Dynamic DPU Offloading and Computational Resource Management in Heterogeneous Systems

Zhaoyang Huang, Yanjie Tan, Yifu Zhu, Huailiang Tan, and Keqin Li, *Fellow, IEEE*

***Abstract*—DPU offloading has emerged as a promising way to enhance data processing efficiency and free up host CPU resources. However, unsuitable offloading may overwhelm the hardware and hurt overall system performance. It is still unclear how to make full use of the shared hardware resources and select optimal execution units for each tenant application. In this paper, we propose DORM, a dynamic DPU offloading and resource management architecture for multi-tenant cloud environments with CPU-DPU heterogeneous platforms. The primary goal of DORM is to minimize host resource consumption and maximize request processing efficiency. By establishing a joint optimization model for offloading decision and resource allocation, we abstract the problem into a mixed integer programming mathematical model. To simplify the complexity of model-solving, we decompose the model into two subproblems: a 0-1 integer programming model for offloading decision-making and a convex optimization problem for fine-grained resource allocation. Besides, DORM presents an orchestrator agent to detect load changes and dynamically adjust the scheduling strategy. Experimental results demonstrate that DORM significantly improves resource efficiency, reducing host CPU core usage by up to 83.3%, increasing per-core throughput by up to 4.61x, and lowering the latency by up to 58.5% compared to baseline systems.**

***Index Terms*—DPU, heterogeneous system, multi-tenant, optimal scheduling.**

I. Introduction

TRADITIONAL data center relies on the central processing unit (CPU) for data transmission and processing. However, with the rapid development of storage and network technology, the data scale and complexity in data centers are experiencing exponential growth, presenting new challenges for CPUs [1], [2]. In response to emerging demands, cloud providers increasingly adopted data processing units (DPUs) for multi-tenant deployment and request processing [3]. DPU offloading lowers the Total Cost of Ownership (TCO) and enhances the processing efficiency of complex computing systems.

Researchers have explored the potential of DPU acceleration from three aspects, i.e., network [4], [5], [6], [7], storage [8], [9], [10], [11], and computation [12], [13], [14], [15]. Table I lists the details of related works, including the offloading strategy, target tasks, and design goals. The majority of existing research utilizes static offloading strategies, concentrating on offloading various applications or functions into DPU and making post-offloading optimization. However, simply offloading all tenant applications can potentially overwhelm the hardware, resulting in resource saturation [16]. As illustrated in Section II-B, full offloading fails to attain optimal performance and may instead result in increased processing latency. Therefore, appropriate scheduling schemes are essential to maintain the load balance across processors.

Furthermore, the multi-tenant environment introduces a heightened level of complexity to system design. Cooperative applications share system resources, and the offloading performance will be influenced by other co-located tasks and system load conditions. FairNIC [14] first explores the potential of leveraging DPU in cloud environments and provides fair and isolated performance for each tenant. LogNIC [15] introduces an optimizer aimed at guiding fine-grained computation placement while partitioning computing resources into multiple virtual instances to address resource contention issues. However, they do not consider tenant deployment for CPU-DPU heterogeneous systems, which poses a crucial question: *how to make optimal offloading decisions and resource allocation for each tenant application, in order to maximize the heterogeneous resource utilization and offloading benefits?*

Several dynamic offloading methods have been proposed in recent years. UNO [4] dynamically places Network Functions (NFs) between the x86 host and DPU based on the current resource and PCIe bandwidth utilization. It aims at offloading as much workload to DPU as possible and ignores the processing efficiency which may lead to higher latency. E3 [12] places microservices among multiple DPUs considering the network topology. iPipe [13] proposes an actor-based scheduler and migrates actors according to processing loads. However, all these approaches make task placement decisions based on system conditions, overlooking key workload-specific characteristics. The lack of consideration for factors such as task complexity, request size, and resource dependencies can result in suboptimal application deployments, which may significantly undermine system performance and compromise overall efficiency. For instance, offloading I/O-intensive tasks might yield higher latency due to frequent occupation of the PCIe interface for memory I/O [17].

Received 4 November 2024; revised 8 May 2025; accepted 26 June 2025. Date of publication 30 June 2025; date of current version 11 August 2025. Recommended for acceptance by M. Kandemir. *(Corresponding author: Huailiang Tan.)*

Zhaoyang Huang, Yanjie Tan, Yifu Zhu, and Huailiang Tan are with the College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China (e-mail: huangzhaoyang@hnu.edu.cn; tanhuailiang@hnu.edu.cn).

Keqin Li is with the College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China, and also with the Department of Computer Science, State University of New York, New Paltz, NY12561 USA (e-mail: lik@newpaltz.edu).

Digital Object Identifier 10.1109/TC.2025.3584501

TABLE I
RELATED WORK

Acceleration	Method	DO	DRA	MT	Target Task	Design Goal
Network	UNO [4] (SOCC-17)	✓	×	×	Network functions	Unify host and DPU offload for flexible packet processing.
	AccelNet [5] (NSDI-18)	×	×	×	Host networking	Azure accelerated networking and fast in-network packet processing.
	AccelTCP [6] (NSDI-20)	×	×	×	Stateful TCP operations	Simplify host TCP stack and accelerating network applications.
	FlexTOE [7] (NSDI-22)	×	×	×	All TCP data-path	Flexible TCP offload with fine-grained parallelism and eliminate host data-path TCP.
Storage	LeapIO [8] (ASPLOS-20)	×	×	✓	Storage service	Cloud storage stack leveraging ARM-based co-processors to offload complex storage services.
	LineFS [9] (SOSP-21)	×	×	×	Distributed file systems	High-performance DPU offload of a distributed file system with pipeline parallelism.
	Xenic [10] (SOSP-21)	×	×	×	Distributed transaction	Minimize communication cost and transaction commit latency.
	Gimbal [11] (SIGCOMM-21)	×	×	✓	Disaggregated NVMe storage	Enabling multi-tenant storage disaggregation on SmartNIC JBOFs.
Computation	E3 [12] (ATC-19)	✓	×	×	Microservices	Energy-efficient microservices on DPU-accelerated servers.
	iPipe [13] (SIGCOMM-19)	✓	✓	×	Distributed applications	Maximize offloading benefits for distributed applications.
	FairNIC [14] (SIGCOMM-20)	×	✓	✓	OVS and KVS	Enabling SoC DPU use in cloud environments and provide performance isolation.
	LogNIC [15] (MICRO-23)	×	✓	✓	Offloaded programs	Analyzing the performance characteristics of a offloaded program under a given traffic profile.
	Ours	✓	✓	✓	**Generic data center tasks**	**Offloading decision and resource allocation.**

* Note that DO is short for dynamic offloading strategy, DRA refers to DPU internal resource allocation, and MT denotes multi-tenancy.

Due to the limited computation resources on DPU, it is hard for all tenant applications to be deployed on it. Therefore, the implementation of suitable scheduling algorithms is essential to optimize placement decisions and ensure the fair allocation of computation resources [13]. In this paper, we introduce an innovative dynamic offloading architecture, called DORM, to offload generic data center tasks onto DPU and alleviate cloud providers from the substantial burden of data center operations. Our focus lies within the multi-tenant cloud environment within CPU-DPU heterogeneous systems, where each tenant holds multiple independent tasks. This multi-user multi-task scenario poses challenges to system design since we must consider both the offloading decisions of all user tasks and the competition for shared hardware resources. To tackle this challenge, we formulate the offloading decision and resource allocation question as a Mixed Integer Programming (MIP) problem and further partition it into two subproblems to alleviate the complexity of problem-solving. The primary objective is to minimize host resource occupation and maximize task processing efficiency. To accommodate dynamic workload changes, DORM also incorporates an orchestrator agent to detect load variations and adjust the offloading and resource allocation strategy based on obtained results.

The main contributions of our work are as follows.

- We introduce DORM, a dynamic DPU offloading architecture designed to release valuable host CPU resources while optimizing request processing efficiency. Leveraging both workload characteristics and runtime statistics, DORM enables optimal scheduling for each tenant application in CPU-DPU heterogeneous systems.
- We study the joint optimization of offloading decisions and resource allocation strategies, formulated as a challenging mixed-integer optimization problem. To streamline the complexity of model-solving, we decompose it into two tractable subproblems to determine the binary offloading decisions and obtain the continuous resource allocation policy, respectively. Moreover, we incorporate an orchestrator agent to accommodate dynamic workload variations and enable adaptive scheduling.
- We compare DORM with the no-offload method DPDK, full-offloading approaches FairNIC and LogNIC, as well as the dynamic offloading strategy iPipe in terms of host CPU saving, throughput, average latency, p99 tail latency, and fairness. Experimental results show that DORM achieves significant host CPU and latency savings while ensuring fair resource allocation.

The rest of this paper is organized as follows: Section II describes the background and motivation. In Section III, we illustrate the design of DORM in detail. Section IV describes the implementation. Experimental results are demonstrated in Section V. Section VI introduces related work. Finally, the conclusion and future work is drawn in Section VII.

II. Background and Motivation

A. On-Going DPUs in Industry

Recently, DPUs have emerged in the data center, offloading various tasks from the host CPU and releasing valuable CPU computing power. Based on their core processors, existing DPUs from the industry can be categorized into three types: FPGA-based, ASIC-based, and SoC-based [5].

