TACS: Decentralized Per-Task Micro-Slicing for Deadline-Aware Provisioning in Distributed Computing Continuum Systems

Shujia Niu, *Student Member, IEEE*, Zhenli He, *Senior Member, IEEE*, Jixian Zhang, Cheng Xie, *Senior Member, IEEE*, and Keqin Li, *Fellow, IEEE*

***Abstract*—Distributed Computing Continuum Systems (DCCS) unify cloud, fog, edge, and Internet of Things (IoT) into a single execution fabric. At scale, heterogeneity and bursty arrivals make it hard to meet per task deadlines. Prior approaches based on learning, optimization, market mechanisms, or class level slicing depend on global state or iterative coordination. Decisions lag arrivals, isolation is scoped to coarse classes rather than individual tasks, and the deadline violation ratio (DVR) rises. We present Task-level Adaptive Computing Slicing (TACS), a fine grained slicing paradigm that delivers task aligned resource governance through autonomous shard management. For each arriving task, TACS executes decentralized scheduling at the shard level to instantiate an ephemeral micro-slice governed by an autonomous shard formed exactly by the task's participants. Within the shard, participants apply closed form rules to make local resource allocation decisions. This design achieves strict task level isolation and precise, scalable matching between supply and demand without global coordination or model retraining. In simulations with 100 to 1000 heterogeneous nodes and a range of loads and heterogeneity levels, TACS maintains DVRs below 1% and provides steadier, higher throughput than advanced baselines. Under adverse conditions, representative baselines exceed 20% DVR and in several cases require retraining when device populations change.**

***Index Terms*—Deadline-aware provisioning, decentralized scheduling, distributed computing continuum systems, heterogeneous resources, per-task micro-slicing, scalability.**

I. Introduction

A. Motivation

DISTRIBUTED Computing Continuum Systems (DCCS) integrate public clouds, fog micro-datacenters, edge clusters, and Internet of Things (IoT) endpoints into a single logical execution substrate spanning from the core to the extreme edge [1]. Edge-cloud federated systems place latency-sensitive tasks near users while offloading less time-critical workloads to cloud resources [2]. Multi-access edge computing environments likewise process real-time requests locally and redirect overflow traffic to neighboring sites or fog clusters during congestion [3]. This continuum model is increasingly adopted because it places compute close to data sources, exploits heterogeneous resources elastically, and improves service efficiency without leaving capacity idle [4].

While DCCS offer significant advantages for distributed computing, they introduce fundamental challenges for resource management. The integration of diverse computing resources across geographical boundaries creates complexity in three key dimensions: *scale*, *heterogeneity*, and *arrival burstiness*. Scale enlarges the number of devices, tasks, and control interactions. Heterogeneity widens the mismatch between task requirements and node capabilities. Arrival burstiness creates instantaneous demand surges and rapidly changing contention. Their effect is not additive. Together, they make it difficult to provide strict *per-task* real-time guarantees while still using distributed resources efficiently.

Existing studies have explored machine learning, optimization, market mechanisms, game-theoretic designs, and slicing for distributed resource management [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23]. These directions provide important capabilities, including workload adaptivity, formal guarantees, economic efficiency, strategic robustness, and service isolation. Yet their control assumptions remain poorly aligned with DCCS. Approaches that rely on centralized controllers or broad global-state collection become increasingly expensive as the platform grows, because state maintenance and coordination traffic expand with system size and geographical span [24], [25], [26], [27]. Methods based on averaged behavior, coarse abstractions, or globally uniform control also struggle to match tasks with different latency budgets, resource demands, and attribute constraints to highly diverse nodes. Learning-based approaches can adapt to workload dynamics, but in DCCS they may require retraining or incremental policy readaptation when device populations, feasible action spaces, or resource combinations change [17], [27], [28]. Optimization-based, market-based, and game-theoretic methods

Received 27 August 2025; revised 31 May 2026; accepted 3 June 2026. Date of publication 5 June 2026; date of current version 16 June 2026. This work was supported in part by the National Natural Science Foundation of China under Grant 62362068, in part by Yunnan Province Special Project under Grant 202403AP140021, in part by the Open Foundation of Yunnan Key Laboratory of Software Engineering under Grant 2023SE208, and in part by the Program for Excellent Young Talents, Yunnan, China. Recommended for acceptance by J. Xu. *(Corresponding author: Zhenli He.)*

Shujia Niu, Zhenli He, and Cheng Xie are with the Yunnan Key Laboratory of Software Engineering, Engineering Research Center of Cyberspace, Yunnan University, Kunming 650500, China, and also with the School of Software and AI, Yunnan University, Kunming 650500, China (e-mail: niushujia@stu.ynu.edu.cn; hezl@ynu.edu.cn; xiecheng@ynu.edu.cn).

Jixian Zhang is with the School of Information Science and Engineering, Yunnan University, Kunming 650504, China (e-mail: zhangjixian@ynu.edu.cn).

Keqin Li is with the Department of Computer Science, State University of New York, New Paltz, NY 12561 USA (e-mail: lik@newpaltz.edu).

This article has supplementary downloadable material available at https://doi.org/10.1109/TPDS.2026.3701061, provided by the authors.

Digital Object Identifier 10.1109/TPDS.2026.3701061

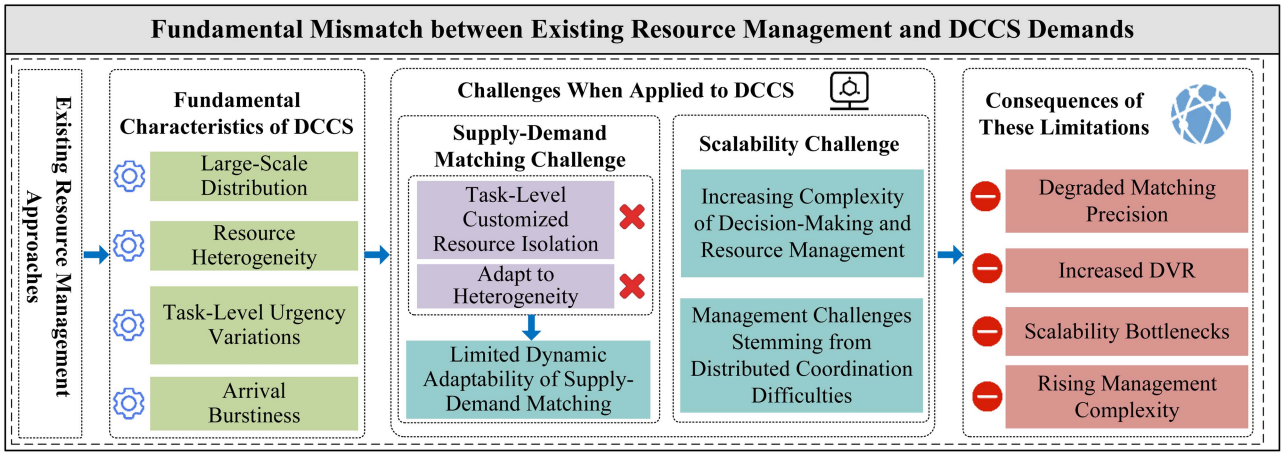


Fig. 1. How DCCS characteristics challenge existing resource management approaches.

avoid retraining, yet their iterative search, equilibrium-seeking, or repeated coordination still add non-negligible delay under bursty online arrivals [8], [9], [10], [11], [12], [13], [14], [15], [16].

To mitigate resource contention under bursty workloads, existing class-level slicing methods [21], [22], [23] typically aggregate diverse tasks into shared resource pools to provide service isolation, thereby protecting critical services while improving utilization and resilience. This constitutes an important step toward handling burst-induced contention. However, because their unit of control is the service class rather than the individual task, tasks with different urgency levels, latency budgets, and resource attribute requirements are still treated in an averaged manner within the same slice. Although more advanced studies have explored finer-grained slicing, these mechanisms generally still rely on centralized infrastructure providers to manage virtualized resource pools. As discussed above, this creates scalability bottlenecks in DCCS.

Taken together, these characteristics of DCCS do not create isolated difficulties; rather, they jointly impose a dual requirement on resource management: the platform must achieve precise supply-demand matching for heterogeneous tasks while preserving scalability under large-scale and bursty operation. As illustrated in Fig. 1, this dual requirement arises from the combined effects of scale, heterogeneity, and arrival burstiness, and it is precisely along these two dimensions that the limitations of existing approaches become most evident. This observation motivates us to design a slicing-based framework that achieves task-level supply-demand matching and robust scalability under DCCS conditions.

B. Our Contributions

To address this gap, we propose **TACS** (**T**ask-level **A**daptive **C**omputing **S**licing), a distributed per-task micro-slicing framework for DCCS. Relative to prevailing designs, TACS changes three aspects of the control plane: it moves the *unit of control* from the service class to the individual task, shifts the *scope of coordination* from platform-level control to a task-local autonomous shard, and replaces iterative or retraining-dependent online decisions with closed-form allocation derived from end-to-end latency constraints. For each arriving task, TACS instantiates an ephemeral micro-slice, forms a shard consisting only of that task's participants, and computes scheduling and allocation decisions locally from latency feasibility. In this way, TACS directly targets the two requirements identified above: precise per-task supply-demand matching and scalable control under DCCS conditions.

The main contributions of this paper are as follows:

- *Task-level micro-slicing through shard-local autonomy:* We introduce a per-task slicing paradigm in which each task is governed inside an autonomous shard formed exactly by its participants. This aligns the control boundary with the task boundary, achieves strict task-level isolation, and makes coordination cost depend on shard size rather than platform size, thereby avoiding centralized global-control bottlenecks and reducing sensitivity to changing node populations.
- *Closed-form latency-driven allocation for heterogeneous resource matching:* We derive closed-form analytical expressions from end-to-end latency constraints to compute task split ratios and per-node resource allocations directly on the critical path. Combined with score-based candidate selection and robustness-oriented provisioning, this design avoids iterative online optimization while enabling deadline-aware matching between heterogeneous task requirements and distributed resource capabilities. Under resource scarcity, a resource tension mechanism overlaps execution with continued recruitment to preserve progress and deadline awareness.
- *Comprehensive validation across scale, load, and heterogeneity:* We evaluate TACS in a simulated DCCS environment with 100 to 1000 heterogeneous nodes under varied arrival intensity, offered load, and heterogeneity levels. TACS sustains deadline violation rates between **0%** and **0.45%** with stable throughput and no retraining, whereas representative advanced baselines exceed **20%** deadline violation under adverse conditions. These results show that TACS maintains deadline-aware provisioning while scaling robustly across diverse DCCS settings.

To the best of our knowledge, TACS is the first framework to realize task-level resource provisioning in DCCS through decentralized shard governance with ephemeral micro-slices and latency-constraint-driven closed-form allocation.

The remainder of this paper is organized as follows: Section II reviews related research. Section III presents the node, task, and latency models. Section IV formulates the problem and objectives. Section V details TACS, including candidate selection, closed-form allocation, shard life-cycle management, and the resource tension mechanism. Section VI reports the experimental evaluation. Section VII concludes the paper.

II. Related Work

We review prior studies through two lenses that map directly to our problem setting in DCCS: *how well the platform matches supply with demand at decision time* and *how well the platform scales as topology, load, and requirements evolve*. Due to space constraints, we focus here on discussing the core limitations of existing approaches, while a comprehensive analysis of related work is provided in Section I of the supplementary material.

A. Supply-Demand Matching and Scalability in DCCS

Effective resource management in DCCS must jointly address two requirements: efficient matching between heterogeneous supply and dynamic demand, and scalable decision-making as the system evolves. Existing studies provide important foundations, but their control assumptions remain difficult to sustain under DCCS conditions.

