Description

As shown in Fig. 3, the Radar framework consists of a scheduler, monitor, and physical resource modules. The monitor module detects topology and load changes as well as hardware failures and repairs and periodically reports the information to the scheduler module to assist in decision-making. The scheduler module executes appropriate algorithms based on the requests and hardware information to make a decision and finally sends the decision information to the physical DDC for further operation.

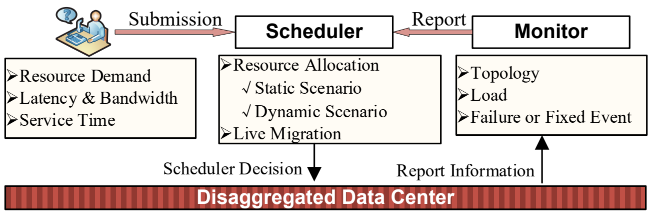


Fig. 3. Radar structure.

As noted in Fig. 3, the framework addresses both static and dynamic scenarios. As the dynamic scenario is characterized by individual requests that arrive over time, the scheduler assumes that it is a dynamic scenario if the first arrival batch comprises a single request. If the first (and only) batch comprises multiple arrivals, the scheduler assumes it is the static scenario.

B. Resource Allocation Policy

We design indices to quantitatively assess the performance of our resource allocation policies, representing preferences when selecting hardware. We consider different disaggregation scales, where the SDC is regarded as a special case of DDC, i.e., a blade-scale DDC.

1) Blade-Scale DDC

The policy here is to select a feasible blade with the highest value of a blade index (η_n^{BS}) defined as:

$$\eta_n^{BS} = \varepsilon \cdot \eta_n^{rel} + (1-\varepsilon) \cdot \eta_n^{eff} \ \forall n \in \boldsymbol{N}, \tag{37}$$

where η_n^{rel} and η_n^{eff} are the reliability and efficiency indices associated with blade n, respectively, and $\varepsilon \in [0,1]$ is the weighting coefficient. The reliability index η_n^{rel} is the probability that blade n does not fail during the service time of the request, and the efficiency index $\eta_n^{eff} = \overline{U}_n = \sum_{r\in\boldsymbol{R}} U_n^r / |\boldsymbol{R}|$, where U_n^r is the utilization of resource r in blade n. The efficiency index is set according to the well-known best-fit (BF) bin-packing scheme, which selects a feasible bin with the least remaining capacity [44].

2) Rack- or Pod-Scale DDC

Since the rack- and pod-scale DDCs are similar, we next consider only the rack-scale DDC. The allocation in a rack-scale DDC is to select a rack and then choose a blade for each type of resource, which involves rack and blade indices.

Similar to (37), the blade index is also the weighted sum of efficiency and reliability indices. The efficiency index is the utilization of the blade. Since a DDC blade has multiple components of the same resource type, the utilization of the blade is calculated by $\sum_{c\in\boldsymbol{C}_n} L_{nc} / \sum_{c\in\boldsymbol{C}_n} A_{nc}$, where L_{nc} and A_{nc} are the load and available capacity of component c in blade n, respectively. The reliability index of a blade is $P_n^i \cdot \prod_{c\in\boldsymbol{C}_n^i} P_{nc}^i$, where $\boldsymbol{C}_n^i$ is the set of components in blade n used by request i. Note that the reliability index here only considers the components used by the request because components are independent and do not interfere with each other. Overall, the blade index (η_n^{RS}) is:

$$\eta_n^{RS} = \varepsilon \cdot P_n^i \cdot \prod_{c\in\boldsymbol{C}_n^i} P_{nc}^i + (1-\varepsilon) \cdot \frac{\sum_{c\in\boldsymbol{C}_n} L_{nc}}{\sum_{c\in\boldsymbol{C}_n} A_{nc}}. \tag{38}$$

The rack index is also the weighted sum of a reliability index and an efficiency index. The reliability index of a rack is the request's reliability when allocated with the selected blades in the rack and the required switches and links. The efficiency index is defined as the average utilization of the $|\boldsymbol{R}|$ selected blades in the rack. Finally, the rack (or pod) with the highest rack index is chosen.

C. The Algorithm for One Request in a Rack-Scale DDC

1) Algorithm's Overview

Fig. 4 provides the algorithm pseudocode, named Arrival-Alloc, for allocating resources to a request. It takes two inputs, i.e., the request i and the rack-scale DDC graph G. The algorithm first sorts the resource types in $\boldsymbol{R}$ in descending order of *competitive ratio* [18], defined as the ratio of the requested resource amount to the capacity per component for each resource type $r \in \boldsymbol{R}$. This operation prioritizes intensive resources, e.g., for $r =$ CPU, a CPU-intensive request has the largest competitive ratio. If components have different capacities, the denominator of the competitive ratio is the average component capacity.

```
ARRIVAL-ALLOC ($i, G$)              //$i$ - request, $G$ - rack-scale DDC
 1: SORT-DES($\boldsymbol{R}, \frac{D_{ir}}{capacity.r}$);   // sort $\boldsymbol{R}$ in descending order of $\frac{D_{ir}}{capacity.r}$
 2: $maxRackIndex$ = -1;
 3: $\phi_{best} = Null$;                 // initialize the best solution as empty
 4: for ($k \in \Gamma$)
 5:     for ($r \in \boldsymbol{R}$)
 6:         $L_{kr}^B = \emptyset$;           // storing feasible blades with $r$ in $k$
 7:         for (each blade $n$ with resource $r$ in rack $k$)
 8:             if (FRAGMENTABLE-BIN-PAC($n, D_{ir}$) == TRUE)
 9:                 $L_{kr}^B$.add($n$);              // record feasible blade
10:         if ($L_{kr}^B == \emptyset$)
11:             go to line 4;      // no feasible solution, try the next rack
12:         SORT-DES($L_{kr}^B, \eta_n^{RS}$); // descending order of blade index
13:     $L_k^B = (L_{kr}^B : r \in \boldsymbol{R})$;       // storing these lists in one set
14:     $p = 1$;                   // index the first resource (type) in $\boldsymbol{R}$
15:     $\phi_k = Null$;              // storing solution in rack $k$
16:     if (BLADE-SELEC-TRAFFIC-SCHED($r, \boldsymbol{R}, L_k^B, \phi_k$) == TRUE)
17:         if (RACK-INDEX($\phi_k$) > $maxRackIndex$)
18:             $max$ = RACK-INDEX($\phi_k$);
19:             $\phi_{best} = \phi_k$;                  //update the best solution
20: if ($\phi_{best} \neq Null$)
21:     FINAL-ALLOC($\phi_{best}$);
22:     return TRUE;
23: return FALSE;
```
Fig. 4. Pseudocode of the algorithm for allocating resources to a request.