FPGA-based DPU offers reconfigurable integrated circuits. Programmers are able to customize the FPGA logic for packet processing and achieve faster performance without introducing much energy consumption. However, to make full use of the chip, it requires dedicated programmers with sufficient skills in Hardware Description Language (HDL) [18]. And it is too expensive for large-scale deployment in data centers [8].

ASIC-based DPU contains hundreds of multi-threaded RISC cores and specialized hardware functions, enabling them to execute parallel workloads with minimal latency [19]. However, compared with other DPUs, they have the worst flexibility and may not be able to handle complex scenarios [14].

Multicore SoC-based DPU usually holds various hardware accelerators for packet processing and specialized functions, PCIe interface for host communication, multicore processors, and onboard memory. Separate from the host system, they are able to run their own operating systems (commonly Linux), making them easier to program [20] and obtain the maximum flexibility compared with the fronted two categories. In this paper, we implement our dynamic offloading architecture DORM on a typical SoC-based DPU Nvidia BlueField-2 [21].

B. Motivation

Although existing research has well explored the potential of DPU in performance isolation and acceleration, there still exist some issues that remain unsolved.

First, full offloading may overwhelm the hardware and hurt overall performance. In order to comprehensively assess the heterogeneous system and demonstrate the impacts of offloading ratios, we characterize four typical data center applications under various offloading ratios between the host CPU and DPU. Details of the experimental setup are given in Section V-A. Take the 50% offloading ratio as an example, we randomly assign half of the tasks on the host CPU and offload the rest of them to the DPU. Note that, considering the variance in processing cores and computing capabilities, we only use eight cores for both the x86 host and DPU. Experimental results under various offloading ratios using native DPDK are depicted in Fig. 1. It's worth noting that offloading all the tasks to DPU does not yield the best performance since DPU computing resources may become saturated. Moreover, ideal offloading ratios differ among various applications and packet sizes, rendering a one-size-fits-all approach impractical. Fig. 2 demonstrates the impact of multi-tenancy on offloading ratios. We increase the number of tenants from 2 to 8 and report the throughput result for three offloading ratios: CPU only, DPU only, and 50% offload to DPU, with the packet size set to 512B. The acceleration provided by DPU offloading may even be offset along with the

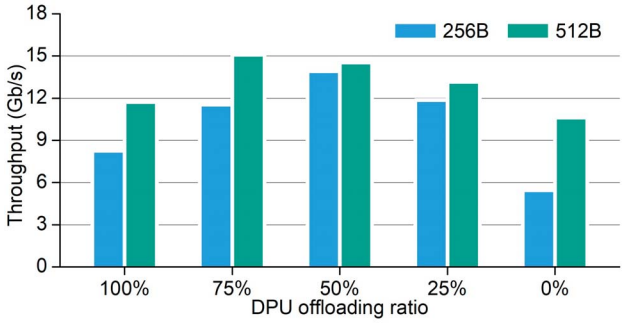


Fig. 1. Performance under varied offloading ratios and packet sizes: optimal offloading ratios vary across different applications and packet sizes.

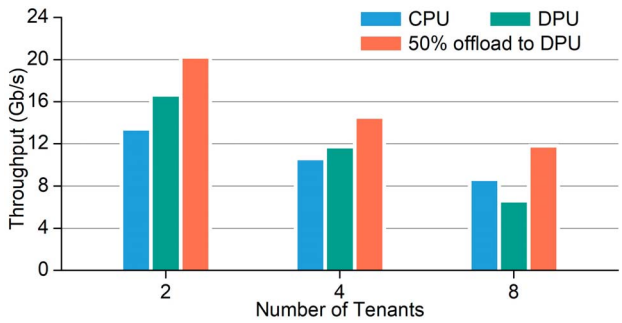


Fig. 2. The impact of multi-tenancy on DPU offloading: acceleration provided by DPU offloading may even be offset as tenant number increases.

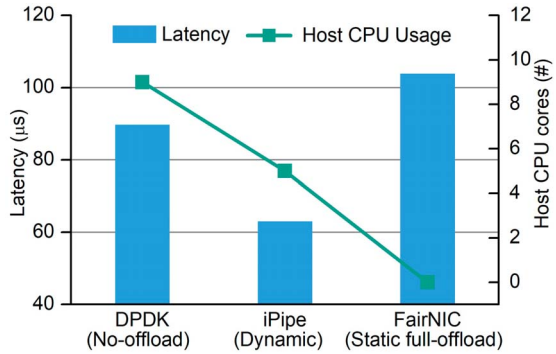


Fig. 3. Performance comparison for four tenants under 256B packet size.

increase in the number of tenants. Therefore, to make full use of available heterogeneous hardware resources, it is necessary to propose a dynamic offloading strategy and select optimal offloading ratios for task processing.

Second, identifying what types of applications benefit most to be executed on DPU and selecting the optimal target device for each application could be troublesome. To investigate the limitations of existing research, we evaluate the average latency and host CPU usage of three representative offloading strategies: DPDK, FairNIC [14], and iPipe [13]. Experimental results for four tenants under the packet size of 256B are shown in Fig. 3. DPDK, which represents a no-offload method, suffers from considerable request delays due to the overhead of packet transmission between the DPU and host, as each packet initially flows through the DPU. FairNIC, which adopts a static full-offloading strategy, offloads all computation to the DPU and effectively

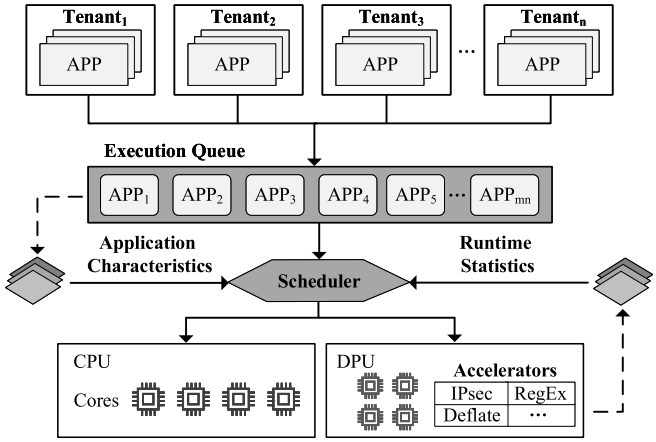


Fig. 4. Architecture of DORM.

frees up host CPU resources. However, the request processing efficiency is limited by the DPU's wimpy cores, resulting in higher system latency. iPipe introduces dynamic offloading by migrating the most demanding tasks to the host side when DPU queue buildup is detected. Although this approach reduces request latency and host CPU usage, it still fails to fully optimize resource utilization, as the selected migration tasks may not be the optimal candidates. Based on the above observation, we propose a dynamic offloading architecture, considering workload characteristics and runtime features to enable optimal scheduling for each application.

III. DORM Architecture

To maximize hardware resource utilization and offloading benefits, we introduce DORM, a dynamic offloading architecture designed for multi-tenant cloud computing systems with CPU-DPU heterogeneous platforms. Our primary goal is to minimize the host CPU core occupation and reduce the processing latency of all users, thereby making optimal offloading decisions and resource allocation for tenants.

A. Design Overview

Considering n tenants are deployed in the system, each having m independent tasks. The architecture of DORM is shown in Fig. 4. In order to select the optimal target device (CPU or DPU) for each tenant application, we first gather application-specific characteristics that provide a comprehensive understanding of the computational demands of each task. Besides, we also capture the runtime performance of both CPU and DPU to account for dynamic behaviors and system capacity, which is essential for real-time decision-making. Inspired by NFCompass [22], we adopt a run-time plus offline profiling to collect the key parameters required for scheduling decisions. Specifically, traffic-related statistics, such as packet sizes, as well as runtime performance metrics, including resource utilization and processing latency, are collected through run-time profiling. Meanwhile, offline profiling is used to extract application-specific characteristics, such as resource dependencies on multi-core processors and hardware accelerators. Then, we abstract the problem for offloading decisions and resource allocation into a mixed-integer programming problem and make optimal scheduling based on the collected information. To streamline the model-solving process, we decompose the proposed model into two subproblems: a 0-1 integer programming problem for offloading decision and a convex optimization problem for resource allocation. Our objective is to minimize host resource consumption and maximize request processing efficiency, offloading computation to DPU as far as possible to release valuable host CPU resources while maintaining the request processing efficiency.

B. Offloading Decision and Resource Allocation Model

1) Users and tasks: Denote the user set as $N = \{1, \cdots, n\}$. Each user holds multiple tasks $M = \{1, \cdots, m\}$ to be executed either at the host CPU or offloaded to DPU. Note that we assume the same number of tasks m for all users only to simplify mathematical notation. The proposed model can be easily extended to other cases with different numbers of tasks. Assume that computation tasks are independent of each other and cannot be further divided into sub-tasks, therefore each task must be executed in a single processor. According to whether user i's task j T_{ij} is processed at the host CPU or offloaded to DPU, we denote the offloading decisions as x_{ij}^C, $x_{ij}^D \in \{0, 1\}$, which are constrained by:

$$x_{ij}^C + x_{ij}^D = 1, \tag{1}$$

where only one of x_{ij}^C and x_{ij}^D for user i's task j T_{ij} could be non-zero.