Machine learning methods, including supervised prediction and reinforcement learning, offer strong adaptivity [5], [6], [7]. Their main difficulty in DCCS is not a lack of modeling power, but the dependence of decision quality on timely state information and stable action semantics. As heterogeneity widens and deployments grow, stale state, changing device populations, and evolving feasible action spaces can reduce policy accuracy and trigger retraining or incremental policy readaptation, thereby adding control-plane delay and weakening responsiveness to topology and workload changes [5], [6], [7].

Formal models based on optimization [8], [9], [10], market mechanisms [11], [12], [13], and game theory [14], [15], [16] provide performance guarantees and economic efficiency. However, these strengths typically rely on iterative search, equilibrium-seeking, or repeated coordination. As the number of resources and participants increases, the resulting computational and communication overhead becomes increasingly difficult to reconcile with low-latency, per-task provisioning under bursty arrivals [8], [9], [10], [11], [12], [13], [14], [15], [16].

Slicing is the most relevant line of work for isolation under contention [17], [18], [19], [20], [21]. Its main limitation in DCCS lies in control granularity. Most existing slicing mechanisms operate at the service-class level, so tasks with different deadlines, urgency levels, and attribute constraints remain grouped into shared resource pools and are still handled in an averaged manner. Even dynamic slicing approaches commonly rely on centralized management models that are ill-suited to DCCS scale [29], or incur additional coordination overhead to maintain policy consistency across distributed units [19], [20], [29], [30], [31], [32]. This weakens their ability to provide precise per-task matching while remaining scalable in large DCCS.

Taken together, prior work has established essential building blocks, but not the specific combination required here: *precise per-task supply-demand matching* together with *scalable coordination under scale, heterogeneity, and arrival burstiness*.

B. Position of TACS

The main distinction of TACS lies in two design choices that are fundamental to DCCS resource management: the *unit of control* and the *scope of coordination*.

Unit of control: Existing slicing-based methods provide effective isolation, but their primary abstraction is typically the service class or shared virtualized resource pool rather than the individual task [17], [18], [19], [20], [21]. In DCCS, this leaves tasks with different latency budgets, urgency levels, and attribute requirements inside the same control boundary, which limits matching precision at the task level. TACS differs by moving the unit of control from the service class to the individual task through ephemeral micro-slices. Together with closed-form analytical expressions derived from end-to-end latency constraints, this allows resource decisions to be tied directly to each task's own feasibility requirements.

Scope of coordination: Many existing approaches rely on broad coordination scopes, centralized resource governance, or extensive cross-system state maintenance to make allocation decisions. Such mechanisms can be effective in smaller or more homogeneous settings, but become increasingly costly in DCCS as the system expands in scale, spans geographically distributed resources, and undergoes continual state variation under bursty arrivals. TACS differs by confining coordination to an autonomous shard formed only by the participants of the task under consideration. As a result, control cost scales with shard size rather than with the full platform.

From this perspective, TACS addresses a requirement that prior studies do not explicitly resolve in DCCS: achieving precise task-level supply-demand matching without coupling online control cost to platform-wide coordination. For completeness and traceability, a compact comparison across key dimensions is provided in Section I of the supplementary material with references to the corresponding studies.

III. System Model

This section fixes the abstractions and notation used throughout the paper. We first unify heterogeneous continuum resources into a common node model, then state the task model and divisibility assumptions, and finally present a latency model that connects end-to-end deadlines with the resources provisioned by shard participants. For ease of reference, Table I provides a condensed guide to the symbols used in the modeling and algorithmic workflow, while the complete notation summary is retained in the supplementary material.

Key modeling assumptions: This paper adopts the following modeling assumptions to make per-task supply–demand

TABLE I
CONDENSED NOTATION GUIDE FOR THE MODELING AND ALGORITHMIC WORKFLOW OF TACS

Symbol	Meaning	Symbol	Meaning
Core model symbols			
t_r	task	v_k	node
p_r	required node attributes	L_r	end-to-end deadline
$\breve{T}_r$	forwarding delay to a qualified committee node	θ_k	task split ratio assigned to v_k
φ_r	result-to-input size ratio	u_a	aggregator uplink allocated to t_r
RTT_k	round-trip time between the aggregator and v_k	$T^w_{r,k}$	waiting latency component
$T^t_{r,k}$	communication latency component	$T^e_{r,k}$	execution latency component
TACS scheduling			
T_s	conservative latency surrogate for candidate evaluation	y_k	normalized reciprocal of T_s
$S(t_r, v_k)$	composite evaluation score	$\breve{p}^k_r$	predicted position of t_r in node v_k's invitation queue
E	minimum theoretically sufficient candidate count	W	actual invitation count with redundancy
N_s	number of admissible nodes with nonzero allocable capacity	λ_t	recent arrival rate of tasks with attributes p_r
μ_c	minimum redundancy invitation factor		
Final allocation and tension handling			
M^+_r	positive-response set	M_r	final participation set after normal scheduling
$M^{\text{incomplete}}_r$	inherited incomplete participant set passed to the tension mechanism	M^{final}_r	final participant set under the tension mechanism
$\tilde{c}_Q$, $\tilde{g}_Q$	minimum CPU/GPU resources required by the last admitted node	$\ddot{c}_Q$, $\ddot{g}_Q$	reserved CPU/GPU resources after redundancy
ε^Q_c, ε^Q_g	CPU/GPU redundancy factors for the last admitted node	$\dot{\varepsilon}$	redundancy growth cap in normal scheduling
$\hat{\varepsilon}$	perceived shortage degree in tension mode	H_r	number of dynamic recruitment rounds in tension mode

matching tractable under strict deadlines. First, each admitted task is divisible (malleable) and can be split proportionally across participating nodes as in (4), so that data and computation shares follow the same ratio and subtasks execute in parallel within a shard. Second, attribute satisfaction is a hard admission constraint: a node is eligible for participation only if the task's required attributes are contained in the node's advertised attributes, and scheduling and allocation are performed to meet the task's end-to-end deadline through the feasibility constraint derived in the latency model. Third, consistent with the modeling approach adopted in recent top-tier literature [33], we consider the post-aggregation delivery latency to the user negligible and therefore omit it from the end-to-end deadline accounting.

A. Node Model

Following the continuum view of Dustdar et al. [1], we treat public clouds, fog micro-datacenters, edge clusters, and IoT endpoints as elements of a single pool of computational nodes. Let

$$V = \{v_1, v_2, \ldots, v_Z\}$$

be the set of all nodes in the DCCS, where Z is the platform size. Each node v_k exposes its *currently allocable* resources

$$\overline{\mathbf{v}}_k \triangleq \{\, c_k,\, g_k,\, u_k,\, p_k \,\}. \tag{1}$$

Here, c_k (in GFLOP/s) and g_k (in GFLOP/s) represent the total currently available CPU and GPU compute resources, respectively. Note that the currently available resources are not the total resources C_k (in GFLOP/s) and G_k (in GFLOP/s) that node v_k can offer to the DCCS. They are the resource remainders after subtracting the already reserved and allocated resources from C_k and G_k, respectively, which is the amount of resources that can be reserved at present. The term u_k is the allocable uplink bandwidth in MB/s. The set p_k enumerates the node's special attributes.

Heterogeneity in DCCS is fundamental [34]. We model capability matching with attribute sets. Let L denote the universal set of attributes supported by the platform (for example, CPU instruction set architecture (ISA), accelerator type, driver stack, trusted execution). Each node advertises $p_k \subseteq L$. A task requiring $p_r \subseteq L$ may execute at v_k only if $p_r \subseteq p_k$. For efficient lookup, we form resource classes

$$\widetilde{p}_w \triangleq \{\, v_k \mid A_w \subseteq p_k \,\},$$

where $A_w \subseteq L$ is the defining attribute combination. The collection $\widetilde{P} = \{\widetilde{p}_1, \ldots, \widetilde{p}_Q\}$ may overlap; nodes in intersections (that is, $v_k \in \widetilde{p}_i \cap \widetilde{p}_j$) can satisfy multiple requirement profiles and are especially useful when shards grow on demand.

DCCS are open, multi-stakeholder ecosystems [34]. We therefore track four scalar attributes per node to support policy and governance without centralization: a balance b_k for incentives, a cumulative completion record $h_k = \sum h_r$ over tasks t_r with per-task indicators $h_r \in \{0, 1\}$, a reputation score R_k derived from h_k and quality-of-service (QoS) behavior, and a committee identity $\tilde{c}_k \triangleq [\dot{c}_k, \eta_k]$ where $\dot{c}_k \in \{0, 1\}$ marks committee membership and η_k is an election score that can depend on R_k and resource headroom. After each task, the platform updates reputation for both aggregator and executor roles; weights can

differ to reflect coordination effectiveness versus computational reliability. TACS is agnostic to the specific update formula and accepts any policy consistent with these signals.

B. Task Model

We adopt the Three-Stage Heterogeneous Computing (TSHC) structure from [35] and model each t_r as a divisible task whose workload can be partitioned proportionally and executed in parallel across multiple participating nodes, which is consistent with common DCCS workloads such as data-parallel processing, inference sharding, and batch/block-based analytics. A task is represented as

$$t_r \triangleq \{\, d_r,\ f_r,\ a_r,\ b_r,\ p_r,\ L_r,\ \dot{p}_r \,\}. \tag{2}$$

The quantity d_r (MB) is the input size, and f_r (GFLOP) is the total floating-point work. The parameter $a_r \in [0,1]$ is the fraction of work in the data computation stage; $1-a_r$ covers CPU preprocessing and final aggregation. Within the data computation stage, $b_r \in [0,1]$ is the fraction of parallel work mapped to GPUs, while $(1-b_r)$ is serial CPU work. The set $p_r \subseteq L$ specifies required attributes. The scalar L_r is the end-to-end deadline from offload to completion. The reward $\dot{p}_r$ accounts for forwarding, committee, and execution incentives according to the deployment's policy.

Let $V_r = \{v_1, \ldots, v_Q\} \subseteq V$ be the participants selected for t_r. The aggregator assigns to v_k an input share d_r^k and a compute share f_r^k such that

$$\sum_k d_r^k = d_r, \qquad \sum_k f_r^k = f_r. \tag{3}$$

Because tasks are divisible, data and compute splits follow the same ratio. We define

$$\theta_k \triangleq \frac{d_r^k}{d_r} = \frac{f_r^k}{f_r}, \tag{4}$$

which is the fraction of t_r executed by v_k. Under light load, a capable node can take $\theta_k = 1$. Under contention, nodes contribute *fragmented* capacity with $\theta_k < 1$, and many such contributions can be aggregated within the shard. We also use $\varphi_r \geq 0$ to denote the result-to-input size ratio, which links compute and communication for t_r.

C. Latency Model

End-to-end latency aggregates decision making, data movement, and execution. Parallelism implies that the slowest participant determines completion. We present a compact model that TACS inverts to compute per-node quotas.

Non-congested regime: When the platform is not congested, all subtasks share the same decision-making latency T_r^w incurred once per task. The end-to-end latency is

$$T_r = T_r^w + \max_{v_k \in V_r} \left\{ T_{r,k}^e + T_{r,k}^t \right\}. \tag{5}$$

Contended regime: Under contention, waiting may interleave with communication and execution. A conservative expression is

$$T_r = \max_{v_k \in V_r} \left\{ T_{r,k}^w + T_{r,k}^e + T_{r,k}^t \right\}, \tag{6}$$

where $T_{r,k}^w$ captures the per-node waiting experienced along the path to the final participation list.