Subsequently, the algorithm scans the racks to find the best solution, i.e., the rack with the maximum rack index. The variable ϕ_{best} in line 3 is a global variable recording the best solution. This variable has a self-defined data structure that records the information on which components, blades, links, and switches are used for providing the resource to request i. The algorithm updates the current optimal solution when a new solution with a higher rack index is found (lines 16 – 19). The algorithm iterates until all racks are checked, and finally, the one with the highest rack index is selected (line 21).

In a candidate rack and for each resource type, the algorithm filters the blades with sufficient remaining capacity by calling the procedure named FRAGMENTABLE-BIN-PAC (line 8). This procedure addresses the problem of allocating resources from the components in the given blade and outputs TRUE when the blade is feasible to host the request. Afterward, the feasible blades are sorted in descending order of the blade index (line 12) to prioritize those with high blade indices. Next, the algorithm calls the procedure BLADE-SELEC-TRAFFIC-SCHED to find the solution in this rack (line 16), i.e., select blades and schedule the traffic.

2) *Allocating Components Resources in a Given Blade*

This corresponds to the procedure FRAGMENTABLE-BIN-PAC in Fig. 4. This procedure is based on the algorithm proposed by LeCun *et al.* [45] to solve the bin-packing problem where the items are splittable. Here, the item to be packed is the resource demand D_{ir}, and the bins are the components in blade n. This procedure operates as follows. Firstly, sort the components in the given blade in decreasing order of P^i_{nc} (a probability term defined in Table I), prioritizing high reliable components. Secondly, try to find out a *perfect* component [45], i.e., the component whose remaining capacity is precisely equal to D_{ir}. If the perfect component exists (e.g., Request 3 in Fig. 5), assign it to the request and terminate the procedure. Otherwise, check the components one by one. When the remaining capacity of a candidate component is larger than the demand (e.g., Request 1 in Fig. 5), allocate resources from this component. Otherwise, allocate all its remaining capacity to the request and use subsequent components to fulfill the remaining demand (e.g., Request 2 in Fig. 5).

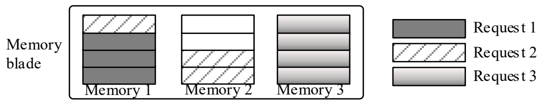


Fig. 5. Three requests on a blade with three memory components, where the three requests arrive in the order of Requests 1, 2, and 3.

3) *Blade Selection and Traffic Scheduling in a Given Rack*

```
BLADE-SELEC-TRAFFIC-SCHED($p, \boldsymbol{R}, L^B_k, \phi_k$)
 1: $r = \boldsymbol{R}[p]$;
 2: $L^B_{kr} = L^B_k.\mathrm{get}(r)$;            // the list of blades of resource $r$
 3: for ($n_{cur} \in L^B_{kr}$)
 4:     if (TRAFFIC-SCHED($\phi_k, n_{cur}$) == FALSE)
 5:         continue;
 6:     $\phi_k.\mathrm{add}(n_{cur})$;            // record the feasible blade
 7:     if ($p == |\boldsymbol{R}|$)               // this is the last resource (type)
 8:         return TRUE;
 9:     else
10:         $p++$;                                 // index the resource (type) next to $r$
11:         res = BLADE-SELEC-TRAFFIC-SCHED($p, \boldsymbol{R}, L^B_k, \phi_k$)
12:         if (res == FALSE)
13:             $\phi_k.\mathrm{remove}(n_{cur})$; // failure in resources after $r$
14:         else
15:             return TRUE
16: return FALSE
```

Fig. 6. Pseudocode of blade selection and traffic scheduling in a rack.

This part corresponds to the procedure BLADE-SELEC-TRAFFIC-SCHED in Fig. 4, and Fig. 6 gives the pseudocode of the procedure. The general idea of this procedure is as follows. In the beginning, we select a blade for the first resource type (r_{1st}). When selecting a blade for the second resource type (r_{2nd}), we also try to schedule the traffic (r_{1st}, r_{2nd}). If the traffic scheduling fails, meaning that this blade is not suitable for r_{2nd}, we try the next blade. Similarly, when selecting a blade for the third resource type (r_{3rd}), we also need to schedule the traffic (r_{1st}, r_{3rd}) and (r_{2nd}, r_{3rd}). This process is repeated for each of the remaining resource types.

The procedure takes the input $p = 1, \ldots, |\boldsymbol{R}|$, to index the current resource type r, i.e., $\boldsymbol{R}[p]$ is the p^{th} element in $\boldsymbol{R}$. Note that input $\boldsymbol{R}$ is an ordered set that has been sorted previously (see line 12 in Fig. 4). The procedure is executed recursively, starting from the first resource type, i.e., $p = 1$. Each time a resource type temporarily determines its host, the procedure goes to the next resource type by incrementing p (line 10). The termination condition is that the procedure finds a feasible blade for the last resource type (lines 7-8).