2) Task processing model: Denote the total available resource capacity of the host and DPU in terms of CPU cycles as C and D, respectively. The parameter e_{ij} represents the expected CPU cycles required by T_{ij} to complete the request, and r_{ij} denotes the proportionality factor of the computing resource capacity allocated to T_{ij}. We further express each computation task T_{ij} as (S_{ij}, L_{ij}), where S_{ij} represents the request size, which may impact the resource allocation parameters e_{ij} and r_{ij}. Additionally, L_{ij} denotes the request processing latency, which can be expressed as follows:

$$L_{ij}^C = \frac{e_{ij}}{r_{ij} \times C}, \tag{2}$$

$$L_{ij}^D = \frac{e_{ij}}{r_{ij} \times D}, \tag{3}$$

where L_{ij}^C denotes the latency for task T_{ij} when deployed on the host CPU, and L_{ij}^D denotes the latency for the DPU.

To represent the task dependency on hardware accelerators, we introduce the variable ACC, where ACC_{ij}^k indicates whether task T_{ij} can be accelerated by ACC_k or not. Specifically, a value of 1 indicates true, otherwise, the value is 0. If applications can be offloaded to dedicated hardware accelerators, multi-core processor utilization on the DPU can be significantly reduced. Overall, the cumulative sum of allocated resources must remain

TABLE II
NOTATIONS AND DESCRIPTIONS

Notation	Description
N	The user set
M	The task set of each user
n	The number of tenants
m	The number of tasks
K	The number of hardware accelerators on DPU
T_{ij}	jth task for Tenant i
S_{ij}	Request size of T_{ij}
x_{ij}^C x_{ij}^D	Indicate whether T_{ij} is deployed on CPU or DPU
e_{ij}	Expected CPU cycles to complete the request
r_{ij}	Proportionality factor of resource allocated to T_{ij}
$X = \{x_{ij}\}$	Offloading decisions
$R = \{r_{ij}\}$	Resource allocation policy
ACC_{ij}^k	Dependency for hardware accelerators on DPU
C D	The total resource capacity of CPU and DPU
L_{ij}^C L_{ij}^D	Processing latency of T_{ij} at host CPU or DPU
T^C T^D	The sets of tasks deployed on the host or DPU
λ_{ij} μ_{ij}	Lagrangian multipliers
α_{ij}	The dual variable

within the total computing resource capacity, as expressed by the following constraint:

$$\sum_{i=1}^{n}\sum_{j=1}^{m} r_{ij}x_{ij}^C \le 1, \tag{4}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{K} r_{ij}(1-ACC_{ij}^k)x_{ij}^D \le 1. \tag{5}$$

C. Problem Formulation and Decomposition

Then the total system cost can be expressed as the weighted sum of allocated host CPU cores and the processing delays for all users. Our primary objective is to minimize the total system cost and make optimal offloading decisions $x_{ij} = [x_{ij}^C, x_{ij}^D]$ and resource allocation r_{ij} under the following constraints that overall hardware resource requirements should be limited by the total capacity. Let $X = \{x_{ij}\}, R = \{r_{ij}\}$, then the overall objective optimization problem can be expressed as:

$$P1: \min_{X,R} \sum_{i=1}^{n}\sum_{j=1}^{m} r_{ij}x_{ij}^C + \sum_{i=1}^{n}\sum_{j=1}^{m}(L_{ij}^C x_{ij}^C + L_{ij}^D x_{ij}^D), \tag{6}$$

$$s.t. \quad (1),(4),(5),$$
$$x_{ij}^C, x_{ij}^D \in \{0,1\}, \quad \forall i,j. \tag{7}$$

Eq. (1) and Eq. (7) impose constraints on the offloading decisions, specifying that each task must be executed on a single processor, either at the host CPU or offloaded to the DPU. Constraints provided by Eq. (4) and Eq. (5) govern the resource allocation policy, ensuring that allocated resources should not surpass the total computing resource capacity. The key symbols utilized in the system model are summarized in Table II.

As described in Eq. (6), the formulated optimization problem is notably characterized as a challenging mixed-integer programming problem, where X represents binary variables and R signifies continuous variables. This problem exhibits inherent complexity, rendering it difficult to solve through conventional methods. Even if we choose to relax the binary constraint specified in Eq. (7), allowing the task offloading decision variables to take values within the range [0,1], the problem represented by Eq. (6) remains non-convex. To streamline the computational model and address this complexity, we employ the Tammer decomposition method [23] to decompose the original high complexity problem into a set of more tractable subproblems. First of all, we reformulate the objective function $P1$, as follows:

$$O(X,R) = \sum_{i=1}^{n}\sum_{j=1}^{m} r_{ij}x_{ij}^C + \sum_{i=1}^{n}\sum_{j=1}^{m}(L_{ij}^C x_{ij}^C + L_{ij}^D x_{ij}^D). \tag{8}$$

Therefore, the objective optimization problem $P1$ can be rewritten as the following equivalent form:

$$\min_X(\min_R O(X,R))$$
$$s.t. \quad (1),(4),(5),(7). \tag{9}$$

Subsequently, we decompose the objective function and constraints into the following distinct models, each featuring different types of decision variables.

1) Offloading Decision Problem: After obtaining the optimal resource allocation policy R^*, the original optimization problem presented in Eq. (6) can be reformulated as a 0-1 integer programming problem denoted as $O(X,R^*)$, where X symbolizes the offloading decision variables.

$$P1.1: \min_X O(X,R^*)$$
$$s.t. \quad (1),(7). \tag{10}$$

2) Resource Allocation Problem: To derive $O(X,R^*)$, we temporarily set $X^0 = \{x_{ij}^0\}$. Then, $O(X,R^*)$ can be reformulated as a minimization problem $O(X^0,R)$ over the variable R, where X^0 is predetermined. It's crucial to emphasize that the separation from the original optimization problem $P1$ into subproblems $P1.1$ and $P1.2$ preserves the overall optimality [23].

$$P1.2: O(X,R^*) = \min_R O(X^0,R)$$
$$s.t. \quad (4),(5). \tag{11}$$

From the objective function of $P1$, it is evident that allocated resources r_{ij} would only be non-zero when either x_{ij}^C or x_{ij}^D equals 1. Consequently, we can transform the objective function Eq. (11) into the following function:

$$O(X^0,R) = \sum_{T_{ij}\in T^C} r_{ij} + \sum_{T_{ij}\in T^C}\frac{e_{ij}}{r_{ij}\times C} + \sum_{T_{ij}\in T^D}\frac{e_{ij}}{r_{ij}\times D}, \tag{12}$$

where T^C and T^D represent the sets of tasks deployed on the host and DPU, respectively. Since the offloading decision is assumed to be determined in this subproblem, $O(X^0,R)$ is a function with respect to r_{ij}. Then we can obtain its Hessian matrix by taking partial derivatives of R, the first-order partial

Algorithm 1: Joint Optimization of Offloading Decisions and Resource Allocation.

Input: The expected resources e_{ij}, accelerator dependency ACC_{ij}^k, and the overall resource capacity C, D
Output: The offloading decisions x_{ij}^* and optimal resource allocation policy r_{ij}^*

```
for i = 1 to n do
  for j = 1 to m do
    Accquire the application characteristics information ACC_ij^k, e_ij, S_ij, and runtime statistics L_ij;
    Find out the optimal resource allocation policy r_ij^* according to P1.2;
    Obtain the optimal offloading decision x_ij^* based on P1.1;
    Update the value of L_ij;
  end for
end for
Return the optimal scheduling policy r_ij^* and x_ij^*;
End
```

derivative is expressed as follows:

$$\frac{\partial O(X^0,R)}{\partial r_{ij}} = 1 + \frac{e_{ij}C}{(r_{ij}C)^2} + \frac{e_{ij}D}{(r_{ij}D)^2}. \tag{13}$$

Eq. (14) shows the second-order partial derivative of R:

$$\frac{\partial^2 O(X^0,R)}{\partial r_{ij}\partial r_{kl}} = \begin{cases} \frac{2e_{ij}}{(r_{ij})^3 C} + \frac{2e_{ij}}{(r_{ij})^3 D}, & \text{if } i=k, j=l, \\ 0, & \text{otherwise.} \end{cases} \tag{14}$$

Due to the non-negativity of expected and allocated resources, we can obtain that the value of the second-order partial derivative $\partial^2 O(X^0,R)/\partial r_{ij}\partial r_{kl}$ are non-negative. Consequently, all the eigenvalues of the Hessian matrix H are non-negative. As a result, it can be concluded that $O(X^0,R)$ is a convex function. Since the constraints are also linear, the computing resource allocation problem $P1.2$ is characterized as a convex optimization problem.

Through the decomposition, we effectively transform the original intricate optimization problem $P1$ into a 0-1 integer programming problem $P1.1$ for offloading decision and a convex function $P1.2$ for computing resource allocation, both of which are more manageable for solving.

D. Algorithmic Solution

As depicted in Algorithm 1, to obtain the optimal policy of offloading decision x_{ij}^* and computing resource allocation r_{ij}^*, we solve the subproblem $P1.2$ and $P1.1$, respectively. The algorithm takes $O(nm)$ to traverse the task set of each tenant, where n stands for the number of tenants, and m for task count. The computation of r_{ij}^* and x_{ij}^* requires $O(K)$ time, where K is the number of accelerators. Therefore, the overall time complexity for Algorithm 1 is $O(nm+K)$.