1) Decision-Making Latency: A node that receives t_r first checks attributes. If $p_r \nsubseteq p_k$ or the node is not eligible to aggregate, it signs and forwards the request to a suitable committee set; the forwarding time is $\breve{T}_r$. A qualified committee node becomes the *aggregator*, filters candidates by $p_r \subseteq p_k$, and probes W nodes in parallel to confirm participation. In the non-congested regime,

$$T_r^w = \breve{T}_r + \max(\mathrm{RTT}_1, \ldots, \mathrm{RTT}_W), \tag{7}$$

where RTT_k is the measured round-trip time between the aggregator and v_k. In practice, TACS records T_r^w with counters; the expression above provides structure for analysis and parameter setting.

2) Communication Latency: Communication has two legs: the aggregator sends d_r^k to v_k, and v_k returns $\varphi_r d_r^k$. We account for sender-side bandwidth in each leg, which is standard in peer-to-peer models. The communication latency for v_k is

$$T_{r,k}^t = \frac{d_r^k}{u_a} + \frac{\varphi_r\, d_r^k}{u_k} + \mathrm{RTT}_k, \tag{8}$$

where u_a is the aggregator's allocable uplink for t_r and u_k is the executor's allocable uplink. Within a task, the aggregator assigns its uplink proportionally to θ_k in (4). When multiple tasks contend for the same uplink, TACS orders transmissions by an urgency level

$$e_r = \frac{L_r - \overline{T_{r,Q}^e}}{L_r}, \tag{9}$$

where $\overline{T_{r,Q}^e}$ is the expected remaining compute latency under current allocations. This simple priority steers scarce bandwidth to tighter budgets.

3) Execution Latency: Execution time depends on node speeds and the TSHC composition [35]. For the assigned work f_r^k on v_k with CPU rate c_k and GPU rate g_k, the CPU-only phases (preprocessing and final aggregation) take $\frac{f_r^k(1-a_r)}{c_k}$. Within the data computation stage, serial CPU work takes $\frac{f_r^k a_r(1-b_r)}{c_k}$ and parallel GPU work takes $\frac{f_r^k a_r b_r}{g_k}$. These stages overlap, hence

$$T_{r,k}^e = \frac{f_r^k(1-a_r)}{c_k} + \max\left(\frac{f_r^k a_r(1-b_r)}{c_k},\ \frac{f_r^k a_r b_r}{g_k} \right). \tag{10}$$

4) Latency Feasibility: For t_r to meet its deadline at v_k, the remaining budget after decision and communication must accommodate execution:

$$T_{r,k}^e \leq L_r - T_{r,k}^w - \left(\frac{d_r^k}{u_a} + \frac{\varphi_r\, d_r^k}{u_k} + \mathrm{RTT}_k \right). \tag{11}$$

This inequality expresses the latency-feasibility condition for node v_k under the deadline constraint: after the decision latency and communication latency are deducted from the task deadline

L_r, the residual time budget must still be sufficient for execution at node v_k.

IV. Problem Definition

Given the node, task, and latency models in Section III, we formalize the per-task provisioning problem for a DCCS operator. The platform observes an online stream of arbitrarily divisible tasks $t_r \triangleq \{d_r, f_r, a_r, b_r, p_r, L_r, \dot{p}_r\}$ and a time-varying pool of heterogeneous nodes $V = \{v_1, \ldots, v_Z\}$, where each node v_k exposes currently allocable resources $\overline{\mathbf{v}}_k \triangleq \{c_k, g_k, u_k, p_k\}$ as in (1). The objective of this study is to devise a task-level resource management mechanism that includes two fundamental subproblems: (1) scheduling, which determines which nodes participate in task execution, and (2) resource allocation, which determines the per-node resource contributions and task split ratios θ_k when multiple nodes collaborate.

A. Decision Context

Given a task $t_r = \{d_r, f_r, a_r, b_r, p_r, L_r, \dot{p}_r\}$ and the current node set V, the following information is available: the input size d_r, compute requirement f_r, CPU-GPU split parameters a_r and b_r, special attribute requirement set $p_r \subseteq L$, deadline L_r, and reward parameter $\dot{p}_r$; current allocable resources $\overline{\mathbf{v}}_k$ of candidate nodes satisfying $p_r \subseteq p_k$; measured round-trip times RTT_k and the forwarding time $\breve{T}_r$; and the result-to-input size ratio $\varphi_r \geq 0$.

B. Decision Variables

For each t_r, the algorithm determines:

- an aggregator node selected from qualified committee candidates,
- a final participation list $M_r \subseteq \{v_k \in V \mid p_r \subseteq p_k\}$,
- allocation decisions $\{\theta_k, \ddot{c}_k, \ddot{g}_k\}_{v_k \in M_r}$ where:
  - $\theta_k \in [0, 1]$ is the task split ratio with $\sum_{v_k \in M_r} \theta_k = 1$,
  - $\ddot{c}_k$ and $\ddot{g}_k$ are the CPU and GPU resources that node v_k allocates to task t_r.

C. Feasibility Constraints

The allocation decisions must satisfy the following constraints:

1) Latency Feasibility: For each participating node $v_k \in M_r$, the execution time must fit within the remaining deadline budget: $T^e_{r,k} \leq L_r - T^w_{r,k} - T^t_{r,k}$.

2) Resource Capacity Constraints: Node resource allocations cannot exceed currently allocable resources:

$$\ddot{c}_k \leq c_k, \quad \ddot{g}_k \leq g_k, \quad \text{for all } v_k \in M_r. \tag{12}$$

Bandwidth allocations must comply with the uplink constraints of the aggregator (u_a) and the nodes (u_k).

3) Attribute Compatibility: All participating nodes must satisfy the task's attribute requirements:

$$p_r \subseteq p_k \quad \text{for all } v_k \in M_r.$$

4) Task Completeness: The task must be fully allocated across participating nodes: $\sum_{v_k \in M_r} \theta_k = 1$.

D. Objectives

1) Primary Objective: Deadline Satisfaction: Accept and provision a task by finding a participation set M_r and a split vector $\boldsymbol{\theta}_r$ that satisfy the constraints above with currently allocable resources. When the initial allocation is insufficient, employ adaptive mechanisms to expand participation while maintaining the deadline constraints.

2) Secondary Objectives: Efficiency and Stability: Subject to deadline satisfaction, the mechanism seeks to achieve:

- *Scalability:* Ensure that coordination and state-maintenance overhead grow with the size of the task participation set rather than with the platform size, thereby avoiding global bottlenecks in large-scale DCCS,
- *Computational efficiency:* Support lightweight latency-aware decision making with low online overhead, avoiding iterative search and exponential solution-space explosion on the critical path,
- *System stability:* Eliminate retraining requirements and avoid dynamic learning adaptation overhead that characterizes machine learning approaches, providing deterministic behavior independent of node population changes,
- *Robustness:* Handle insufficient initial allocations through adaptive mechanisms, and accommodate state staleness and transient contention through redundant strategies,

E. Formal Statement

Given t_r, $\{\overline{\mathbf{v}}_k, \mathrm{RTT}_k\}_k$, and $\breve{T}_r$, *find* participation set M_r, split vector $\boldsymbol{\theta}_r$, and resource allocations $\{\ddot{c}_k, \ddot{g}_k\}_{v_k \in M_r}$ such that attribute compatibility, resource constraints, and latency feasibility hold, while achieving the primary and secondary objectives above within the deadline-imposed time constraints.

V. Proposed Approach

DCCS must satisfy the resource attribute requirements of each task in large-scale distributed heterogeneous resource environments, ensure task completion within deadlines to guarantee service quality, while simultaneously addressing resource contention and task interference issues arising from the massive, diverse, and distributed nature of the underlying infrastructure. The formal objectives in Section IV therefore indicate that an effective provisioning mechanism should exhibit three key properties: per-task resource isolation and fine-grained management, precise scheduling and allocation guided by analytic latency budgets, and scalable coordination under large-scale distribution, heterogeneity, and bursty arrivals. To realize these properties, we propose TACS, which is elaborated in detail in this section.

A. TACS: Task-Level Adaptive Computing Slicing

The core idea of TACS is to provide fine-grained resource isolation and task-level resource management for large-scale distributed heterogeneous environments through shard-local autonomous decision-making. It aims to complete tasks within deadline budgets while enabling supply-demand matching

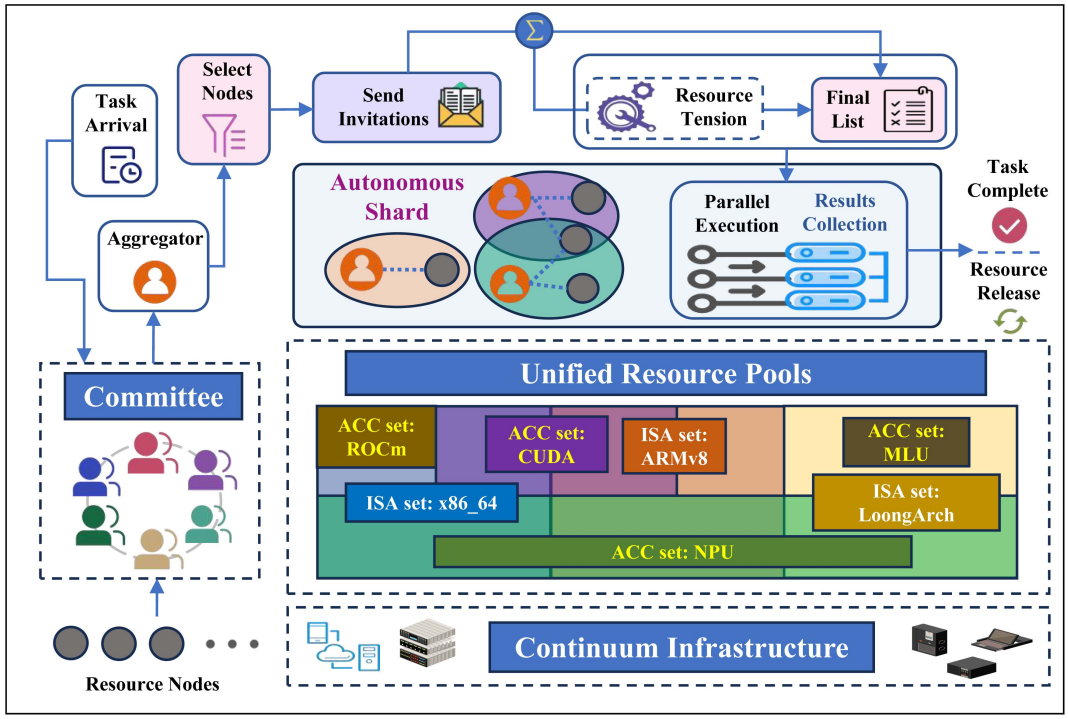


Fig. 2. Overall architecture and task-level control flow of TACS over integrated resource pools in DCCS.

adapted to DCCS characteristics. TACS aligns control boundaries with task boundaries: for every arriving task, the platform instantiates an ephemeral micro-slice and binds it to a shard whose membership consists exactly of that task's participants. The shard maintains all task-local state and makes all resource management decisions internally through autonomous governance, ensuring that coordination cost scales with the number of participants rather than with platform size.

Operationally, the system partitions resources on demand to form computing slices, with slice-related decisions made through parallel intra-shard autonomy. A single node may contribute to several shards when serving multiple tasks concurrently, and a single shard may span multiple nodes when a task aggregates capacity across the continuum. Each shard processes task-state tracking and related resource-management decisions driven by the task lifecycle (detailed in Section V-B). Because allocation and state evolution are confined to the shard, cross-task resource contention and state coupling are reduced, and per-task QoS can be honored precisely without resorting to global coordination. Fig. 2 summarizes the TACS framework over unified DCCS resource pools. It shows how heterogeneous continuum resources are integrated into unified resource pools, how task-specific shards are formed on demand, and how shard formation, shard-local autonomy, and the resource tension mechanism interact to support parallel execution, result aggregation, and resource release.