For a current resource type r indexed by p, the procedure scans blades in L^B_{kr} to look for the first blad that satisfies the traffic requirement (line 4). Here, another procedure named TRAFFIC-SCHED is called to schedule the traffic between the current resource type r and the previous resource types $r_{pri} = \boldsymbol{R}[p_{pri}], p_{pri} = 1, \ldots, p-1$. If the traffic scheduling succeeds, the procedure temporarily records the current blade n_{cur} as the host for r (line 6), and go for the next resource type $r_{next} = \boldsymbol{R}[p+1]$ (line 10). Then, the procedure recursively calls the algorithm itself (line 11) to find a feasible blade for r_{next}. If the next resource type cannot find a feasible blade satisfying the traffic requirement, it abandons blade n_{cur} (line 13), then tries the next blade for resource type r.

The traffic scheduling procedure TRAFFIC-SCHED searches for a path for each resource pair, which is operated as follows. Firstly, it excludes links with insufficient capacity. Then, it runs the shortest path algorithm with the weight of each link setting as $w_{mn} = \tau^{pro}_{mn} + 0.5 \cdot \tau_m + 0.5 \cdot \tau_n$, where, τ^{pro}_{mn}, τ_m, and τ_n are the propagation delay of link (m, n) and the delays at the endpoints m and n, respectively. If the path exists, it further checks whether the latency meets the requirement and schedules the traffic along this path if it does.

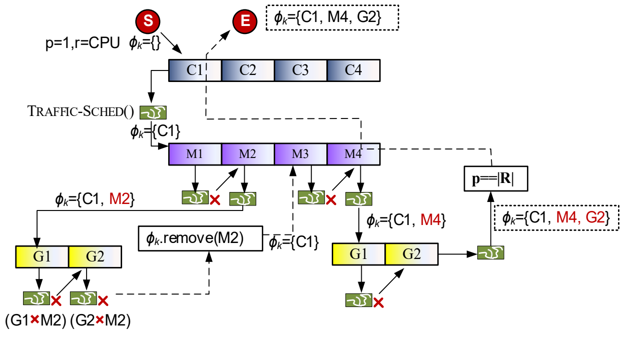


Fig. 7. Example of the blade selection procedure.

Fig. 7 illustrates the procedure by an example, which includes three resource types, sorted in the order of CPU, memory, and GPU. It starts from the first resource type, i.e., CPU, and checks whether the first CPU blade (C1) is feasible. As it is the first resource type, there are no previous resource types, so no traffic needs to be scheduled, and C1 is directly recorded as the candidate blade (recorded in ϕ_k). Then, it goes for the second resource type, i.e., memory, and needs to schedule the traffic between memory and CPU. However, selecting M1 as the candidate leads to failure in traffic scheduling, so M1 is not feasible, and M2 is recorded as a temporary host for the memory. When it comes to the last resource type, GPU, the procedure can not find a feasible blade when scheduling the traffic between memory and GPU. It returns to the memory blade selection step and abandons M2 but tries M3 and M4. Finally, C1, M4, and G2 are selected, and traffic scheduling is also complete.

4) Algorithm's Complexity

We first analyze the algorithm's time complexity of the algorithm provided in Fig. 4. We consider it takes $O(n\log n)$ to sort a list with n elements, as many sorting algorithms have this complexity. In addition, we assume the number of resource types and the number of components in each blade are constant. Accordingly, lines 4, 8, and 12 take the time of $O(1)$, $O(|\Gamma| \cdot |N_{kr}|)$, and $O(|\Gamma| \cdot |N_{kr}| \cdot \log|N_{kr}|)$, respectively. Here $|N_{kr}|$ is the number of blades of each resource type per rack. The most time-consuming execution is the blade selection procedure, which is to select one blade for each resource type from a rack. Since the total number of blade combinations is $|N_{kr}|^{|\boldsymbol{R}|}$, the complexity of the blade selection procedure is $O\left(|N_{kr}|^{|\boldsymbol{R}|}\right)$. Thus, line 16 takes the time of $O\left(|\Gamma| \cdot |N_{kr}|^{|\boldsymbol{R}|}\right)$. In total, the time complexity of the algorithm is $O\left(|\Gamma| \cdot |N_{kr}| \cdot \log|N_{kr}| + |\Gamma| \cdot |N_{kr}|^{|\boldsymbol{R}|}\right)$. It is equal to $O\left(|\Gamma| \cdot |N_{kr}|^{|\boldsymbol{R}|}\right)$ since $|\boldsymbol{R}| > 1$. Since $|\boldsymbol{R}|$ has a typical value of 3 to 4, the complexity is very high. Fortunately, the DDC architectures are basically rack-scale, where the base term $|N_{kr}|$ does not have a high value, at most a dozen.

The space complexity of this algorithm is $O(|\boldsymbol{R}|)$. The extra space is needed mainly to recursively call the BLADE-SELEC-TRAFFIC-SCHED procedure.

D. Batch Allocation for the Static Scenario

For this scenario, we introduce the SHUFFLE-AND-UPDATE procedure as follows. Given a set of requests $\boldsymbol{I}$, we first randomize the order of requests in $\boldsymbol{I}$, then apply the algorithm shown in Fig. 4 to each request according to the order and calculate the objective function, recorded as F_m. Nextly, we set an integer number which we refer to as an *epoch size*. We then repeat the randomize-and-try to find a better solution, i.e., a solution with the objective F_1, and $F_1 > F_m$. If a better solution can be found after repeating the random-and-try K times, such that $K <$ *epoch size*, we update the optimal solution with F_1. We call this process *one epoch of the SHUFFLE-AND-UPDATE optimization process*. The above optimization process is further repeated until no better solution is found in any epoch.