1) Resource Allocation Policy: Initially, to solve the decomposed subproblem $P1.2$, we employ the Karush-Kuhn-Tucker (KKT) conditions, a well-known method for addressing constrained optimization problems [24]. The Lagrangian function can be expressed as follows according to Eq. (11):

$$\begin{aligned} L(R,\boldsymbol{\lambda},\boldsymbol{\mu}) &= O(X^0,R) + \lambda_{ij} g(R) + \mu_{ij} h(R), \\ g(R) &= 1 - \sum_{T_{ij}\in T^C} r_{ij}, \\ h(R) &= 1 - \sum_{T_{ij}\in T^D} \sum_{k=1}^{K} r_{ij}(1 - ACC_{ij}^k), \end{aligned} \tag{15}$$

where $g(R)$ and $h(R)$ represents the slack of constraints (4) and (5), respectively. The symbols λ_{ij} and μ_{ij} denote the Lagrangian multipliers. The optimal solution is then derived by applying the KKT conditions, which comprise four essential components:

1) stationarity

$$\frac{\partial L}{\partial R} = 1 + \frac{e_{ij}C}{(r_{ij}C)^2} + \frac{e_{ij}D}{(r_{ij}D)^2} - \lambda_{ij} - \sum_{k=1}^{K} \mu_{ij}(1 - ACC_{ij}^k) = 0, \tag{16}$$

2) primal feasibility Eq. (4) (5), 3) dual feasibility necessitating $\lambda_{ij} \geq 0$ and $\mu_{ij} \geq 0$, 4) complementary slackness $\lambda_{ij} g(R) = 0$ and $\mu_{ij} h(R) = 0$. By solving the stationarity condition along with the primal and dual feasibility conditions, we obtain the optimal resource allocation policy r_{ij}^* as follows:

$$r_{ij}^* = \sqrt{\frac{(K - \sum_{k=1}^{K} ACC_{ij}^k) CD}{e_{ij}(C+D)}}. \tag{17}$$

2) Offloading Decision-Making: As discussed above, after obtaining the optimal solution of computing resource allocation r_{ij}^*, the objective function can be transformed into a 0-1 integer programming problem $P1.1$ for $X = \{x_{ij}\}$. Since $x_{ij} = [x_{ij}^C, x_{ij}^C]$, the problem is reformulated as follows:

$$\begin{aligned} P2: &\min_{X^C, X^D} O(X^C, X^D, R^*) \\ &s.t. \quad (1), (7). \end{aligned} \tag{18}$$

To tackle $P2$, we initially relax the constraint (7) in $P2$, transforming it into a linear programming problem $P3$,

$$P3: \min_{X^C, X^D} \sum_{i=1}^{n}\sum_{j=1}^{m} r_{ij}^* x_{ij}^C + \sum_{i=1}^{n}\sum_{j=1}^{m} (L_{ij}^C x_{ij}^C + L_{ij}^D x_{ij}^D), \tag{19}$$

$$s.t. \quad x_{ij}^C + x_{ij}^D = 1, \tag{20}$$

$$x_{ij}^C, x_{ij}^D \geq 0, \quad \forall i,j. \tag{21}$$

Notably, any feasible solution of $P3$ also satisfies $P2$. Subsequently, we solve $P3$ using the primal-dual method by introducing the dual variable α_{ij} and constructing the Lagrangian

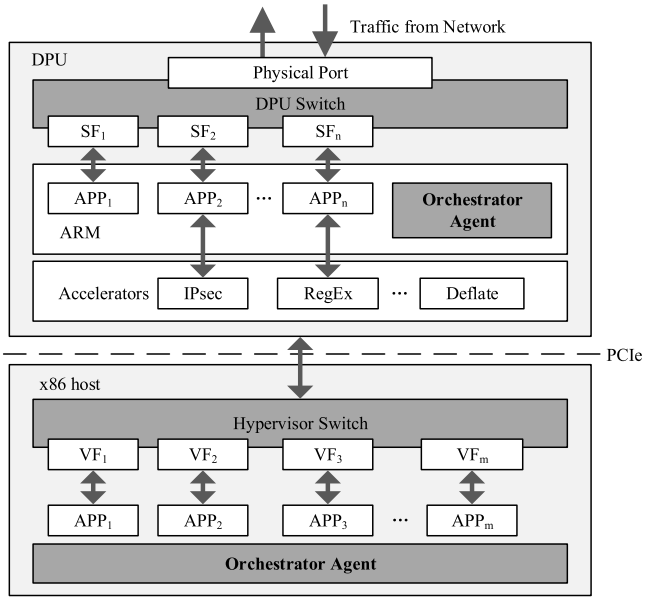


Fig. 5. Orchestrator agent to detect load changes and dynamically adjust the scheduling strategy.

function as follows:

$$\begin{aligned} L(X^C,X^D,\boldsymbol{\alpha}) &= \sum_{i=1}^{n}\sum_{j=1}^{m} r_{ij}^{*} x_{ij}^{C} \\ &+ \sum_{i=1}^{n}\sum_{j=1}^{m} (L_{ij}^{C} x_{ij}^{C} + L_{ij}^{D} x_{ij}^{D}) + \alpha_{ij}(1 - x_{ij}^{C} - x_{ij}^{D}) \\ &= \sum_{i=1}^{n}\sum_{j=1}^{m} (r_{ij}^{*} + L_{ij}^{C} - \alpha_{ij}) x_{ij}^{C} + \sum_{i=1}^{n}\sum_{j=1}^{m} (L_{ij}^{D} - \alpha_{ij}) x_{ij}^{D} + \alpha_{ij}. \end{aligned} \tag{22}$$

Since the coefficients of x_{ij}^{C} and x_{ij}^{D} must be non-negative, the dual problem $P4$ can be formulated as:

$$P4: \max_{\boldsymbol{\alpha}} \sum_{i=1}^{n}\sum_{j=1}^{m} \alpha_{ij}, \tag{23}$$

$$s.t. \quad \alpha_{ij} \le r_{ij} + L_{ij}^{C}, \quad \forall i,j, \tag{24}$$

$$\alpha_{ij} \le L_{ij}^{D}, \quad \forall i,j, \tag{25}$$

where $\boldsymbol{\alpha} = \{\alpha_{ij}\}$ denotes the set of dual variables corresponding to the constraints (20) in $P3$, and $\alpha_{ij} = min\{r_{ij} + L_{ij}^{C}, L_{ij}^{D}\}$. After solving the above simplified dual problem, the optimal offloading decisions x_{ij}^{*} can be subsequently recovered [25].

E. Adaptation to Dynamic Workload Changes

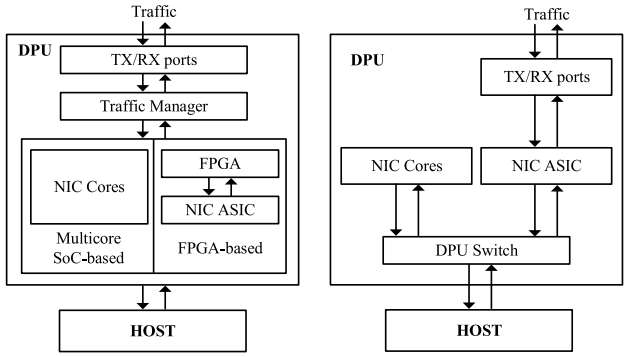
In order to effectively navigate the challenges posed by dynamic workload changes, DORM employs an adaptive approach by preserving the characteristics of each tenant application and deploying an orchestrator agent on both the host and DPU sides. As illustrated in Fig. 5, the orchestrator agent plays a crucial role in monitoring system conditions, encompassing application characteristics and runtime statistics, and triggering the migration or resource reallocation process when necessary to uphold optimal performance. To simulate the multi-tenant cloud environment, virtual machines (VMs) are instantiated on the host side using KVM, and each application is assigned a pair of virtual functions (VFs) generated through Single Root Input/Output Virtualization (SR-IOV). For applications deployed on DPU, we assign them a pair of scalable functions (SFs) which are similar to VFs on the host side.

To achieve resource isolation and alleviate the competition for shared hardware resources among multiple tenants, DORM allocates corresponding resources to each tenant based on the computed results of r_{ij}^{*}. Upon receiving a packet, DORM analyses the packet header and efficiently forwards it to the corresponding applications based on the destination IP address *dst_ip*. When a new task appears, DORM evaluates the available headroom on DPU. If there exist adequate resources on DPU, DORM preferentially deploys applications on the DPU to minimize host CPU utilization. However, if the remaining DPU resources are insufficient to satisfy the requirements of incoming requests, DORM initiates the migration process for processing efficiency optimization, adjusting application deployment strategies based on the formulated scheduling model.

Additionally, DORM actively monitors variations in tenant traffic and identifies instances of head-of-line blocking, which typically occurs in scenarios where certain packets within a network experience delays, leading to increased latency and degraded Quality of Service (QoS). In multi-tenant cloud environments, where diverse applications coexist and compete for resources, mitigating head-of-line blocking is crucial for sustaining optimal system performance. In response, DORM adopts a proactive strategy by initiating migration or resource reallocation processes, relocating appropriate applications to less congested or more resource-abundant processors, and dynamically adjusting the computing resources distribution to alleviate the contention. Similar to iPipe, we regard system latency as signals for migration and initiate the process only when substantial performance degradation is observed or DPU experiences an overload condition to avoid excessive migration. During the migration process, DORM prioritizes retaining applications that can benefit from hardware accelerators on the DPU to the fullest extent. Experimental findings presented in Sec. V-F confirm that the additional performance and resource overhead introduced by this constant monitoring and adaptive adjustments remains within acceptable limits.