B. Task Lifecycle-Driven Shard Lifecycle Management

In TACS, the shard's lifetime is tied tightly to the lifetime of its task. A shard comes into being when a task t_r is admitted, evolves as participants are recruited and resources are committed, and is dismantled once results are verified and rewards are settled.

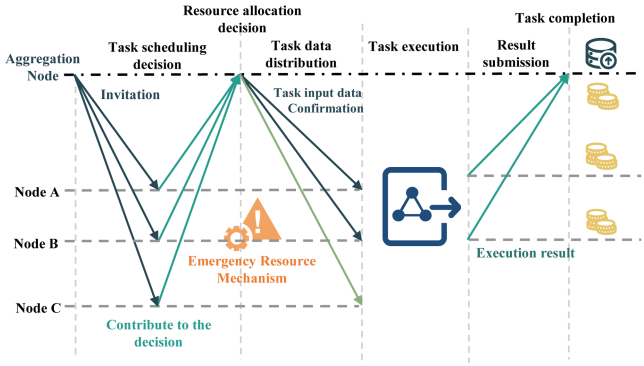


Fig. 3. Shard Lifecycle Management.

Fig. 3 illustrates the scheduling and resource-allocation workflow within the shard lifecycle. The entire shard lifecycle unfolds in six distinct phases following Algorithm 1:

Task Scheduling Decision Phase: Tasks are forwarded to qualified committee nodes with a measurable forwarding delay $\breve{T}_r$, after which a single committee node becomes the aggregator for t_r. The aggregator sends invitations to candidate nodes satisfying the attribute requirements $p_r \subseteq p_k$, initiating the participant recruitment process.

Resource Allocation Decision Phase: Candidate nodes respond to invitations and contribute decision information. Based on the responses, the aggregator executes the Task Scheduling and Resource Allocation algorithm (Section V-C). If standard scheduling succeeds, the aggregator determines the execution

set M_r and task split ratios $\{\theta_k\}$; if resource demands cannot be satisfied, the Resource Tension Mechanism (Section V-D) is triggered for dynamic recruitment.

Task Data Distribution Phase: With the issuance of confirmation messages, the shard is finalized. The aggregator distributes task input data to all nodes in the participation set M_r according to the computed split ratios θ_k, while sending execution confirmations. Messages that confirm participation serve a dual purpose: they both lock in the membership of M_r and convert the tentatively held resources into committed task-specific reservations that trigger the transfer of task inputs to the corresponding nodes. Consequently, for a confirmed participant, execution proceeds on the reserved share once its inputs arrive, rather than entering an additional uncontrolled node-local execution queue after confirmation.

Task Execution Phase: Each participating node executes its assigned task portion in parallel within the shard. Each node $v_k \in M_r$ processes input share $d_r^k = \theta_k \times d_r$ and compute share $f_r^k = \theta_k \times f_r$. The aggregator tracks progress and maintains shard-local state, ensuring that data movement and control converge in time.

Task Completion and Result Submission Phase: Nodes return execution results to the aggregator upon completion. The aggregator collects and aggregates partial results from all participating nodes.

Reward Distribution and Resource Release Phase: The aggregator validates the correctness of aggregated results. If the task meets its specification, all participants including the aggregator receive corresponding rewards according to $\dot{p}_r$. Finally, the slice is dismantled, all reserved resources are released back to the resource pool, and the shard lifecycle ends.

Because state and control remain confined to the shard, many shards can advance concurrently with limited cross-task interference.

For conciseness, the detailed phase-by-phase justification is deferred to Section II-C of the supplementary material. The resulting control-plane computational complexity of Algorithm 1 is

$$\mathcal{C}_{\text{Alg. 1}} = \begin{cases} O(N_s \log N_s), & \text{normal scheduling path,} \\ O((H_r+1)N_s \log N_s), & \text{resource tension path.} \end{cases} \tag{13}$$

C. Task Scheduling and Resource Allocation Decisions

A task admitted into the DCCS is routed based on network latency to the nearest committee nodes satisfying the attribute requirements p_r, with the forwarding process incurring a measurable delay $\breve{T}_r$. Other qualifying committee nodes, though not receiving the task directly, are notified of the task arrival to facilitate their data analysis and statistics tracking. From that moment, the selected committee node becomes the aggregator for the task, serving as the task's coordinator and decision maker: it interrogates admissible nodes about their instantaneous availability, estimates end-to-end latency contributions, and coordinates invitations so that the emerging shard can meet the deadline L_r with predictable quality of service. Rather than issuing broad, speculative requests, the aggregator advances through an orderly screen—estimate—invite process that reduces avoidable contention and shortens task startup latency. Fig. 3 illustrates the overall process architecture, while the remainder of this subsection details the mechanics and explains why each design choice matters for scale and reliability in DCCS.

Algorithm 1: Task Lifecycle Management.

Input: Task $t_r = \{d_r, f_r, a_r, b_r, p_r, L_r, \dot{p}_r\}$; node set V
Output: Verified task execution result for t_r
Forward t_r to qualified committee nodes with delay $\breve{T}_r$, and elect a qualified committee node as the aggregator for t_r
$\Omega_r \leftarrow$ Task Scheduling and Resource Allocation(t_r, V)
if Ω_r *is an execution result* **then**
    // The emergency path has been completed inside Alg. 2 via Alg. 3
    return Ω_r
$\{M_r, \theta_r\} \leftarrow \Omega_r$
Form the shard, finalize reservations for nodes in M_r, and distribute task inputs according to θ_r
Execute in parallel on nodes in M_r; the aggregator collects and aggregates results
Verify completion status and distribute rewards
Release reserved resources and dismantle the slice
return *Task execution result*

1) Evaluation Score Calculation: The aggregator $\mathbf{r}_r$ first ranks non-busy candidates v_k to expose those that are both fast and dependable. Latency is assessed with a conservative surrogate that captures the dominant components of the end-to-end path for t_r:

$$T_{\text{s}} = \frac{d_r}{u_a} + \frac{\varphi_r \times d_r}{u_k} + RTT_k + \frac{f_r \times (1-a_r)}{c_k} + \max\left(\frac{f_r \times a_r \times (1-b_r)}{c_k}, \frac{f_r \times a_r \times b_r}{g_k}\right), \tag{14}$$

where d_r and f_r denote the input size and total floating-point work of t_r, u_a is the aggregator's allocable uplink for t_r, u_k is the allocable uplink of v_k, φ_r reflects result selectivity, RTT_k is the measured round-trip time between the aggregator and v_k, and c_k, g_k are the currently available CPU and GPU compute resources of v_k. Screening uses the same d_r and f_r across candidates to ensure a fair, task-centric comparison. The reciprocal of T_{s} is normalized to y_k to represent the latency component. Beyond latency, the composite evaluation incorporates the reputation score $R_k \in [0, 100]$ as defined in the system model. Additionally, $\breve{p}_r^k$ represents the predicted position of task t_r in node v_k's invitation queue, determined by simulating the node's invitation acceptance mechanism, with the specific node admission strategy detailed in Section V-C2. The composite evaluation is

$$S(t_r, v_k) = \omega_c \times y_k + \frac{\omega_r \times R_k}{100} + \frac{\omega_p}{\breve{p}_r^k}, \tag{15}$$

where ω_c, ω_r, and ω_p control, respectively, the sensitivity to expected latency, to long-run reliability, and to the near-term rejection risk implied by the node's local queue. Sorting by $S(\cdot)$ produces a stable ordering that favors low-delay, high-reputation nodes while gently diversifying away from crowded queues.

2) Candidate List Generation: Starting from the score-ordered list, the aggregator sequentially constructs a candidate prefix and computes an expected task split ratio $\overline{\theta_k}$ for each node. Because the invitation-and-response process has not yet completed, the actual waiting time is still unavailable at this stage. Let N_s denote the total count of admissible nodes with nonzero allocable capacity that satisfy $p_r \subseteq p_k$. TACS therefore uses a conservative screening surrogate with $RTT_{\max} \triangleq \max(RTT_1, \ldots, RTT_{N_s})$ and $\beta_{r,k} \triangleq \frac{d_r}{u_a} + \frac{\varphi_r \times d_r}{u_k}$. The resulting candidate-stage estimator is

$$\overline{\theta_k} = \frac{[L_r - \breve{T}_r - RTT_{\max} - RTT_k] \times c_k}{f_r \times (1 - a_r \times b_r) + \beta_{r,k} \times c_k}. \tag{16}$$

This closed form directly inverts the remaining deadline budget into the expected share that node v_k can contribute under conservative planning. The detailed derivation from $\overline{T^e_{r,k}}$ to (16) is deferred to Section II-B of the supplementary material.

The aggregator continues down the list, accumulating $\overline{\theta_k}$ until the theoretical coverage $\sum_{i=1}^{E} \overline{\theta_i}$ first meets or exceeds one. If this sum cannot reach or exceed unity, the Resource Tension Mechanism mentioned in Section V-D will be triggered to ensure task completion.

The above reasoning is coupled with a measured over-provisioning step to offset stale state and transient contention. Due to the untimely updates of node state information, or the intensified resource competition caused by high concurrency of task execution, resources previously marked as available may already be occupied during actual allocation, leading to time consumption from repeated requests. To address this problem, TACS employs a minimum redundancy invitation factor μ_c that scales the number of invitations beyond the theoretical minimum required for task completion. If the top E candidates suffice in principle, the aggregator still invites the top W candidates in the score-ordered list to absorb short-term variability:

$$W = \min(N_s, \lceil E \times \max(\mu_c \times N_s, \lambda_t, 1) \rceil), \tag{17}$$

where λ_t is the recent arrival rate calculated by the aggregator through task arrival statistics as the average number of tasks with specific attribute requirements p_r arriving over the previous several time units (e.g., 3 time units), and $\mu_c \leq 1$ bounds the minimum redundancy. The first E invitations are flagged as formal to signal high likelihood of inclusion, while the remaining $W - E$ are flagged informal so that nodes can make informed decisions about pre-reservation without starving other workloads.

Upon receiving invitations, the core of node resource allocation decisions lies in comprehensively evaluating the information sent by the aggregator against their own state, running corresponding invitation processing strategies to rank the invitation list by preference, and then selecting invitations sequentially based on their available resources. For node resource allocation decisions, TACS can design different node decision mechanisms according to the system's optimization objectives. For example, considering node revenue, task invitations can be selected based on task profitability. Due to space limitations, we present a robustness-oriented example in Section II of the supplementary material.

3) Final List Generation: Once responses arrive, the aggregator waits at most $\max(RTT_1, \ldots, RTT_W)$, forms the positive-response set M_r^+, and reorders M_r^+ by the evaluation score for consistency. The candidate-stage quantity $\overline{\theta_k}$ is only a screening estimate; after the actual waiting time $T^w_{r,k}$ is observed, the aggregator recomputes the final allocation θ_k for each responding node. The remaining execution budget is

$$\overline{T^e_{r,k}} = L_r - T^w_{r,k} - \left(\frac{d_r^k}{u_a} + \frac{\varphi_r \times d_r^k}{u_k} + RTT_k \right). \tag{18}$$

Using $d_r^k = \theta_k \times d_r$ and the shorthand $\beta_{r,k}$ defined above, the per-node allocation becomes

$$\theta_k = \frac{[L_r - T^w_{r,k} - RTT_k] \times c_k}{f_r \times (1 - a_r \times b_r) + \beta_{r,k} \times c_k}. \tag{19}$$

If $L_r - T^w_{r,k} - RTT_k \leq 0$, the schedule is fragile, and TACS uses a tolerance-adjusted closed-form recomputation:

$$\theta_k = \frac{[\tilde{f} \times L_r - T^w_{r,k} - RTT_k] \times c_k}{f_r \times (1 - a_r \times b_r) + \beta_{r,k} \times c_k}. \tag{20}$$

The detailed rationale for this recomputation is deferred to Section II-B of the supplementary material. This tolerance-adjusted recomputation is retained only to preserve an explicit algebraic continuation of the sequential scan when the directly inverted numerator becomes nonpositive under the observed waiting time. The deadline-feasibility guarantee stated in Theorem 1 is deliberately restricted to the direct positive-residual branch $L_r - T^w_{r,k} - RTT_k > 0$.