E. Migration for the Dynamic Scenario

In order to reduce the number of accepted requests failing to complete their service, we propose a migration mechanism to restore the interrupted requests. Fig. 8 gives the pseudocode of this procedure. The procedure first excludes the failed element from the DDC graph, then, for each interrupted request, calls the procedure ARRIVAL-ALLOC (See Fig. 4) based on the residual graph and re-allocates resources to this request if it succeeds. It should be noted that the failed element can be repaired and can serve other requests after being repaired.

```
MIGRATION-BASED-RESTORATION($e, G$)    // $e$ – the failed element
1: $\boldsymbol{I^{intr}}$ = HOSTED-REQUESTS($e$);   // get the set of interrupted requests
2: $G$.remove($e$);                      // remove failed element from $G$
3: failNum = 0;                  // number of failing restored requests
4: for ($i \in \boldsymbol{I^{intr}}$)
5:     success = ARRIVAL-ALLOC($i, G$)
6:     if (success == FALSE)              // fail migration
7:         failNum ++;                    // fail to complete service
8: return failNum;
```
Fig. 8. Pseudocode of the migration-based restoration when e fails.

F. Discussion on Implementation

The implementation of our proposed resource management framework needs the support of operating systems (OSes) for DDCs, e.g., LegoOS [12] and GiantVM [46] LegoOS is based on a new OS model, named splitkernel, where each resource component maintains an independent manager (or monitor). LegoOS provides the capability that allows an application to use multiple components of the same resource type, which can be combined with our proposed components allocation procedure. In addition, LegoOS provides fault-tolerance mechanisms, which can be combined with the migration-based restoration algorithm to resume interrupted services. GiantVM is a hypervisor system that can create a VM spanning multiple components and machines (i.e., blades), which may also be used to implement our proposed framework.

VII. EVALUATION

This section presents the numerical results for validating the performance of the DDC against the SDC. Both the static scenario and dynamic scenario are considered.

A. Static Scenario

1) Parameter settings

The MILP can only find optimal solutions for small-size problems due to its high computational complexity. We consider only one pod consisting of one spine switch and 3 racks with one leaf switch, 3 blades per rack, and 3 components per blade. Three types of resources are considered: $\boldsymbol{R} = (CPU, memory, accelerator)$, with a per component capacity of 100 units. The age of each hardware element is U(10^2,10^4) time units, where U(*a*, *b*) is a random value uniformly distributed within the interval (*a*, *b*). Henceforth, we do not add units to simulation time measures, which are understood to be time units. All the requests are assumed to be given at time $t = 0$, and the service time of each request is U(1,100). Four kinds of requests are considered: CPU-intensive, memory-intensive, accelerator-intensive, and random. If request i is of r_1-intensive ($r_1 \in \boldsymbol{R}$), i.e., the first three kinds of requests, its demand is set as

$$D_{ir} = \begin{cases} U(40{,}80), r = r_1 \\ U(10{,}40), r \neq r_1 \end{cases}.$$

For a random request, its demand is $D_{ir} = U(10{,}80)\ \forall r \in \boldsymbol{R}$.

We consider four test cases considering the settings for reliability and network latency/bandwidth, namely, S1 – S4. In S1, all hardware elements except the resource components are resilient, and we assume that requests have no latency and bandwidth requirements. Each component's reliability follows the Weibull distribution with scale parameter $\eta_e = 10^6$ and shape parameter $\beta_e = U(1,2)$. This setting ensures that the hardware mean TBF (MTBF) is far longer than the request service time by four to five orders of magnitude. S2 is extended from S1, where blades become not resilient, whose reliability also follows the above Weibull distribution. Similarly, S3 is extended from S2, where each switch's reliability also follows the above Weibull distribution. Lastly, S4 further considers the latency constraints based on S3. With reference to the literature (e.g., [18], [40]), we set the latency-related parameters as follows. Hardware delays are set as: τ_n^{bld} = U(50,60) ns, τ_n^{sw}= U(100,150) ns, $\tau_{mn}^{pro} = d \times L_{pro}$, where L_{pro} = 5 ns/m and d = 2m for each intra-rack link while U(10, 30) m for each inter-rack link. The latency requirement for the traffic between CPU and memory is U(500, 900) ns; U(1, 100) µs otherwise.

We use the commercial solver AMPL/Gurobi to solve the MILP formulations to provide solutions with respect to the objective function (3). The simulation environment for the heuristic algorithms is built through Java.

2) *Results*

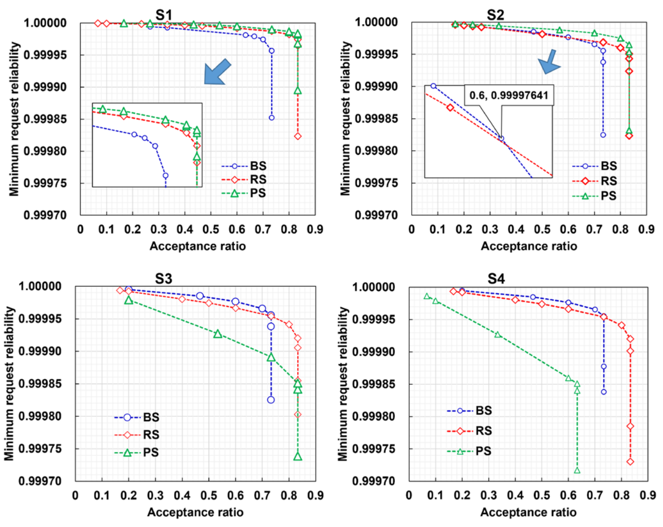


Fig. 9. Pareto fronts in terms of minimum request reliability vs. acceptance ratio for four different cases (S1 – S4) (obtained by MILP).