IV. Implementation

DORM implementation. Our dynamic offloading architecture DORM is built on top of the Nvidia BlueField-2 DPU, which is a typical Multicore SoC-based DPU. The BlueField-2 DPU contains an Nvidia Mellanox ConnectX-6 NIC, 16GB on-board DDR4 DRAM, a variety of specific hardware accelerators, and eight ARMv8 A72 cores running a customized Linux version. In this paper, we implement DORM using the open software framework DOCA (Data-Center-Infrastructure-On-A-Chip-Architecture) [26], released by Nvidia which empowers us to develop custom applications and protocols on both the host and DPU side.



(a) On-path DPU (b) Off-path DPU

Fig. 6. DPUs can be further classified as on-path or off-path based on how their core processors interact with network traffic.

Applications development. We have developed four applications across two distinct domains using the C programming language. (1) Distributed real-time analytics (RTA). RTA [27] involves real-time analysis and processing of substantial data volumes to extract meaningful information and patterns. Pattern-matching modules [28] are employed to filter out irrelevant tuples, and a quicksort algorithm is used to sort data tuples based on the counting results. The encryption acceleration engine on the DPU is utilized for secure data encryption and decryption during RTA. (2) Network function virtualization (NFV). NFV encompasses various functions, including a packet scheduler [29], flow classifier [30], and IPv4 router [31]. The packet scheduler is responsible for managing and scheduling the order of packets transmitted through the network and ensuring that they are sent and received according to certain rules. Flow Classifier is a network function for traffic classification and identification that utilizes Naive Bayes to group the network traffic based on flow characteristics such as source or destination IP address and port numbers. The IPv4 router is responsible for routing IPv4 packets through the internet, forwarding them from the source address to the destination based on the information in the routing table. Traffic matching and classification during IPv4 router can be accelerated by the regular expression (RegEx) acceleration engine on the DPU.

Generality of DORM. In this subsection, we delve into the compatibility of DORM across various DPU models. DPUs can be further categorized into two types: on-path and off-path, depending on the way DPU cores interact with network traffic. As illustrated in Fig. 6(b), off-path DPUs, such as BlueField [21] and Stingray [32], perform off-path processing, i.e., the core processor is not directly involved in the data transmission and reception path. Instead, these DPUs leverage packet switches to determine whether data should be sent directly to the host or processed by the core processors on the DPU. In contrast, as depicted in Fig. 6(a), on-path DPUs, such as Cavium LiquidIO [33], incorporate core processors directly into the data processing path, thereby enabling streamlined and pre-scheduled data processing. While there is a distinction in the data processing path across different DPU architectures, other components exhibit minimal differences. Therefore, even though DORM is currently implemented on the off-path DPU platform Nvidia BlueField-2, the core architecture of DORM is platform-agnostic. It can be easily extended to accommodate other DPU models with varying configurations, which will be explored in our future work.

V. Evaluation

In this section, we compare our DORM with four DPU offloading architectures, including the no-offload method DPDK, full-offload solutions FairNIC [14] and LogNIC [15], as well as the dynamic offloading strategy iPipe [13] in terms of host CPU saving, throughput, average latency, p99 tail latency distribution, and fairness.

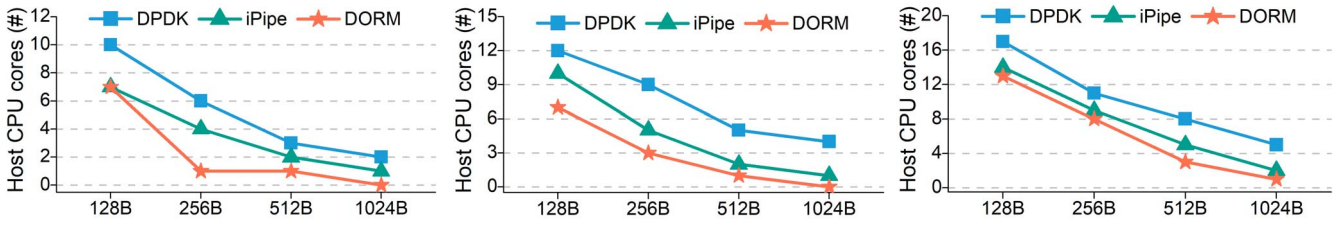
A. Experimental Setup

1) Experimental Platform: Both the client and server hold an Intel 8171M 52-core processor running at 2.6GHz, 128GB DRAM, and 512GB NVMe SSD. We equip them with an Nvidia BlueField-2 25Gbps DPU and a regular Intel E810 25Gbps NIC, respectively. Two host machines are connected back-to-back via an SFP28 cable. We install Ubuntu 20.04 as the operating system and implement our proposed DORM using the DOCA software framework. DPDK Pktgen [34] serves as the traffic loads generator, producing TCP payloads in various formats at fixed intervals to simulate traffic conditions for multiple tenants.

2) Workloads: For a comprehensive assessment of the proposed DORM, we implement four typical data center applications that present both compute-intensive and memory-intensive behaviors, including distributed real-time analytics (RTA) [27] and three network functions, i.e., packet scheduler [29], flow classifier [30], and IPv4 router [31]. We increase the number of tenants deployed on the system from 2 to 8. For two tenants, each of them runs RTA and scheduler, four tenants run the aforementioned four applications respectively. To demonstrate the effect of hardware accelerator allocation and isolation, for 8 tenants, we assign 4 of them to run RTA, 2 of them to run the IPv4 router, one to run packet scheduler, and the rest to run the flow classifier.

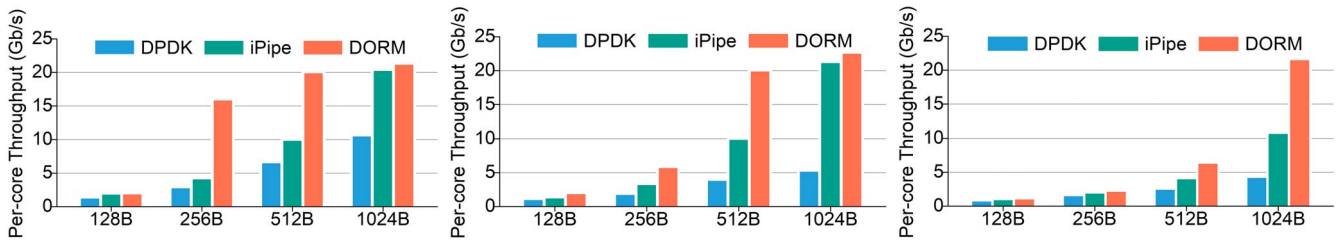
B. Host CPU Saving

Initially, we demonstrate the host CPU core savings achieved by DORM in comparison to DPDK and iPipe. Since both FairNIC and LogNIC employ full offloading strategies, running without host CPU involvement, we do not compare DORM with them in terms of host CPU saving and per-core throughput. Similar to iPipe, we configure the system to achieve the highest possible throughput and record the number of host CPU cores used by each method. Fig. 7 illustrates the overall host CPU usage under different packet sizes when achieving maximum throughput. As the packet sizes increase, the number of packets per second decreases, therefore the amount of host CPU cores required by each tenant decreases accordingly. The iPipe scheduler tends to retain lightweight tasks on DPU and moves heavyweight ones to the host which may also consume a significant number of host CPU cores. Different from iPipe, DORM



(a) Number of tenants = 2 (b) Number of tenants = 4 (c) Number of tenants = 8

Fig. 7. Host CPU usage under various packet sizes.



(a) Number of tenants = 2 (b) Number of tenants = 4 (c) Number of tenants = 8

Fig. 8. Per-core throughput analysis.

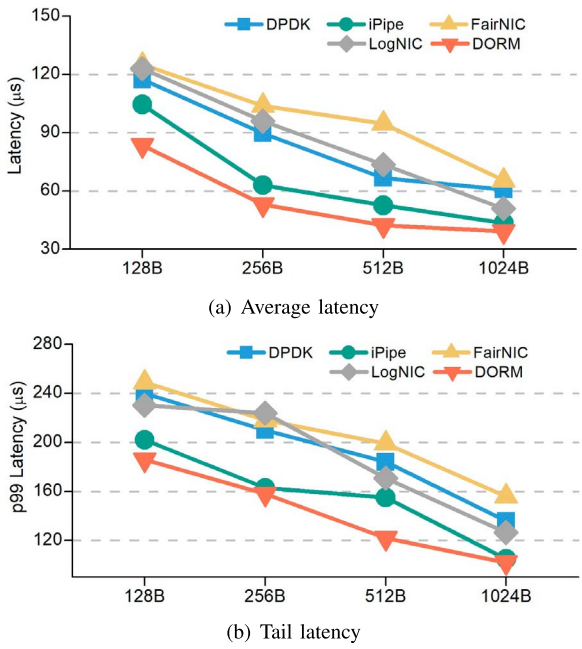
prioritizes keeping applications running on DPU as much as possible and selectively moving the most suitable tasks to the host CPU based on their workload characteristics. The experimental findings underscore DORM's remarkable capability to optimize resource management within heterogeneous systems and significantly reduce host CPU utilization. Specifically, DORM achieves up to 83.3%, and 75.0% host CPU saving compared to DPDK and iPipe, respectively.