The aggregator then scans the reordered set M_r^+ sequentially and admits nodes into the final participation set M_r until $\sum_{v_k \in M_r} \theta_k \geq 1$. For multi-node tasks, the last admitted node's share is trimmed to $1 - \sum_{v_i \in M_r \setminus \{v_Q\}} \theta_i$ to avoid over-allocation; for single-node tasks it is clamped to 1.

The last admitted node is trimmed and then right-sized. Under $\overline{T^e_{r,Q}}$, the minimum CPU and GPU reservations are

$$\tilde{c}_Q = \frac{f_r^Q \times (1 - a_r \times b_r)}{\overline{T^e_{r,Q}}}, \tag{21}$$

$$\tilde{g}_Q = \max\left(\frac{f_r^Q \times a_r \times b_r}{\overline{T^e_{r,Q}} - \frac{f_r^Q \times (1 - a_r)}{\tilde{c}_Q}}, \frac{b_r \times \tilde{c}_Q}{1 - b_r} \right), \tag{22}$$

where ε_c^Q and ε_g^Q denote the CPU and GPU redundancy factors for the last admitted node v_Q. The final committed reservations are

$$\ddot{c_Q} = \min\left(\varepsilon_c^Q \times \tilde{c}_Q, c_Q \right), \tag{23}$$

$$\ddot{g_Q} = \min \left(\varepsilon_g^Q \times \max\left(\frac{f_r^Q \times a_r \times b_r}{\overline{T^e_{r,Q}} - \frac{f_r^Q \times (1 - a_r)}{\ddot{c_Q}}}, \frac{b_r \times \ddot{c_Q}}{1 - b_r} \right), g_Q \right), \tag{24}$$

The detailed derivation, redundancy rationale, and boundary-case handling are deferred to Section II-B of the supplementary material.

Once the complete task scheduling and resource allocation decisions are obtained, the aggregator sends the relevant decisions as confirmation messages for node participation in the computing slice, along with the corresponding task data for each node, to the nodes on the final participation list M_r. Simultaneously, the aggregator notifies the responding but ultimately unselected nodes that the task decision has been completed, which enables them to remove the task from their invitation lists and unlock the corresponding tentative reservations, preventing interference with other tasks. The complete workflow appears in Algorithm 2, which spells out the four phases end to end and makes explicit the precise points where estimates are refined into commitments. In particular, when Phase 2 cannot form a theoretically sufficient candidate set, Algorithm 2 transfers control directly to Algorithm 3 with $M_r^{\text{incomplete}} = \emptyset$, after which the dynamic recruitment process is handled entirely by Algorithm 3. When Phase 4 ends with $\sum_{v_k \in M_r} \theta_k < 1$, it transfers control with the currently admitted set M_r as the inherited incomplete participant set.

For conciseness, the detailed phase-by-phase justification is deferred to Section II-C of the supplementary material. The resulting control-plane computational complexity of Algorithm 2 is

$$\mathcal{C}_{\text{Alg. 2}} = O(N_s \log N_s). \tag{25}$$

D. Resource Tension Mechanism

Across multiple shards, we adopt an asynchronous, shard-independent scheduling semantics: each shard progresses on its own timeline without any global synchronization barrier. When the aggregator cannot assemble a feasible candidate set or finalize the participation list for a shard in time, the shard does not stall; instead, it activates the *resource tension mechanism* as a controlled fallback. The mechanism starts from the partial scheduling state exported by Algorithm 2. Let $M_r^{\text{incomplete}}$ denote the inherited participant set at the moment the normal scheduler gives up. If Phase 2 fails before any feasible accepted set is formed, then $M_r^{\text{incomplete}} = \emptyset$. If Phase 4 ends after positive replies have been collected but before full coverage is reached, then $M_r^{\text{incomplete}} = M_r$, namely the accepted set accumulated by the normal scheduler. The aggregator promotes these nodes to the shard's *final participants* and starts their assigned work without delay, while continuing recruitment over the currently admissible nodes ranked by the same evaluation score used in normal scheduling. Each positive reply triggers an immediate recomputation of that node's share θ_k under scarcity and an update to the cumulative allocation. Because the cumulative allocation is monotonically nondecreasing, the loop terminates as soon as the sum first reaches or exceeds one, at which point the last entrant's share is trimmed to close precisely at one.

The allocation computation in tension mode follows the same closed-form formulas as the standard scheduling path. When a new node v_k enters during the tension mechanism, the aggregator computes θ_k using the same allocation formula from Section V-C. However, the redundancy factor calculations in (23) and (24) employ different parameters to account for the heightened uncertainty in resource-constrained scenarios. Specifically, the system uses $\hat{\varepsilon}$ as the perceived degree of resource shortage for redundancy factor computation, which governs how conservatively we size resource contributions under tension conditions. Additionally, the urgency contribution factor $\ddot{e}$ in tension mode replaces the standard factor $\dot{e}$ used in normal scheduling, ensuring that CPU and GPU reservations adequately absorb the compound uncertainty of congested links, competing requests, and potential node failures. If the cumulative allocation crosses one upon adding node v_k, the aggregator trims θ_k to $1 - \sum_{v_i \in M_r^{\text{final}} \setminus \{v_k\}} \theta_i$ and immediately recomputes the required resources for this trimmed share using (23) and (24) with the same tension-mode parameters, ensuring that the final node is properly sized rather than over-committed.

Algorithm 3 summarizes this procedure.

Algorithm 2: Task Scheduling and Resource Allocation.

Input: Task t_r; node set V
Output: Scheduling outcome for t_r
// Phase 1: Evaluation Score
foreach *non-busy node* $v_k \in V$ *with* $p_r \subseteq p_k$ **do**
  Compute T_s via Eq. (14), normalize its reciprocal to y_k, and compute $S(t_r, v_k)$ via Eq. (15)
end
Sort all candidates by descending score
// Phase 2: Candidate List Generation
Predefine E as the minimum theoretically sufficient candidate count, W as the redundancy-aware invitation count, and Θ_{tot} as the accumulated expected share; set $E \leftarrow 0$ and $\Theta_{\text{tot}} \leftarrow 0$
repeat
  Select the next node v_k, estimate $\overline{\theta_k}$ via Eq. (16), update $\Theta_{\text{tot}} \leftarrow \Theta_{\text{tot}} + \overline{\theta_k}$, and set $E \leftarrow E + 1$
until $\Theta_{\text{tot}} \geq 1$ *or all nodes have been considered*
if $\Theta_{\text{tot}} < 1$ **then**
  $M_r^{\text{incomplete}} \leftarrow \emptyset$
  return *Resource Tension Mechanism*$(t_r, V, M_r^{incomplete})$
// Phase 3: Invitation and Response
Compute W via Eq. (17); send formal invites to the top E candidates and informal invites to the next $W - E$; wait $\max(RTT_1, \ldots, RTT_W)$; form M_r^+; reorder M_r^+ by descending score
// Phase 4: Final Assignment
$Q \leftarrow 0$; $\Theta_{\text{fin}} \leftarrow 0$; $M_r \leftarrow \emptyset$
foreach *node* $v_k \in M_r^+$ **do**
  Compute θ_k via Eq. (19)
  if $L_r - T_{r,k}^w - RTT_k \leq 0$ **then**
    Recompute θ_k via Eq. (20)
  $M_r \leftarrow M_r \cup \{v_k\}$; $\Theta_{\text{fin}} \leftarrow \Theta_{\text{fin}} + \theta_k$; $Q \leftarrow Q + 1$
  if $\Theta_{\text{fin}} \geq 1$ **then**
    Trim θ_Q if needed, compute $\ddot{c}_Q$ and $\ddot{g}_Q$ via Eq. (23) and Eq. (24), and break
  end
end
if $\Theta_{\text{fin}} \geq 1$ **then**
  $\theta_r \leftarrow \{\theta_k\}_{v_k \in M_r}$; send confirmations that finalize reservations and task data to nodes in M_r
  notify nodes in $M_r^+ \setminus M_r$ to release tentative reservations; **return** $\{M_r, \theta_r\}$
$M_r^{\text{incomplete}} \leftarrow M_r$
return *Resource Tension Mechanism*$(t_r, V, M_r^{incomplete})$

Algorithm 3: Resource Tension Mechanism.

Input: Task t_r; available node set V; incomplete participant set $M_r^{\text{incomplete}}$
Output: Task execution result for t_r under resource tension

$M_r^{\text{final}} \leftarrow M_r^{\text{incomplete}}$
Compute current θ_k for all $v_k \in M_r^{\text{final}}$ and set $\Theta_{\text{cum}} \leftarrow \sum_{v_k \in M_r^{\text{final}}} \theta_k$
Send confirmations and stream inputs to nodes in M_r^{final}
Start execution on nodes in M_r^{final}
// Continuous dynamic recruitment
while $\Theta_{\text{cum}} < 1$ **do**
  $V^{\text{rem}} \leftarrow \{v_k \in V \setminus M_r^{\text{final}} \mid p_r \subseteq p_k\}$
  Sort V^{rem} by descending evaluation score
  Form the current invitation batch from the top-ranked admissible nodes and send invitations
  foreach v_k *that replies* accept **do**
    // Admit and allocate inside the tension mechanism
    Compute θ_k under shortage degree $\hat{\varepsilon}$
    $M_r^{\text{final}} \leftarrow M_r^{\text{final}} \cup \{v_k\}$
    $\Theta_{\text{cum}} \leftarrow \Theta_{\text{cum}} + \theta_k$
    if $\Theta_{\text{cum}} \geq 1$ **then**
      Apply the last-node trimming rule to θ_k
      Recompute required resources via Eq. (23) and Eq. (24) with the tension-mode parameters
      Send confirmation and stream inputs to v_k
      break
    else
      Send confirmation and stream inputs to v_k
      Start execution on v_k in parallel with continued recruitment
    end
  end
end
Wait for results from all participants, aggregate at the aggregator, distribute rewards, and release reserved resources
return *Task execution result*

For conciseness, the detailed phase-by-phase justification is deferred to Section II-C of the supplementary material. The resulting control-plane computational complexity of Algorithm 3 is

$$\mathcal{C}_{\text{Alg. 3}} = O(H_r N_s \log N_s)\,. \tag{26}$$

Viewed end to end, the resource tension mechanism complements the baseline scheduler with a principled safety net. It also explains how TACS remains responsive when demand spikes or attribute constraints restrict admissible supply. By turning unavoidable scarcity into structured, monotonic progress and by overlapping negotiation with execution, the mechanism maintains deadline awareness without sacrificing stability.

E. Feasibility Guarantee for Closed-Form Allocation

In the final assignment stage, non-last participants are handled in two algebraic cases. When $L_r - T_{r,k}^w - RTT_k > 0$, TACS uses the direct closed-form rule in (19). When this residual term is nonpositive, TACS invokes the tolerance-adjusted recomputation in (20) only to preserve the sequential accumulation logic. The feasibility guarantee below is therefore stated for the direct positive-residual branch together with the trimmed last-node right-sizing step.

Theorem 1 (Feasibility Guarantee for Closed-Form Allocation): For any task t_r, let M_r denote the final participation set returned by the normal scheduling path of Algorithm 2. Assume that $L_r - T_{r,k}^w - RTT_k > 0$ holds for all $v_k \in M_r$. Under the condition that the per-task reserved resources used for latency evaluation are realizable during execution, the final allocation produced by TACS by computing shares for non-last nodes through the direct closed-form rule in (19) and performing share trimming and right-sizing for the last node via (23) and (24) satisfies the system-model node-level latency feasibility constraint in (11), and consequently the conservative end-to-end latency satisfies $T_r \leq L_r$.