We first evaluate the performance of different disaggregation scales using the MILP. Fig. 9 presents the results of the minimum requests reliability vs. the acceptance ratio in S1 – S4. Here, the number of requests is 30. In the figure, “BS”, “RS”, and “PS” correspond to the blade-scale, rack-scale, and pod-scale DDCs, respectively. We use dashed lines to connect the results to approximate the Pareto fronts [47]. Each approximate Pareto front provides a set of optimal solutions in terms of acceptance ratio and minimum request reliability based on given preferences. It is obtained by solving the MILP with varied weight factors [47]. The ordinate intervals in S1 – S4 are the same in Fig. 9 so that the differences between the Pareto fronts in S1-S4 can be clearly observed. We observe that PS performs the best in S1 and S2 while performing significantly worse in S3 and S4. This can be explained by the fact that PS has a larger disaggregation scale than RS and BS, so it can achieve higher minimum request reliability and acceptance ratio. However, a request in PS uses more network elements than in BS and RS. Therefore, when the switches are not resilient and latency requirements are considered, the performance of PS sharply decreases.

We also observe that RS performs better than BS in S1, while in S2 – S4, the performance of RS drops faster than BS. The reason is as follows. RS has higher flexibility and efficiency than BS, so RS outperforms BS in S1. However, service reliability in RS is significantly influenced by the network, so the performance of RS decreases sharply when blades and switches are not resilient (S2 – S3). In S4, the approximate Pareto fronts of RS and BS cross at (0.999955, 0.73). Accordingly, if setting a reliability threshold higher than 0.999955 (corresponding to the left side of the intersection), the acceptance ratio of RS is lower than BS while higher than BS otherwise. This is because requests in RS require more switches and blades than in BS, so their achievable reliability in RS may not be as high as in BS, and more requests in RS will be rejected if the reliability threshold is very high. On the contrary, when relaxing the threshold, the benefits of resource disaggregation manifest, where resource stranding in BS is significantly avoided in RS so that RS can achieve a higher acceptance ratio than BS. Overall, the performance difference of RS and BS on both sides of the intersection again demonstrates that resource disaggregation can improve resource efficiency, while the benefit will be offset if the network is not resilient.

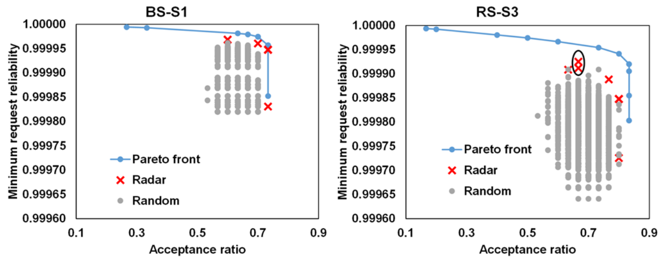


Fig. 10. Minimum request reliability vs. acceptance ratio obtained by Radar in S1 for BS and S3 for RS.

Fig. 10 provides the results of Radar in S1 for BS and S3 for RS, and for comparison, we also show the Pareto fronts and other 10000 random solutions. The results of Radar are obtained with varied weighting coefficients in the blade and rack indices (See Section IV.B). We can observe that Radar performs close to the approximate Pareto fronts and outperforms all random solutions, demonstrating that Radar has high efficiency. In addition, Radar consistently maintains a high acceptance ratio. For example, in RS-S3, the acceptance ratio obtained by Radar is consistently larger than 0.8, also demonstrating its high efficiency. Note that in Fig. 10, we aim to show all the results obtained by Radar and Random, not only the optimal solutions. There are some solutions with the same acceptance ratio but different minimum request reliability obtained by Radar and Random, e.g., the two data points that have been circled in RS-S3. Clearly, having such two solutions, we will not choose the inferior one (the lower of the two). We only aim to illustrate such solutions and explain how they are obtained. This is explained through the 12th line of the algorithm in Fig. 4, which is the operation that sorts the blades in descending order of the blade index. Increasing the weighting coefficients of the blade index prioritizes highly reliable blades. However, the acceptance ratio may remain unchanged since the capacity does not change.

The above results demonstrate that an imperfect network will offset the benefits of hardware disaggregation. Backup is an intuitive way to improve reliability performance, and Fig. 11 gives the results for the case extended from S3, where every blade and switch has a backup. We observe from Fig. 11 that the results are very close to those in S1. This is reasonable as backup dramatically increases the reliability of switches and blades, making the performance very close to when the switches and blades are resilient. However, backup is not an efficient approach, and it leads to increased costs. Next, the results in the dynamic scenario show that improved performance is achievable without backup.

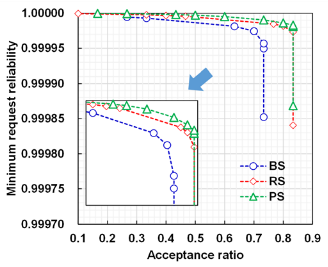


Fig. 11. Pareto fronts for a new case extended from S3 by applying backup to every hardware element.

B. Dynamic Scenario

1) Parameter settings

We consider one pod consisting of 10 spine switches and 9 racks with 2 leaf switches, 48 blades per rack, and 3 components per blade. Other settings, like the hardware capacity and request resource demand, are the same as in the static scenario. Here we do not consider the latency and bandwidth restrictions. We simulate a dynamic system, including the events of request arrival, request departure, hardware (components, blades, and switches) failure, and hardware repair. The request arrival follows a Poisson process with an arrival rate of 1.5, and service time follows an exponential distribution with a mean of 1000. The TBF of each element follows the Weibull distribution with parameters the same as in the static scenario. The time to repair follows another exponential distribution with a mean of 1. With this setting, hardware availability ranges from 99.999% to 99.9999%, which is consistent with hardware availability in real-world DCs [48]. In the simulation, we assume that the interrupted requests are immediately restored after the scheduler decides where to migrate the service. The simulation time is 10^6.