C. Overall Performance

In this subsection, we first evaluate the per-core throughput of DPDK, iPipe, and our DORM across various packet sizes and increase the number of tenants from 2 to 8 as illustrated in Fig. 8. Subsequently, we assess the system's average latency and p99 tail latency saving across various packet sizes and network loads when the number of tenants is 4, results are presented in Fig. 9 and Fig. 10, respectively.

Per-core throughput. To obtain per-core throughput, we record the host CPU usage and overall system throughput. Experimental results under various packet sizes are depicted in Fig. 8, revealing substantial performance enhancements achieved by DORM compared to DPDK and iPipe. DORM demonstrates considerable improvements in per-core throughput, surpassing DPDK and iPipe by up to 4.61X and 3.67X, respectively. This notable improvement is attributed to DORM's ability to make optimal scheduling decisions for each tenant based on workload characteristics and runtime statistics, leading to overall enhanced performance while conserving host resources to a minimum.

Latency saving. We conduct a comprehensive assessment of the system's average latency and tail latency at the 99*th* percentile for four tenants across various scenarios, as illustrated in Fig. 9(a) and Fig. 9(b). As each packet request initially flows through DPU, DPDK experiences considerable request delays due to

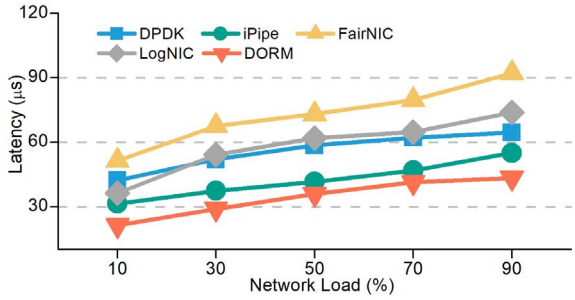


(a) Average latency

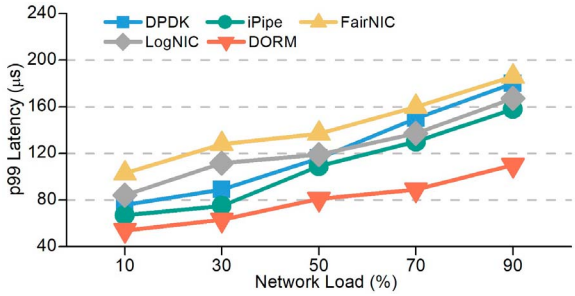
(b) Tail latency

Fig. 9. Latency performance comparison across diverse packet sizes.

packet transmission overhead between the DPU and host. While full offloading solutions like FairNIC and LogNIC effectively free up host CPUs, their request processing efficiency is limited by the DPU's wimpy cores under high-load conditions. Furthermore, FairNIC adopts a static resource partitioning strategy that struggles to accommodate the diverse resource demands of each tenant, resulting in increased latency. Although LogNIC mitigates this issue through flexible calculation partition, it still suffers from substantial performance degradation under intense load scenarios. iPipe mitigates system latency by employing a dynamic



(a) Average latency



(b) Tail latency

Fig. 10. Latency performance evaluation under various network loads.

migration strategy that transfers actors heavily load the DPU's processing capacity to the host processor when there is a queue build-up on the DPU side. However, it does not consider the specific characteristics of each tenant application, potentially leading to suboptimal migration choices. In comparison, DORM reduces average latency by up to 36.6μs, 21.0μs, 52.4μs, and 42.9μs, lowering the tail latency by up to 33.7%, 21.3%, 38.7%, and 29.4%, compared with DPDK, iPipe, FairNIC, and LogNIC, respectively. DORM optimizes the deployment of tenant applications within heterogeneous systems, effectively reducing queuing delays and request processing time.

Further, to demonstrate the capability of DORM to adapt to varying workloads, we generate 512B requests and gradually increase the networking load. Experimental results are presented in Fig. 10, which exhibits a similar trend to the findings presented in Fig. 9. As the network load escalates, system latency gradually increases due to network congestion. DORM achieves remarkable latency reductions of up to 49.5%, 32.1%, 58.5%, and 46.7%, lowering the tail latency by up to 40.7%, 31.5%, 50.8%, and 43.5%, compared with DPDK, iPipe, FairNIC, and LogNIC, respectively. Experimental results demonstrate that DORM can deliver faster response times and improved service quality for latency-sensitive tenants.

D. Fairness

Fairness is commonly utilized to evaluate the system's capability in equitably distributing resources among tenants in a shared computing environment. Ensuring fairness contributes to improving overall performance and stability. To quantitatively assess the fairness of resource allocation, we calculate the fairness index, denoted as *fair*, using the following equation

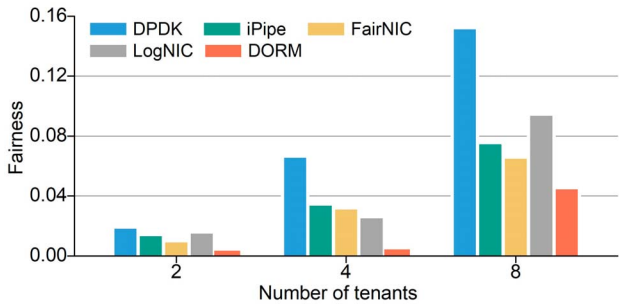


Fig. 11. Evaluation of fairness across multiple tenants, illustrating the variance between expected and actual weights, lower values indicating better fairness.

according to research [35] and [36]:

$$fair = \sum_{i=1}^{n} \left| w_i - \frac{T_i}{\sum_{i=1}^{n} T_i} \right|, \tag{26}$$

where n represents the number of tenants, w_i means the expected weight of $Tenant_i$. The numerator T_i represents the throughput of $Tenant_i$ during the interval [$t1$, $t2$], while the denominator means the overall system throughput. Consequently, the fairness metric can be expressed as the cumulative sum of the absolute difference between the expected and actual weight of each tenant. A smaller value indicates a higher level of fairness.

Fig. 11 presents the results of the fairness index under the packet size of 256B. As the number of tenants increases, each tenant receives fewer resources, leading to intensified resource competition among tenants and making it harder to maintain fairness. However, DORM improves the fairness index by up to 93.5%, 87.6%, 86.6%, and 83.6%, compared with DPDK, iPipe, FairNIC, and LogNIC. Despite FairNIC offering strict resource isolation for tenants, its static resource allocation strategy fails to accommodate the varying resource demands of multiple tenants. In contrast, DORM ensures fair resource allocation based on actual demands by considering both the workload characteristics and runtime statistics of each tenant application, thereby minimizing the deviation between ideal and actual weights.

E. Real-World Traffic Traces

To assess the responsiveness of DORM to dynamic system changes, we simulate real-world scenarios using the anonymized CAIDA dataset of Internet traffic traces [37]. CAIDA contains traces collected from high-speed monitoring systems deployed on a commercial backbone link. Although IP addresses within the dataset are anonymized, key traffic characteristics are preserved, including packet sizes, transport protocols, and detailed flow statistics, providing a realistic and representative view of actual network behavior under varying conditions. To simulate dynamic multi-tenant scenarios, we instantiate four tenants within the system, each assigned a distinct subset of traffic flows extracted from the CAIDA traces. As depicted in Fig. 12, we observe the throughput performance of one selected tenant over 150 seconds. Without appropriate resource isolation strategies for multiple tenants, DPDK and iPipe struggle to maintain stable performance under competing

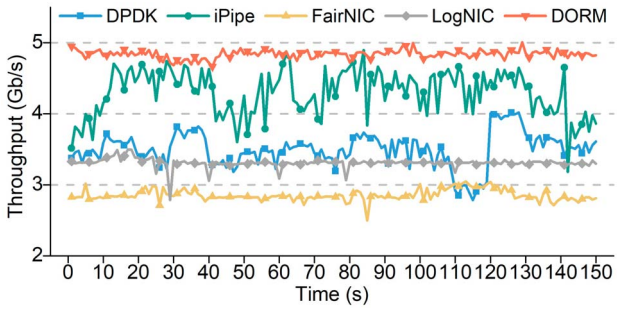


Fig. 12. Performance fluctuations in real-world traffic traces across different time epochs.

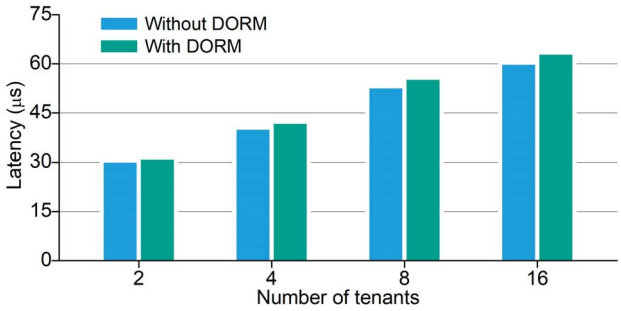


Fig. 13. Performance overhead incurred by DORM.

demands, leading to significant throughput fluctuations over time. On the other hand, FairNIC and LogNIC guarantee small performance fluctuations through strict resource isolation. However, they fall short in optimizing resource allocation among multiple tenants, resulting in notable inefficiencies. In contrast, DORM exhibits superior stability and consistent performance, effectively managing resource distribution even in the face of dynamic and unpredictable conditions.

F. Overhead

To maximize heterogeneous resource utilization and offloading benefits, DORM abstracts the problem into a mixed-integer optimization problem. However, the decision-making and scheduling process may introduce additional performance and resource overhead to the system.