For conciseness, only the theorem statement and a proof sketch are included in the main text. The complete proof is provided in Section II-C of the supplementary material.

Proof: Under the stated assumption, every non-last node in M_r is assigned through the direct closed-form rule in (19), which is obtained by solving the node-level latency feasibility constraint under the observed waiting time $T_{r,k}^w$ and the same planning parameters used by TACS in final assignment. Substituting the induced split $d_r^k = \theta_k d_r$ and $f_r^k = \theta_k f_r$ back into that constraint therefore makes the inequality hold by construction for every non-last participant on the guaranteed branch. For the last node, the final share is fixed by trimming, and (23) and (24) right-size the CPU and GPU reservations for that fixed workload under the remaining execution budget. Under the assumption that the resource reservations used in this latency evaluation are realizable during execution, the last node satisfies the same node-level feasibility inequality. Hence (11) holds for all $v_k \in M_r$, and taking the maximum over M_r yields $T_r \leq L_r$. The tolerance-adjusted recomputation in (20) is outside the scope of this theorem and is used only to preserve an explicit algebraic continuation of the scan when a scanned node has nonpositive residual budget. □

VI. Performance Evaluation

This section turns the design commitments of TACS into measurable evidence. We evaluate whether per-task microslicing, shard-local control, and the resource tension mechanism together deliver deadline-aware provisioning in a DCCS without sacrificing scalability or stability. To validate TACS effectiveness in DCCS environments, we specifically evaluate the following key questions: whether TACS can achieve precise supply-demand matching under strict latency budgets L_r while maintaining scalability when confronting the challenges posed by DCCS's arrival burstiness, heterogeneity, and large-scale characteristics. To make these questions actionable, we adopt workloads that vary input sizes d_r, compute footprints f_r, attribute constraints p_r, and CPU/GPU mix (c_k, g_k), and we sweep network conditions through realistic round-trip times RTT_k and bandwidth configurations consistent with the system model in Section III. All symbols and constraints follow the definitions introduced earlier, including the scoring weights $(\omega_c, \omega_r, \omega_p)$, the tolerance factor $\tilde{f}$, and the redundancy controls in (23) and

TABLE II
EXPERIMENTAL SETTINGS SUMMARY

Item	Summary
Number of nodes	Z swept from 100 to 1000.
Arrival rates	2.5–20 tasks/s; 100 tasks/s in the 1000-node heterogeneity test.
Load levels	Offered load in $[40{,}000, 90{,}000]$ GFLOP/s; ablation load pressure in $[36{,}537, 92{,}576]$ GFLOP/s.
Baselines	Dual-Timescales Optimization, Slicing Window Adaptation, GREET-based, and Static Slice.

(24). Regarding the impact of each tunable parameter in the algorithm on its performance, we conducted parameter experiments. Due to space limitations, the results and related analysis are presented in Section IV of the supplementary material.

A. Simulation Setup

Guided by the DCCS reference architecture of Dustdar et al. [1], we emulate a large-scale continuum that grows from 100 to 1000 heterogeneous resources. The pool blends edge devices, cloud virtual machines (VMs), and serverless configurations, which we refer to uniformly as *computing nodes*. Each node exposes the attributes defined in Section III, including its CPU and GPU capacities (c_k, g_k), access and peer bandwidths, round-trip time RTT_k, and a vector of special attributes p_k used to match tasks with eligible executors. To stress scalability, we sweep the cluster size while keeping the workload mix constant; to examine heterogeneity, we vary the composition of edge, VM, and serverless instances and their (c_k, g_k) budget according to the ranges defined in the system model.

1) Workload and Arrival Process: Tasks arrive as a memoryless process with an exponential inter-arrival distribution parameterized by the mean rate configured for each experiment. Upon arrival, a task advertises its special-attribute requirement p_r and is matched only against nodes whose p_k satisfy that constraint. The remaining task fields follow the notation introduced earlier: input size d_r, compute footprint f_r, deadline L_r, and the decomposition factors (a_r, b_r) that split preprocessing, parallel compute, and aggregation in accordance with Section III. The key settings of scale, load, and arrival intensity used in the experiments are summarized in Table II.

2) Baselines and Adaptation: To position TACS against representative designs, we compare it with four baselines. The first three are adapted from recent advanced systems that all strive for dynamic and efficient resource management through network slicing, employing distinctly different architectures and decision making to balance service guarantees of individual slices with overall network efficiency: **Dual-Timescales Optimization (2024)** employing machine learning for dynamic scheduling, **Slicing Window Adaptation (2024)** employing dynamic window adjustment, and **GREET-based method (2023)** employing game-theoretic arbitration for fair competition [17], [18], [30]. We port these methods to our DCCS by aligning their resource and timing models with Section III and by interfacing their schedulers with the same node inventory, attribute matching, and communication layer. The fourth baseline is a traditional static slicing strategy that fixes slice proportions per task type. Table III summarizes for each method the correspondence between slices and tasks, the slice-adjustment policy, the allocation mechanism, and the scheduling logic. We retain the original decision granularity and control flow of each baseline and apply only the minimal adaptations that are necessary for compatibility and feasibility while remaining faithful to the corresponding paper's design intent, thereby ensuring a fair comparison.

For the baselines adapted from prior work, we preserve the original method design and optimization logic as much as possible, and make only the minimal modifications required for compatibility with our DCCS simulator and for tractable execution in the large-scale setting. For Dual-Timescales Optimization, the original fine-timescale scheduler uses an explicit high-dimensional binary action whose dimension grows with the node population and does not converge reliably in the thousand-node regime. We therefore modify only this action parameterization, replacing it with a scalable continuous preference-vector representation followed by feasibility masking and decision mapping. Apart from this necessary change, the baseline remains faithful to the original method structure and policy logic. Slicing Window Adaptation performs re-slicing at slicing-window boundaries and executes DDQN scheduling in each $1\,\mathrm{s}$ slot within the active window. The GREET-based baseline is updated online on each task arrival, honoring guaranteed shares first and then allocating residual capacity through competitive sharing.

To ensure reproducibility, Table IV reports the key hyperparameters, update cadence, and convergence criteria of all baselines. Full implementation details are provided in Section III of the supplementary material.

3) Fairness and Configuration Discipline: All methods run over the same node set, the same link-level delays and bandwidths, and the same task arrival trace. Unless otherwise stated, we evaluate each method under its default configuration tuned for stable operation; the parameter sensitivity results are reported in the supplementary material. For a controlled comparison, all methods share the same attribute information and matching rule for $\{p_r, p_k\}$, and they use the same resource and network accounting, including (c_k, g_k) and RTT_k.

Statistical reporting and stability visualization: Results for TACS are mean $\pm$ std over 10 runs; baselines are evaluated with fixed converged policies under the same traces; due to space constraints, the detailed evaluation protocol and reporting rationale are provided in Section III-C of the supplementary material.

B. Ablation Study

To disentangle how each control mechanism contributes to TACS performance under resource pressure, we conduct an ablation study on a 100-node DCCS under increasing system load. We remove, one at a time, three core mechanisms: the *Resource Tension Mechanism* that continues recruiting participants when cumulative allocation risks falling short, *Redundant Invitations* that hedge against stale availability information during

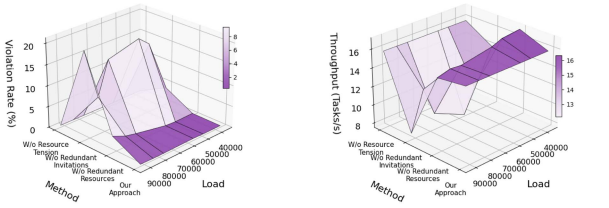
TABLE III
BASELINE METHODS COMPARISON

Method	Slice Unit	Slice Adjustment	Resource Allocation	Task Scheduling
Dual-Timescales Optimization	Task type	Large-timescale AEF optimization	Slice-share-based	TD3
Slicing Window Adaptation	Delay class	Dynamic window adjustment	Slice-share-based	DDQN
GREET-based	Task type	Guaranteed share + flexible competition	Slice-share-based	Urgency-aware
Static Slice	Task type	Fixed slice shares	Slice-share-based	Load-aware

TABLE IV
CONDENSED KEY SETTINGS OF THE BASELINE METHODS

Method	Key Settings	Update Cadence	Stop Rule
Dual-Timescales Optimization	TD3: MLP $(512, 256)$, lr 3×10^{-4}, bs 256, buffer 10^5, $\gamma = 0.95$, $\tau = 0.005$, delayed update $d = 2$, smoothing $(0.2, 0.5)$; AEF: pop 30, iter 20, $K_0 = 500$, $\alpha = 0.3$.	Coarse slice adjustment + fine TD3	Stop when moving-average return gain falls below threshold over consecutive windows and DVR remains below threshold, or when the training budget is exhausted.
Slicing Window Adaptation	DDQN: $\gamma = 0.98$, Adam 3×10^{-4}, $\epsilon: 1.0\to0.02$ (decay 0.998), buffer 10^4, bs 64, target update 30; re-slicing: $\kappa_1 = \kappa_2 = 0.01$, iter 1000, step $0.5\to0.001$.	Re-slice per window; DDQN every 1 s	After ≥ 1000 epochs, stop when evaluated cumulative reward plateaus; otherwise run to 5000 epochs.
GREET-based	α-fair utility ($\alpha = 2$); CPU quota 0.5×0.33; GPU quota $0.5\times\{0.40/0.25/0.35\}$; BW quota $0.7\times\{0.30/0.50/0.30\}$; high-BW node $\times1.1$; caps $0.5/0.5/0.85$; $e_v = 0.33$; min task weight 0.05; min alloc. 0.10.	Online per arrival	Theorem-7 fixed-point contraction condition.
Static Slice	Fixed slice shares by task type.	No online re-slicing	Not applicable.

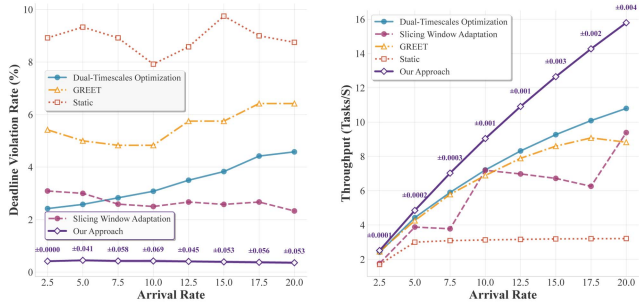
Note: Full implementation details are provided in Section III of the supplementary material.



(a) Deadline violation rate (DVR): violated tasks / total tasks (%)

(b) System throughput: completed tasks / second (tasks/s)

Fig. 4. Component Impact Assessment under Increasing Load.



(a) Deadline violation rate (DVR): violated tasks / total tasks (%)

(b) System throughput: completed tasks / second (tasks/s)

Fig. 5. Supply–demand matching under varying arrival intensity.

the invitation phase, and *Redundant Resources* that introduce safety margins in the last admitted node.