2) Results

Fig. 12 gives the simulation results with varied weight coefficients. The first two subfigures give the results when migration is not applied. We observe that the blocking probability of PS and RS is significantly lower than that of BS. This is because of the significant efficiency benefit of hardware disaggregation. However, the second subfigure shows that, in terms of the number of accepted requests failing to complete service, PS has the highest value, second by RS and last by BS. The results further demonstrate that a vulnerable network may offset the benefits of disaggregation. The last two subfigures give the results when the migration is applied. We observe that PS and RS outperform BS in both blocking probability and the number of accepted requests failing to complete service. The results demonstrate that our proposed Radar framework can overcome the challenges created when the network is not resilient. Notice that in the static scenario, we have demonstrated that the performance of PS is significantly affected by latency. Therefore, only the RS can overcome the challenges caused by an imperfect network when considering latency.

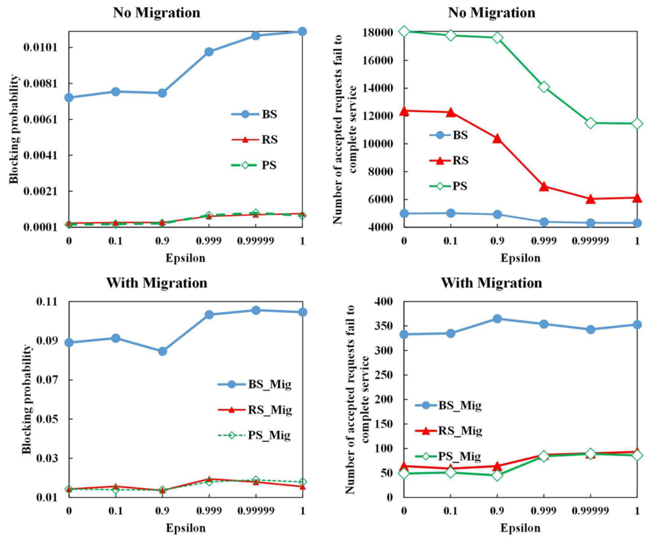


Fig. 12. Blocking probability and the number of requests failing to complete service with varying coefficients.

VIII. Conclusion and Future Research Directions

We have studied service reliability in DDCs, considering network impact and different disaggregation scales. A MILP formulation and the Radar framework have been proposed for the resource allocation in DDCs, considering both reliability and utilization aspects. We have demonstrated that an imperfect network significantly offsets the improvement of resource efficiency and service reliability brought by hardware disaggregation. The results also demonstrate that hardware backup helps overcome such offset, which, however, may lead to increased cost. The proposed Radar framework employs a migration-based restoration, which can also overcome the offset without backup. Numerical results have shown that the rack-scale architecture is currently the best option, as a larger disaggregation scale faces latency challenges, which may lead to severe performance deterioration.

DDC is a relatively new research area that gives rise to many research opportunities. In the following, we consider the limitations of this work that lead to potential extensions and future research directions. Firstly, we have assumed that the demand required by a request is always additive from multiple resource components. This may not be the case in certain applications, e.g., a task/process should be executed on a single CPU component [12]. Secondly, the service request we have considered is like a VM or a task, which requires multiple different resources to form one VM or complete one task. However, there are some applications, such as parallel computing, where multiple VMs are required to jointly complete multiple tasks for a single job. Such applications have not been considered in this paper. Thirdly, as discussed, we have assumed no waiting room, and blocked requests are rejected without a re-allocation attempt. Finally, in response to "the trends of AI for cloud, fog, edge, serverless, and quantum computing" to improve automation in DCs [49], we can study how to integrate AI tools like reinforcement learning with resource allocation in DDCs.

References

[1] D. Mourtzis, *Design and operation of production networks for mass personalization in the era of cloud technology*. Elsevier, 2022.

[2] F. Psarommatis, P. A. Dreyfus, and D. Kiritsis, "The role of big data analytics in the context of modeling design and operation of manufacturing systems," in *Design and operation of production networks for mass personalization in the era of cloud technology*: Elsevier, 2022, pp. 243-275.

[3] X. Sun, N. Ansari, and R. Wang, "Optimizing resource utilization of a data center," *IEEE Communications Surveys & Tutorials,* vol. 18, no. 4, pp. 2822-2846, Fourthquarter 2016.

[4] L. Yin, J. Luo, and H. Luo, "Tasks scheduling and resource allocation in fog computing based on containers for smart manufacturing," *IEEE Transactions on Industrial Informatics,* vol. 14, no. 10, pp. 4712-4721, Oct. 2018.

[5] D. Fernández-Cerero, J. A. Troyano, A. Jakóbik, and A. Fernández-Montes, "Machine learning regression to boost scheduling performance in hyper-scale cloud-computing data centres," *Journal of King Saud University-Computer and Information Sciences,* vol. 34, no. 6, pp. 3191-3203, Jun. 2022.

[6] W. Zhang, R. Yadav, Y.-C. Tian, S. K. K. S. Tyagi, I. A. Eelgendy, and O. Kaiwartya, "Two-phase industrial manufacturing service management for energy efficiency of data centers," *IEEE Transactions on Industrial Informatics,* 2022.

[7] K. Kaur, S. Garg, G. Kaddoum, E. Bou-Harb, and K.-K. R. Choo, "A big data-enabled consolidated framework for energy efficient software defined data centers in iot setups," *IEEE Transactions on Industrial Informatics,* vol. 16, no. 4, pp. 2687-2697, Apr. 2020.

[8] N. Kumar, G. S. Aujla, S. Garg, K. Kaur, R. Ranjan, and S. K. Garg, "Renewable energy-based multi-indexed job classification and container management scheme for sustainability of cloud data centers," *IEEE Transactions on Industrial Informatics,* vol. 15, no. 5, pp. 2947-2957, May 2019.