To measure the overhead introduced by DORM in large-scale cloud environments, we conducted experiments by incrementally increasing the number of tenants from 2 to 16 in the packet size of 512B. As depicted in Fig. 13 and Fig. 14, we record the latency and resource utilization results under two scenarios: tenant deployment with and without our proposed DORM. Our findings demonstrate that DORM only introduces a latency increase of no more than 4.3%, which remains within acceptable bounds for performance impact. Furthermore, we measure the time required for DORM to solve the MIP problem under conditions involving 4 tenants, and it turns out that DORM only consumes approximately 3*ms* to make an optimal scheduling decision. Since DORM requires constant monitoring and adaptive adjustment, we also evaluate the resource consumption it induces. As shown in Fig. 14, DORM exhibits a modest increase in resource usage, with an average rise of 4.2%

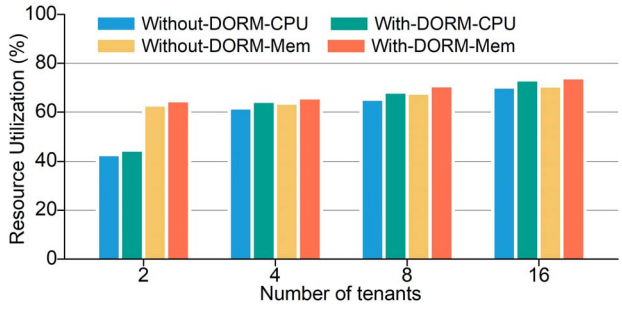


Fig. 14. Resource overhead of DORM.

in CPU utilization and 3.8% in memory consumption. Nevertheless, with the further expansion to hundreds or even thousands of tenants, the computational time required to solve the formulated MIP problem increases substantially. The heightened frequency of monitoring and resource reallocation further amplifies computational and resource burdens, resulting in delayed decision-making. In response, we intend to explore approximate heuristic algorithms in future work to reduce time complexity while preserving scheduling optimality.

VI. Related Work

DPU offloading. DPU has recently emerged as a valid option to alleviate the host CPU burden by offloading various tasks. Researchers leverage the multicore processors and hardware accelerator engines on DPUs for task acceleration [4], [6], [9], [10], [12], [13], [38], [39], [40], [41], [42]. For example, AccelTCP [6] simplifies the host stack operations by offloading stateful TCP operations to the NIC stack. LineFS [9] offloads CPU-intensive distributed file system operations and organizes them into distinct execution stages to enable pipeline parallelism. Xenic [10] proposes a DPU-accelerated distributed transaction processing system with co-designed data structures that store key-value objects in host memory and utilize DPU memory for remote access. Li et al. [43] analyze the performance of middlebox offloading, highlight the key capability of DPU in flow table management, packet processing through embedded ARM subsystem, and hardware-accelerated connection tracking. However, due to the constrained processing capability of DPU, full offloading may saturate DPU hardware resources and lead to degraded performance, so appropriate scheduling algorithms are essential to make optimal placement decisions. UNO [4] proposes a generalized SDN-controlled NF offload architecture that dynamically places NFs across DPU and host based on the current resource utilization. However, it may cause higher latency on account of the optimization objective of minimizing x86 host resource usage and prioritizing place tasks on the DPU side. E3 [12] presents a microservice execution platform for DPU-accelerated servers and places microservices among multiple DPUs according to their network topology for better energy efficiency. iPipe [13] proposes an actor-based framework for distributed application offloading. The scheduler places as much computation on the DPU as possible and migrates the actor that contributes most to the NIC's processing load to the host side when it fails to handle the incoming packets promptly.

However, all these approaches make task placement decisions based on system conditions and ignore the workload characteristics, which may potentially lead to inappropriate application deployments. In contrast, DORM dynamically selects the optimal target device for each tenant application, taking both workload characteristics and runtime statistics into consideration.

Multi-tenant resource allocation. Recently, cloud providers have started to deploy DPUs in the multi-tenant cloud environment. However, co-located tenants will compete for shared hardware resources, posing challenges to system design. LeapIO [8] introduces an innovative cloud storage stack utilizing ARM-based co-processors to offload intricate storage services and expose virtual NVMe storage to the guest VMs. Gimbal [11] proposes a software storage switch that orchestrates NVMe-oF commands among multiple co-located tenants, providing fairness and performance optimizations for tenant applications. However, these solutions mainly focus on multi-tenant storage solutions with isolation mechanisms and overlook the allocation of DPU internal resources. In contrast, FairNIC [14] takes a pioneering step in sharing DPUs across multiple tenants. It achieves isolation for each tenant in terms of typical packet processing, core cycles, shared memory, and fixed-function coprocessor access. PANIC [3] presents a hybrid push/pull packet scheduler on FPGA-based DPU and provides cross-tenant performance isolation and low-latency load-balancing across parallel offload engines. However, these approaches also lean towards a static offloading strategy, lacking adaptability to the dynamic resource demands of multiple tenants. MTDA [44] establishes an independent virtual channel for each tenant to submit offloading requests and employs a credit-based mechanism to ensure fair DPU resource allocation. SuperNIC [45] groups network tasks into virtual chains and maps them to various forms of physical chains based on system load and resource availability. However, these solutions primarily focus on DPU internal resource management. In contrast, DORM targets dynamic scheduling within CPU-DPU heterogeneous architectures, considering both the offloading decisions of all user tasks and optimal allocation of shared hardware resources.

VII. Conclusion and Future Work

In this paper, we introduce DORM, a novel DPU dynamic offloading architecture designed for multi-tenant cloud environments. DORM aims to maximize the utilization of heterogeneous hardware resources and minimize request processing latency. To achieve this, we formulate the offloading decision problem using mixed-integer programming, enabling the system to make optimal scheduling and resource allocation decisions for each tenant application. To accommodate dynamic workload changes, DORM incorporates an orchestrator agent to periodically monitor application characteristics within the system, detect load variations, and initiate task migration and resource reallocation processes as necessary. Experimental results demonstrate that DORM can effectively free up valuable CPU cycles, leading to improved overall application processing efficiency.

Future Work. Currently, the proposed dynamic offloading architecture is implemented within a single CPU-DPU heterogeneous system. In future research, we intend to investigate the feasibility of scaling this architecture to distributed clusters comprising multiple CPU-DPU nodes, facilitating effective communication and resource management across these computing nodes. Additionally, we plan to validate the portability and effectiveness of DORM across varying DPU models, such as FPGA-based Silicom C5010X and ASIC-based Intel E2000. These extensions are expected to enhance the robustness of DORM in increasingly complex large-scale data centers while providing valuable insights for the development of more efficient resource scheduling strategies in future heterogeneous computing environments.

References

[1] Z. Wang, H. Huang, J. Zhang, F. Wu, and G. Alonso, "FpgaNIC: An FPGA-based versatile 100gb SmartNIC for GPUs," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2022, pp. 967–986.

[2] A. M. Caulfield et al., "A cloud-scale acceleration architecture," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, Piscataway, NJ, USA: IEEE Press, 2016, pp. 1–13.

[3] J. Lin, K. Patel, B. E. Stephens, A. Sivaraman, and A. Akella, "PANIC: A high-performance programmable NIC for multi-tenant networks," in *Proc. 14th USENIX Symp. Oper. Syst. Des. Implement. (OSDI)*, 2020, pp. 243–259.

[4] Y. Le et al., "UNO: Uniflying host and smart NIC offload for flexible packet processing," in *Proc. Symp. Cloud Comput.*, 2017, pp. 506–519.

[5] F. Daniel et al., "Azure accelerated networking: SmartNICs in the public cloud," in *Proc. 15th USENIX Symp. Netw. Syst. Des. Implement. (NSDI)*, 2018, pp. 51–66.

[6] Y. Moon, S. Lee, M. A. Jamshed, and K. Park, "AccelTCP: Accelerating network applications with stateful TCP offloading," in *Proc. 17th USENIX Symp. Netw. Syst. Des. Implement. (NSDI)*, 2020, pp. 77–92.

[7] R. Shashidhara, T. Stamler, A. Kaufmann, and S. Peter, "FlexTOE: Flexible TCP offload with fine-grained parallelism," in *Proc. 19th USENIX Symp. Netw. Syst. Des. Implement. (NSDI)*, 2022, pp. 87–102.

[8] H. Li et al., "Leapio: Efficient and portable virtual nvme storage on arm socs," in *Proc. 25th Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2020, pp. 591–605.

[9] J. Kim et al., "Linefs: Efficient SmartNIC offload of a distributed file system with pipeline parallelism," in *Proc. ACM SIGOPS 28th Symp. Oper. Syst. Princ.*, 2021, pp. 756–771.

[10] H. N. Schuh, W. Liang, M. Liu, J. Nelson, and A. Krishnamurthy, "XeNIC: SmartNIC-accelerated distributed transactions," in *Proc. ACM SIGOPS 28th Symp. Oper. Syst. Princ.*, 2021, pp. 740–755.

[11] J. Min et al., "Gimbal: Enabling multi-tenant storage disaggregation on SmartNIC JBOFs," in *Proc. ACM SIGCOMM Conf.*, 2021, pp. 106–122.

[12] M. Liu, S. Peter, A. Krishnamurthy, and P. M. Phothilimthana, "E3: Energy-Efficient microservices on SmartNIC-Accelerated servers," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC),* 2019, pp. 363–378.