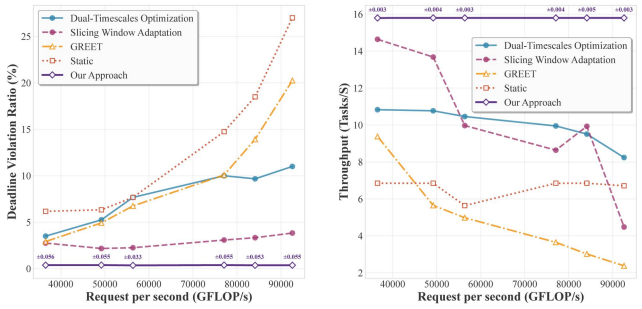
Fig. 4 shows that complete TACS keeps the deadline violation rate within 0.33%–0.42% while sustaining throughput near 15.80 tasks/s across the tested load range. Once any mechanism is removed, the gap becomes visible under load: removing the *Resource Tension Mechanism* raises violations to 0.42%–0.50%, removing *Redundant Invitations* causes the most severe degradation with violations rising to 20.83% and throughput falling to 8.22 tasks/s, and removing *Redundant Resources* increases violations to 3.87% under high pressure. Due to space constraints, the detailed component-wise interpretation, including the quantified degradation of each variant and the explanation of their complementary roles, is provided in Section III-F of the supplementary material.

C. Supply–Demand Matching Performance

Anchored in prior studies on deadline-aware orchestration and large-scale resource control [36], [37], we evaluate supply–demand alignment with two service-centric indicators: the *deadline violation rate*, which tests whether provisioned resources and their orchestration satisfy each task's latency budget, and *system throughput*, which reports how effectively the substrate converts available capacity into completed work. Read together, these curves tell whether a method preserves reliability while pushing the capacity frontier in DCCS.

To validate TACS's supply-demand matching capabilities against DCCS's *arrival burstiness* characteristic, we design arrival-rate sweep and load ramp experiments to test system performance in maintaining resource supply and task demand alignment under varying task arrival intensities and system load conditions.

1) Arrival-Rate Sweep: Fig. 5(a) and Fig. 5(b) summarize the behavior as the task arrival rate increases. Specifically, TACS attains a mean violation rate between 0.36% and 0.45%, and a mean throughput ranging from 2.51 to 15.79 tasks/s. In contrast, Dual-Timescales Optimization shows violations between 2.42% and 4.58% with throughput from 2.41 to 10.8 tasks/s, Slicing Window Adaptation exhibits violations from 2.33% to 3.09%



(a) Deadline violation rate (DVR): violated tasks / total tasks (%)

(b) System throughput: completed tasks / second (tasks/s)

Fig. 6. Supply–demand matching under increasing system load.

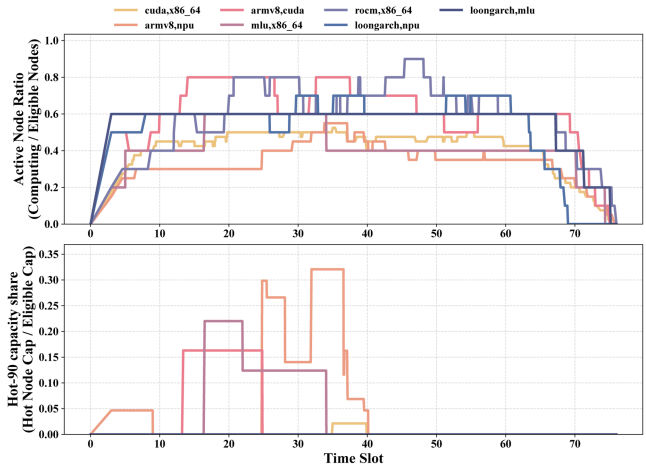


Fig. 7. Micro-level load balancing characterization under high demand: active-node ratio and Hot-90 capacity share over time.

with throughput spanning 1.76 to 9.39 tasks/s, GREET demonstrates violations ranging from 5% to 6.42% with throughput between 2.45 and 8.83 tasks/s, while Static slicing produces violations from 8.75% to 9.75% with throughput limited to 1.7 to 3.21 tasks/s. This gap is consistent with the design: TACS computes, at the level of an individual task, the exact CPU and GPU contributions needed to close its end-to-end latency, then hardens those budgets inside a shard so that interference from other jobs is structurally contained. The slicing-window adaptation baseline is the closest competitor because it shortens decision windows under pressure; however, it still plans at the slice type rather than per task, which leaves residual coupling that surfaces as sporadic misses under bursty contention. The GREET-based method illustrates a different limitation. Once guaranteed shares saturate at high intensity, the pool of excess resources available for weight-based competition shrinks sharply. Even though urgent tasks carry higher weights, the residual pool becomes too small to restore feasibility, and violation rates rise. A static slice is least adaptable, as fixed proportions cannot track a time-varying workload mix. Throughput grows with arrival intensity for TACS, dual-timescale optimization, and GREET, yet TACS leads in the tested region because it can assemble fractional contributions from multiple nodes into a feasible shard without waiting for a single node to free a monolithic block, which improves the match between offered supply and instantaneous demand.

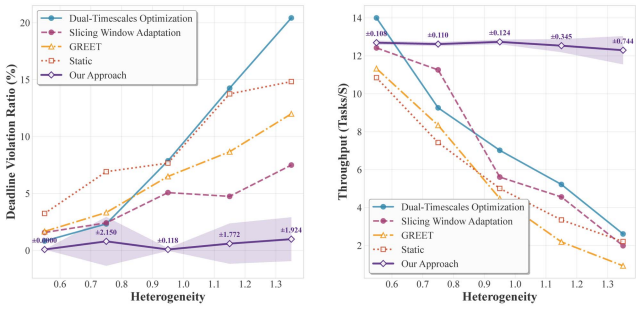
2) Load Ramp: Under increasing offered load, the contrast becomes clearer in Fig. 6(a) and Fig. 6(b). TACS keeps the mean deadline violation rate within a narrow $0.35\%$to 0.38% range and sustains a nearly flat mean throughput of 15.79 to 15.8 tasks/s. In contrast, baselines exhibit much wider performance variations: Dual-Timescales shows 3.5% to 27% violations with 6.71 to 15.8 tasks/s throughput, Slicing Window demonstrates 2.17% to 20.83% violations with 4.48 to 15.8 tasks/s throughput, GREET exhibits 4.92% to 20.25% violations with 2.38 to 9.38 tasks/s throughput, and Static slicing shows 6.33% to 27% violations with 6.71 to 6.85 tasks/s throughput. TACS's stability primarily benefits from two synergistic mechanisms. First, the shard lifecycle management mechanism dynamically allocates and releases resources as tasks progress through preprocessing, parallel compute, and aggregation phases, thereby promptly reclaiming transient idle resources. Second, when the initial participant set cannot provide sufficient resources, the resource tension mechanism enables continued recruitment of additional nodes while confirmed participants begin task execution, converting potential deadline violations into on-time completions without blocking critical execution paths. GREET saturates its guaranteed shares at high load and its proportional fairness on the residual pool disperses scarce capacity across many contenders; many tasks receive partial help but not enough to satisfy end-to-end budgets, which depresses throughput and raises misses. The dual-timescale method protects throughput better than static slicing through TD3 micro-decisions, yet it lacks the per-task isolation and fine-grained composition that keep TACS inside a tight reliability envelope. The slicing-window method adapts quickly by shortening windows, which helps it hold capacity, but without per-task budgeting it cannot fully neutralize interference when several latency-sensitive paths collide.

3) Micro-Level Load Balancing Patterns and Hotspot Coverage Assessment: To further examine the micro-level operating pattern of TACS under high load, this subsection follows the load-ramping experiment and selects the load level of $92{,}576\ \mathrm{GFLOP/s}$ as the observation point. With reference to prior studies [38], [39], we extract two time-varying diagnostics from node-level operating trajectories at this load level. The active-node ratio denotes the proportion of eligible nodes that actually undertake task load, reflecting the spreading degree of load over available nodes. The Hot-90 capacity share denotes the share of available computing capacity covered by nodes whose compute utilization exceeds 90%, where node utilization is measured by the capacity-weighted aggregation of CPU and GPU occupancy on the same node, and is used to characterize the coverage of high-utilization hotspots.

As shown in Fig. 7, after the initial transient, the active-node ratio remains at a medium-to-high level for most of the run, whereas the Hot-90 capacity share stays close to zero and rises only briefly in a few intervals. This suggests that under high demand, TACS tends to spread pressure across the eligible node set rather than sustain persistent hotspots on a small subset of

nodes. Due to space constraints, the exact metric definitions, the extraction procedure, and further analysis are provided in Section III-G of the supplementary material.

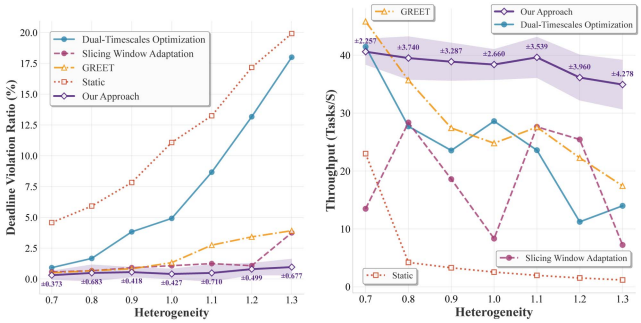
4) Interpretation and Takeaways: The curves support a consistent interpretation. Facing DCCS's *arrival burstiness* challenge, TACS achieves supply-demand alignment at the granularity where work actually happens through task-level micro-slicing. By computing required contributions from each node for the job at hand, protecting those allocations inside a shard that moves with the task, and admitting fractional contributions without waiting for whole-node availability, TACS effectively mitigates the two classic pitfalls of distributed provisioning: overcommitment that breaks deadlines and under-utilization that wastes capacity. The result is a system that maintains deadline miss ratios near zero while sustaining the throughput ceiling set by the substrate under varying arrival intensities and loads. Further node-level occupancy patterns indicate that TACS is able to distribute the load more sufficiently within the eligible node set rather than sustained pressurization on a small number of nodes. This behavior is consistent with the control logic of TACS: candidate nodes are ranked by the comprehensive evaluation score, so that nodes with higher short-term admission risk or those in congested queues naturally move backward in the ranking; and because the resource shortfall is filled by expanding the participant set and accumulating coverage, the system in the high-pressure stage is less likely to form a structural pattern of "some nodes being overloaded for a long time while other nodes are idle".



(a) Deadline violation rate (DVR): violated tasks / total tasks (%)

(b) System throughput: completed tasks / second (tasks/s)

Fig. 8. Impact of resource heterogeneity in a 100-node DCCS.



(a) Deadline violation rate (DVR): violated tasks / total tasks (%)

(b) System throughput: completed tasks / second (tasks/s)

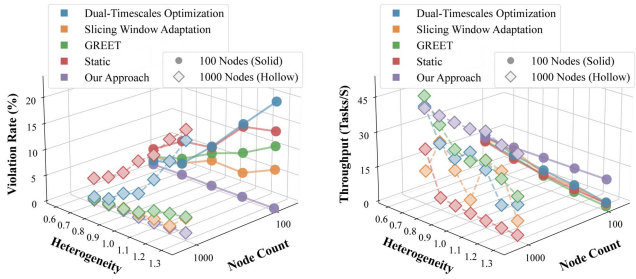
Fig. 9. Heterogeneity impact at scale (1000-node DCCS).

D. Scalability Performance

To validate TACS's scalability performance against DCCS's *heterogeneity* and *scale* characteristics, we stress TACS along two critical dimensions: increasing resource heterogeneity and growing substrate size. Heterogeneity is controlled by a parameter that sets the standard deviation of a log-normal distribution governing node capacities; moderate values (0.4–0.8) emulate mainstream cloud or enterprise clusters with multi-generation hardware under managed variance, while higher values (0.8–1.5) reflect real DCCS and edge settings where device classes and vendors mix freely. This knob directly widens the spread of available $\{c_k, g_k\}$ across nodes, which challenges any policy that assumes uniformity.