[9] N. Gholipour, E. Arianyan, and R. Buyya, "A novel energy-aware resource management technique using joint VM and container consolidation approach for green computing in cloud data centers," *Simulation Modelling Practice and Theory,* vol. 104, p. 102127, Nov. 2020.

[10] Q. Fang, J. Wang, Q. Gong, and M. Song, "Thermal-aware energy management of an HPC data center via two-time-scale control," *IEEE Transactions on Industrial Informatics,* vol. 13, no. 5, pp. 2260-2269, Oct. 2017.

[11] C. Guo, K. Xu, G. Shen, and M. Zukerman, "Temperature-aware virtual data center embedding to avoid hot spots in data centers," *IEEE Transactions on Green Communications and Networking,* vol. 5, no. 1, pp. 497-511, 2020.

[12] Y. Shan, Y. Huang, Y. Chen, and Y. Zhang, "LegoOS: A disseminated, distributed {OS} for hardware resource disaggregation," in *Proc. Symposium on Operating Systems Design and Implementation*, 2018, pp. 69-87.

[13] T. Harvey, "Hype cycle for compute infrastructure, 2021," Gartner, 2021.

[14] A. Moskovsky and P. Lavrenko, "Composable disaggregated environments for HPC workloads," in *Proc. CEUR Workshop*, 2020, pp. 43-51.

[15] A. S. Al-Harrasi and S. Ali, "Investigating the challenges facing composable/disaggregated infrastructure implementation: A literature review," in *Proc. International Arab Conference on Information Technology*, 2021, pp. 1-5.

[16] ResearchAndMarkets.com, "The worldwide composable infrastructure industry is expected to grow at a CAGR of 21% between 2021 to 2027." [Online]. Available: https://www.globenewswire.com/news-release/2021/10/13/2313556/28124/en/The-Worldwide-Composable-Infrastructure-Industry-is-Expected-to-Grow-at-a-CAGR-of-21-Between-2021-to-2027.html. [Accessed: 6-Apr-2022].

[17] T. Coughlin, "Digital storage and memory," *Computer, IEEE,* vol. 55, no. 1, pp. 20-29, Jan. 2022.

[18] G. Zervas, H. Yuan, A. Saljoghei, Q. Chen, and V. Mishra, "Optically disaggregated data centers with minimal remote memory latency: Technologies, architectures, and resource allocation," *Journal of Optical Communications and Networking,* vol. 10, no. 2, pp. A270-A285, Feb. 2018.

[19] X. Guo, X. Xue, F. Yan, B. Pan, G. Exarchakos, and N. Calabretta, "DACON: A reconfigurable application-centric optical network for disaggregated data center infrastructures," *Journal of Optical Communications and Networking,* vol. 14, no. 1, pp. A69-A80, Jan. 2022.

[20] A. Pagès, F. Agraz, and S. Spadaro, "On the impact of IT resources disaggregation in optically interconnected data centres," in *Proc. ECOC*, 2019, pp. 1-4.

[21] O. O. Ajibola, T. E. El-Gorashi, and J. M. Elmirghani, "Network topologies for composable data centers," *IEEE Access,* vol. 9, pp. 120955-120984, Sep. 2021.

[22] Intel.com, "Intel® rack scale design (Intel® RSD) architecture specification (software v2.5)." [Online]. Available: https://www.intel.com/content/www/us/en/architecture-and-technology/rack-scale-design/architecture-spec-v2-5.html. [Accessed: 29-Jul-2021].

[23] A. Pagès, R. Serrano, J. Perelló, and S. Spadaro, "On the benefits of resource disaggregation for virtual data centre provisioning in optical data centres," *Computer Communications,* vol. 107, pp. 60-74, Jul. 2017.

[24] X. Guo, F. Yan, G. Exarchakos, X. Xue, B. Pan, and N. Calabretta, "On the workload deployment, resource utilization and operational cost of fast optical switch based rack-scale disaggregated data center network," in *Proc. OFC*, 2020, pp. 1-3.

[25] A. D. Papaioannou, R. Nejabati, and D. Simeonidou, "The benefits of a disaggregated data centre: A resource allocation approach," in *Proc. GLOBECOM*, 2016, pp. 1-7.

[26] C. Zhang, P. Zhang, S. Zheng, Z. Yang, R. Liu, and K. Huang, "An efficient self-healing architecture for improving the RAS characteristics of RISC-V server and its quantitative evaluation method," *IEEE Embedded Systems Letters,* to appear, 2022.

[27] C. Guo, X. Wang, G. Shen, S. Bose, J. Xu, and M. Zukerman, "Exploring the benefits of resource disaggregation for service reliability in data centers," *IEEE Transactions on Cloud Computing,* (to appear).

[28] A. Carbonari and I. Beschasnikh, "Tolerating faults in disaggregated datacenters," in *Proc. HotNets Workshop*, 2017, pp. 164-170.

[29] P. X. Gao *et al.*, "Network requirements for resource disaggregation," in *Proc. Symposium on Operating Systems Design and Implementation*, 2016, pp. 249-264.

[30] H. M. M. Ali, T. E. El-Gorashi, A. Q. Lawey, and J. M. Elmirghani, "Future energy efficient data centers with disaggregated servers," *Journal of Lightwave Technology,* vol. 35, no. 24, pp. 5361-5380, Dec. 2017.

[31] O. O. Ajibola, T. E. El-Gorashi, and J. M. Elmirghani, "Energy efficient placement of workloads in composable data center networks," *Journal of Lightwave Technology,* vol. 39, no. 10, pp. 3037-3063, May 2021.