[13] M. Liu, T. Cui, H. Schuh, A. Krishnamurthy, S. Peter, and K. Gupta, "Offloading distributed applications onto SmartNICs using iPipe," in *Proc. ACM Special Interest Group Data Commun.*, 2019, pp. 318–333.

[14] S. Grant, A. Yelam, M. Bland, and A. C. Snoeren, "SmartNIC performance isolation with fairNIC: Programmable networking for the cloud," in *Proc. Annu. Conf. ACM Special Interest Group Data Commun. Appl., Technol., Archit., Protocols Comput. Commun.*, 2020, pp. 681–693.

[15] Z. Guo et al., "LogNIC: A high-level performance model for SmartNICs," in *Proc. 56th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2023, pp. 916–929.

[16] J. Liu, C. Maltzahn, C. Ulmer, and M. L. Curry, "Performance characteristics of the bluefield-2 SmartNIC," 2021, *arXiv:2105.06619*.

[17] S. Wang et al., "Smartchain: Enabling high-performance service chain partition between SmartNIC and CPU," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Piscataway, NJ, USA: IEEE Press, 2020, pp. 1–7.

[18] T. Döring, H. Stubbe, and K. Holzinger, "SmartNICs: Current trends in research and industry," *Netw. Archit. Serv.*, vol. 19, May 2021.

[19] S. Choi, M. Shahbaz, B. Prabhakar, and M. Rosenblum, "λ-NIC: Interactive serverless compute on programmable SmartNICs," in *Proc. IEEE 40th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Piscataway, NJ, USA: IEEE Press, 2020, pp. 67–77.

[20] X. Wei, R. Cheng, Y. Yang, R. Chen, and H. Chen, "Characterizing off-path SmartNIC for accelerating distributed systems," in *Proc. 17th USENIX Symp. Oper. Syst. Des. Implement. (OSDI)*, 2023, pp. 987–1004.
[21] "NVIDIA bluefield-2 DPU." NVIDIA. Accessed: Jun. 2023. [Online]. Available: https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/documents/datasheet-nvidia-bluefield-2-dpu.pdf
[22] Y. Hu and T. Li, "Enabling efficient network service function chain deployment on heterogeneous server platform," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Piscataway, NJ, USA: IEEE Press, 2018, pp. 27–39.
[23] K. Tammer, "The application of parametric optimization and imbedding to the foundation and realization of a generalized primal decomposition approach," *Math. Res.*, vol. 35, pp. 376–386, 1987.
[24] L. Vandenberghe and S. P. Boyd, *Convex Optimization.* New York, NY, USA: Cambridge Univ. Press, 2004.
[25] C. Swamy and A. Kumar, "Primal–dual algorithms for connected facility location problems," *Algorithmica*, vol. 40, pp. 245–269, Jul. 2004.
[26] "DOCA." NVIDIA. Accessed: Jul. 2024. [Online]. Available: https://developer.nvidia.com/networking/doca
[27] P. M. Phothilimthana, M. Liu, A. Kaufmann, S. Peter, R. Bodik, and T. Anderson, "Floem: A programming system for NIC-accelerated network applications," in *Proc. 13th USENIX Symp. Oper. Syst. Des. Implement. (OSDI)*, 2018, pp. 663–679.
[28] "Implementing regular expression." Russ Cox. [Online]. Available: https://swtch.com/~rsc/regexp/
[29] M. Alizadeh et al., "pFabric: Minimal near-optimal datacenter transport," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 435–446, 2013.
[30] J. Liu, Z. Tian, P. Liu, J. Jiang, and Z. Li, "An approach of semantic web service classification based on naive bayes," in *Proc. IEEE Int. Conf. Services Comput. (SCC)*, Piscataway, NJ, USA: IEEE Press, 2016, pp. 356–362.
[31] J. Kim, K. Jang, K. Lee, S. Ma, J. Shim, and S. Moon, "NBA (network balancing act) a high-performance packet processing framework for heterogeneous processors," in *Proc. 10th Eur. Conf. Comput. Syst.*, 2015, pp. 1–14.
[32] "Broadcom stingray ps250 2x50-gb high-performance data center SmartNIC." Accessed: Aug. 2024. [Online]. Available: https://docs.broadcom.com/doc/PS250-PB
[33] Cavium. "Liquidio SmartNIC family of intelligent adapters provides high performance industry-leading programmable server adapter solutions for various data center deployments." Marvell. Accessed: Aug. 2024. [Online]. Available: https://www.marvell.com/ethernet-adapters-and-controllers/liquidio-smart-nics/index.jsp
[34] K. Wiles. "The pktgen application—pktgen 3.2.4 documentation." Pktgen. Accessed: Jun. 2023. [Online]. Available: https://pktgen-dpdk.readthedocs.io/en/latest/
[35] A. Gulati, A. Merchant, M. Uysal, P. Padala, and P. Varman, "Efficient and adaptive proportional share i/o scheduling," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 37, no. 2, pp. 79–80, 2009.
[36] H. Tan, L. Huang, Z. He, Y. Lu, and X. He, "DMVL: An I/O bandwidth dynamic allocation method for virtual networks," *J. Netw. Comput. Appl.*, vol. 39, pp. 104–116, Jun. 2014.
[37] "The Caida UCSD anonymized internet traces." CAIDA. Accessed: Sep. 2024. [Online]. Available: https://www.caida.org/catalog/datasets/passive_dataset/
[38] J. Zhang et al., "SmartDS: Middle-tier-centric SmartNIC enabling application-aware message split for disaggregated block storage," in *Proc. 50th Annu. Int. Symp. Comput. Archit.*, 2023, pp. 1–13.
[39] D. Du, Q. Liu, X. Jiang, Y. Xia, B. Zang, and H. Chen, "Serverless computing on heterogeneous computers," in *Proc. 27th ACM Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2022, pp. 797–813.
[40] M. Tork, L. Maudlej, and M. Silberstein, "Lynx: A SmartNIC-driven accelerator-centric architecture for network servers," in *Proc. 25th Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2020, pp. 117–131.
[41] Y. Qiu et al., "Automated SmartNIC offloading insights for network functions," in *Proc. ACM SIGOPS 28th Symp. Oper. Syst. Princ.*, 2021, pp. 772–787.
[42] H. Ji et al., "STYX: Exploiting SmartNIC capability to reduce datacenter memory tax," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2023, pp. 619–633.
[43] F. Li, Q. Chen, J. Shen, X. Wang, and J. Cao, "Performance characteristics and guidelines of offloading middleboxes onto bluefield-2 DPU," *IEEE Trans. Comput.*, vol. 74, no. 2, pp. 609–622, Feb. 2025.
[44] Z. Huang, Y. Tan, Y. Zhu, H. Tan, and K. Li, "MTDA: Efficient and fair DPU offloading method for multiple tenants," *IEEE Trans. Services Comput.*, vol. 17, no. 6, pp. 3971–3984, Nov./Dec. 2024.
[45] W. Lin, Y. Shan, R. Kosta, A. Krishnamurthy, and Y. Zhang, "SuperNIC: An FPGA-based, cloud-oriented SmartNIC," in *Proc. ACM/SIGDA Int. Symp. Field Programmable Gate Arrays*, 2024, pp. 130–141.

Zhaoyang Huang received the B.S. degree in computer science and technology from China West Normal University of Computer Science, in 2020. She is currently working toward the Ph.D. degree with the College of Computer Science and Electronic Engineering, Hunan University, Changsha, China. Her research interests include cloud computing and data centers.

Yanjie Tan received the B.S. and M.S. degrees from Huazhong University of Science and Technology, China, in 2011 and 2015, respectively, and the Ph.D. degree from Hunan University, Changsha, China, in 2021. He is a Postdoctoral Researcher with the College of Computer Science and Electronic Engineering, Hunan University. His research interests include real-time system and image and video processing.

Yifu Zhu received the B.S. degree from the College of Electronic Science and Engineering, Jilin University, in 2019, and the M.S. degree from Hunan University, Changsha, China, in 2023. He is currently working toward the Ph.D. degree with the College of Computer Science and Electronic Engineering, Hunan University. His research interests include FPGA and real-time systems.

Huailiang Tan received the B.S. degree from the Central South University, China, in 1992, the M.S. degree from Hunan University, Changsha, China, in 1995, and the Ph.D. degree from the Central South University, China, in 2001. He has more than eight years of industrial R&D experience in the field of information technology. From 2010 to 2011, he was a Visiting Scholar with Virginia Commonwealth University. Currently, he is a Full Professor of computer science and technology with Hunan University. His research interests include high performance I/O, image and video processing, and embedded systems.

Keqin Li (Fellow, IEEE) received the B.S. degree in computer science from Tsinghua University, in 1985, and the Ph.D. degree in computer science from the University of Houston, in 1990. He is a SUNY Distinguished Professor with the State University of New York and a National Distinguished Professor with Hunan University, Changsha, China. He has authored or co-authored more than 1060 journal articles, book chapters, and refereed conference papers. Since 2020, he has been among the world's top few most influential scientists in parallel and distributed computing regarding single-year impact (ranked #2) and career-long impact (ranked #4) based on a composite indicator of the Scopus citation database. He is a member of the SUNY Distinguished Academy. He is an AAAS Fellow, an AAIA Fellow, an ACIS Fellow, and an AIIA Fellow. He is a member of Academia Europaea (Academician of the Academy of Europe).