[32] R. Lin, Y. Cheng, M. De Andrade, L. Wosinska, and J. Chen, "Disaggregated data centers: Challenges and trade-offs," *IEEE Communications Magazine,* vol. 58, no. 2, pp. 20-26, 2020.

[33] M. Amaral *et al.*, "DRMaestro: Orchestrating disaggregated resources on virtualized data-centers," *Journal of Cloud Computing,* vol. 10, no. 1, pp. 1-20, Mar. 2021.

[34] S. Angel, M. Nanavati, and S. Sen, "Disaggregation and the application," in *Proc. HotCloud*, 2020.

[35] L. Ferreira *et al.*, "Optimizing resource availability in composable data center infrastructures," in *Proc. Latin-American Symposium on Dependable Computing*, 2019, pp. 1-10.

[36] M. Alizadeh and T. Edsall, "On the data path performance of leaf-spine datacenter fabrics," in *Proc. IEEE Annual Symposium on High-Performance Interconnects*, 2013, pp. 71-74.

[37] B. Cao *et al.*, "Multiobjective 3-D topology optimization of next-generation wireless data center network," *IEEE Transactions on Industrial Informatics,* vol. 16, no. 5, pp. 3597-3605, May 2019.

[38] C.-S. Li, H. Franke, C. Parris, B. Abali, M. Kesavan, and V. Chang, "Composable architecture for rack scale big data computing," *Future Generation Computer Systems,* vol. 67, pp. 180-193, Feb. 2017.

[39] V. Shrivastav *et al.*, "Shoal: A network architecture for disaggregated racks," in *Proc. Symposium on Networked Systems Design and Implementation*, 2019, pp. 255-270.

[40] V. Mishra, J. L. Benjamin, and G. Zervas, "MONet: Heterogeneous memory over optical network for large-scale data center resource disaggregation," *Journal of Optical Communications and Networking,* vol. 13, no. 5, pp. 126-139, May 2021.

[41] Z. Ding, Y.-C. Tian, M. Tang, Y. Li, Y.-G. Wang, and C. Zhou, "Profile-guided three-phase virtual resource management for energy efficiency of

data centers," *IEEE Transactions on Industrial Electronics,* vol. 67, no. 3, pp. 2460-2468, Mar. 2019.

[42] D. Wang, W. Zhang, H. He, and Y.-C. Tian, "Efficient hybrid central processing unit/input–output resource scheduling for virtual machines," *IEEE Transactions on Industrial Electronics,* vol. 68, no. 3, pp. 2714-2724, Mar. 2021.

[43] L. Liu, Y. Ding, X. Li, H. Wu, and L. Xing, "A container-driven service architecture to minimize the upgrading requirements of user-side smart meters in distribution grids," *IEEE Transactions on Industrial Informatics,* vol. 18, no. 1, pp. 719-728, Jan. 2021.

[44] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham, "Worst-case performance bounds for simple one-dimensional packing algorithms," *SIAM Journal on Computing,* vol. 3, no. 4, pp. 299-325, Dec. 1974.

[45] B. LeCun, T. Mautor, F. Quessette, and M.-A. Weisser, "Bin packing with fragmentable items: Presentation and approximations," *Theoretical Computer Science,* vol. 602, pp. 50-59, Oct. 2015.

[46] X. Jia, J. Zhang, B. Yu, X. Qian, Z. Qi, and H. Guan, "GiantVM: A novel distributed hypervisor for resource aggregation with DSM-aware optimizations," *ACM Transactions on Architecture and Code Optimization (TACO),* vol. 19, no. 2, pp. 1-27, Jun. 2022.

[47] S. Smale, "Global analysis and economics I: Pareto optimum and a generalization of morse theory," in *Dynamical systems*: Elsevier, 1973, pp. 531-544.

[48] PaloAltoNetworks.com, "What is a data center?" [Online]. Available: https://www.paloaltonetworks.com/cyberpedia/what-is-a-data-center. [Accessed: 6-Aug-2022].

[49] S. S. Gill *et al.*, "AI for next generation computing: Emerging trends and future directions," *Internet of Things,* vol. 19, p. 100514, Aug. 2022.

Chao Guo received his Master degree in information and communication engineering from Soochow University, Suzhou, China. He is currently working toward a Ph.D. degree with the Department of Electrical Engineering, City University of Hong Kong, Hong Kong. His research interests include the areas of data center resouce scheduling and network optimization, network slicing, and cable path planning.

Moshe Zukerman (M'87–SM'91–F'07–LF'20) received the B.Sc. degree in industrial engineering and management, the M.Sc. degree in operations research from the Technion – Israel Institute of Technology, Haifa, Israel, and the Ph.D. degree in engineering from University of California, Los Angeles, in 1985. He was an independent consultant with the IRI Corporation and a Postdoctoral Fellow with the University of California, Los Angeles, in 1985–1986. In 1986–1997, he was with Telstra Research Laboratories (TRL), first as a Research Engineer and, in 1988–1997, as a Project Leader. He also taught and supervised graduate students at Monash University in 1990–2001. During 1997-2008, he was with The University of Melbourne, Victoria, Australia. In 2008 he joined City University of Hong Kong as a Chair Professor of Information Engineering, and a team leader. He has over 300 publications in scientific journals and conference proceedings. He has served on various editorial boards such as Computer Networks, IEEE Communications Magazine, IEEE Journal of Selected Areas in Communications, IEEE/ACM Transactions on Networking and Computer Communications.

Tianjiao Wang received a B.Sc. degree in electronic and communication engineering from Qufu Normal University, Shandong, China, and an M.Sc. degree in electronic and communication engineering from Beihang University, Beijing, China, in 2015 and 2018, respectively. She is currently working toward a Ph.D. degree with the Department of Electrical Engineering, City University of Hong Kong, Hong Kong. Her research interests include the areas of popularity prediction and cable network topology optimization.