1) Heterogeneity Sweep At 100 Nodes: Fig. 8(a) and Fig. 8(b) report deadline violation and throughput as the heterogeneity parameter grows from 0.55 to 1.35. Across heterogeneity levels, TACS achieves a near-zero violation rate on average, with the mean ranging from 0.08% to 0.99%, while maintaining a consistently stable mean throughput between 12.30 and 12.74 tasks/s (a variation of approximately 3.5%). This stability follows directly from per-task slicing and the closed-form allocation in Section V: each shard asks only for the CPU and GPU contributions it needs from each participant, so differences in raw node capability do not propagate as timing uncertainty. Baselines show substantial performance degradation as heterogeneity increases: Dual-Timescales Optimization exhibits violations spanning 0.83% to 20.42% with throughput ranging from 2.61 to 14 tasks/s, Slicing Window Adaptation demonstrates violations from 1.58% to 7.5% with throughput between 1.99 and 12.42 tasks/s, GREET shows violations ranging from 1.67% to 12% with throughput spanning 0.93 to 11.33 tasks/s, while Static slicing exhibits violations from 3.25% to 14.83% with throughput between 2.21 and 10.85 tasks/s. Dual-Timescales Optimization's degradation stems from its reliance on TD3 with a continuous action space; when capacity dispersion widens, identical action magnitudes map to very different execution effects across nodes, which complicates function approximation and yields miscalibrated allocations. By contrast, Slicing Window Adaptation employs DDQN over a discrete action space and focuses on node selection; this avoids modeling continuous magnitudes under skew and therefore degrades more gracefully by preferentially picking stronger nodes when they are present. GREET's guaranteed shares remain helpful at moderate skew, but its residual pool for weight-based competition thins as dispersion increases, limiting its ability to restore feasibility under bursts. Static slicing, as expected, cannot reshape fixed proportions to track widening variance.

2) Scaling Out to 1000 Nodes: To probe scalability, we increase the platform to 1000 nodes while keeping task intensity fixed. We set the arrival rate to 100 tasks/s to match the larger substrate and vary heterogeneity from 0.7 to 1.3. Fig. 9(a) and Fig. VI-D2 show that TACS maintains a mean



(a) Deadline violation rate (DVR): violated tasks / total tasks (%)

(b) System throughput: completed tasks / second (tasks/s)

Fig. 10. Cross-scale performance comparison under varying heterogeneity (100 vs. 1000 nodes).

violation rate within 0.31%–0.97% while sustaining a mean throughput of 34.93–40.61 tasks/s across the full range. At low heterogeneity (0.7–0.8), several baselines are competitive: Dual-Timescales posts 0.92%—18% violations with 11.23—41.47 tasks/s; GREET achieves 0.50%—3.92% with 17.44—45.83 tasks/s; and Slicing Window Adaptation records 0.58%—3.75% with 7.23—28.38 tasks/s. As dispersion intensifies (0.9–1.3), the spreads widen. Dual-Timescales' violations expand to 3.83%–18% with throughput 11.23–28.61 tasks/s, reflecting the difficulty of stabilizing continuous control under skew. Slicing Window Adaptation ranges from 0.92% to 3.75% with 7.23–27.63 tasks/s; its discrete choices remain smoother than continuous magnitudes but still oscillate as the candidate set broadens. GREET ranges from 0.83% to 3.92% with 17.44–27.58 tasks/s, limited by fair sharing on an over-subscribed residual pool. Static slicing remains the least adaptable, with 4.58%–19.92% violations and 1.19–23.01 tasks/s throughput.

3) Cross-Scale View and Interpretation: Finally, Fig. 10(a) and Fig. 10(b) juxtapose the 100-node and 1000-node regimes. Throughput fluctuations become more pronounced for baselines as scale magnifies the heterogeneity of choices. DDQN-based Slicing Window Adaptation exhibits selection oscillations when confronted with a much larger and more skewed candidate set, which explains its irregular ups and downs across heterogeneity levels. TD3-based Dual-Timescales produces comparatively smoother magnitudes, yet the function-approximation burden of mapping continuous actions to highly nonuniform execution effects leads to under- or over-shoot and, ultimately, wider violation bands. TACS remains the most stable because each shard reasons locally about its participants' concrete capabilities, composes fractional contributions to meet a single task's budget, and is therefore insensitive to the global diversity that destabilizes system-wide learners. In other words, TACS scales by construction: decentralization removes the central planner's bottleneck, and per-task budgeting prevents heterogeneity from leaking into timing variance.

4) Topology-Aware Shard Coordination: To evaluate shard coordination under different physical infrastructures, we vary only the network topology while keeping task generation, node population, offered load, and all algorithm parameters fixed.

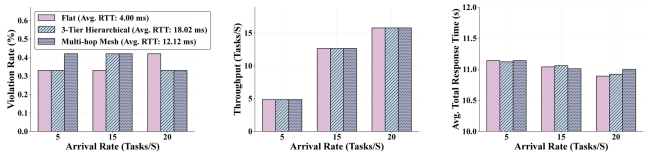


Fig. 11. Topology sensitivity of shard coordination under representative continuum structures.

Due to space constraints, the physical-graph construction, the shortest-path-based RTT derivation, and the additional topology sensitivity details are presented in Section III-E of the supplementary material.

The Flat topology approximates a low-latency local area network (LAN) or single-datacenter setting with near-uniform one-hop reachability. The 3-Tier Hierarchical topology captures the cloud–fog–edge organization and enforces cross-domain relaying through a cloud aggregation layer. The Multi-hop Mesh topology models decentralized collaboration on a scale-free multi-hop interconnection.

Fig. 11 reports deadline violation rate, throughput, and average end-to-end response time under three arrival-rate settings. The mean RTT increases from 4.00 ms in Flat to 12.12 ms in Multi-hop Mesh and 18.02 ms in 3-Tier Hierarchical, which reflects the additional coordination distance introduced by multi-hop relaying and cross-domain traversal. Even with this RTT inflation, the throughput curves remain nearly overlapping across topologies, the deadline violation rate stays within 0.33%–0.42%, and the average end-to-end response time differs by only about 0.1 s. These results show that TACS maintains effective shard coordination under the topology-level RTT inflation covered by the evaluated continuum structures: latency-aware candidate scoring absorbs the changed coordination inputs, and shard-local governance keeps feasibility and scalability stable, as reflected by consistently low DVR, overlapping throughput curves, and nearly unchanged response time across the tested structures.

VII. Conclusion and Future Work

This paper addresses the DCCS supply–demand matching and scalability challenges arising from the compounding effects of scale, heterogeneity, and arrival burstiness by introducing *TACS*. TACS leverages task-level micro-slicing, analytic supply-demand matching guided by latency constraints, and autonomous shard governance to achieve precise resource provisioning and effective scalability.

Large-scale simulations show that TACS maintains deadline violation rates below 1% and stable throughput across varied scales and operating conditions, while avoiding retraining and substantially outperforming existing methods. To the best of our knowledge, TACS is the first framework to achieve task-level resource provisioning in DCCS through distributed shard governance with ephemeral micro-slices.

We also note the main limitations and corresponding next steps.

- *Online-estimation fidelity and control-plane overhead:* Biases in the inputs used for closed-form invitation and resource-allocation decisions can affect robustness. In addition, although this paper quantifies protocol-induced signaling volume on the control plane, the practical wall-clock impact of invitation broadcasting, shard confirmation, and dynamic recruitment under realistic network dynamics and churn remains to be validated. *Planned work:* add adaptive calibration with confidence-aware scoring and incorporate batching and rate control into the invitation window W of (17).
- *Extension to precedence-constrained DAG tasks:* The current task model targets divisible workloads that admit proportional parallel partitioning. *Planned work:* extend TACS to stage-level micro-slicing, treating each DAG stage as a task instance and coordinating dependencies at stage boundaries while reusing shard-local scheduling and closed-form allocation within each stage.
- *Aggregator trust and fault tolerance:* Although TACS favors reputable, well-connected nodes when selecting shard organizers, malicious or faulty aggregators can still misroute or delay tasks. *Planned work:* add cross-shard consensus and committee signatures.
- *Validation beyond simulation:* The current evaluation remains simulation-based with tailored workloads and does not yet establish external validity on real traces or on a running prototype. *Planned work:* perform trace-driven replay using edge–cloud task logs and IoT data streams, and build an edge–cloud testbed prototype to quantify end-to-end deadline satisfaction and control-plane overhead under realistic conditions.

Together, these directions broaden task generality, strengthen robustness, and improve external validity without changing the core design.

REFERENCES

[1] S. Dustdar, V. C. Pujol, and P. K. Donta, "On distributed computing continuum systems," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 4, pp. 4092–4105, Apr. 2023.

[2] I. Cohen, A. Calagna, P. Giaccone, and C. F. Chiasserini, "Distributed asynchronous service provisioning in edge-cloud multi-tier networks," *IEEE Trans. Mobile Comput.*, vol. 25, no. 1, pp. 644–659, Jan. 2026.

[3] F. G. Wakgra, B. Kar, S. B. Tadele, S.-H. Shen, and A. U. Khan, "Multi-objective offloading optimization in MEC and vehicular-fog systems: A distributed-TD3 approach," *IEEE Trans. Intell. Transp. Syst.*, vol. 25, no. 11, pp. 16897–16909, Nov. 2024.

[4] A. B. Rahman, P. Charatsaris, E. E. Tsiropoulou, and S. Papavassiliou, "Symbiotic resource pricing in the computing continuum era," *IEEE Trans. Mobile Comput.*, vol. 24, no. 7, pp. 6474–6487, Jul. 2025.

[5] T. Liu, S. Ni, X. Li, Y. Zhu, L. Kong, and Y. Yang, "Deep reinforcement learning based approach for online service placement and computation resource allocation in edge computing," *IEEE Trans. Mobile Comput.*, vol. 22, no. 7, pp. 3870–3881, Jul. 2023.

[6] Y. Cai, P. Cheng, Z. Chen, W. Xiang, B. Vucetic, and Y. Li, "Graphic deep reinforcement learning for dynamic resource allocation in space-air-ground integrated networks," *IEEE J. Sel. Areas Commun.*, vol. 43, no. 1, pp. 334–349, Jan. 2025.

[7] Y. Li et al., "Task placement and resource allocation for edge machine learning: A GNN-based multi-agent reinforcement learning paradigm," *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 12, pp. 3073–3089, Dec. 2023.

[8] Y. Cheng, C. Liang, Q. Chen, and F. R. Yu, "An efficient resource allocation scheme with uncertain network status in edge computing-enabled networks," *IEEE Trans. Mobile Comput.*, vol. 24, no. 3, pp. 1249–1263, Mar. 2025.

[9] D. Van Huynh, V.-D. Nguyen, S. R. Khosravirad, G. K. Karagiannidis, and T. Q. Duong, "Distributed communication and computation resource management for digital twin-aided edge computing with short-packet communications," *IEEE J. Sel. Areas Commun.*, vol. 41, no. 10, pp. 3008–3021, Oct. 2023.

[10] K. Li, P. Zhu, Y. Wang, F.-C. Zheng, and X. You, "Joint uplink and downlink resource allocation toward energy-efficient transmission for URLLC," *IEEE J. Sel. Areas Commun.*, vol. 41, no. 7, pp. 2176–2192, Jul. 2023.

[11] J. Pang, Z. Han, R. Zhou, R. Zhang, J. C. S. Lui, and H. Chen, "Eris: An online auction for scheduling unbiased distributed learning over edge networks," *IEEE Trans. Mobile Comput.*, vol. 23, no. 6, pp. 7196–7209, Jun. 2024.

[12] U. Habiba, S. Maghsudi, and E. Hossain, "A repeated auction model for load-aware dynamic resource allocation in multi-access edge computing," *IEEE Trans. Mobile Comput.*, vol. 23, no. 7, pp. 7801–7817, Jul. 2024.

[13] X. Wang, D. Wu, X. Wang, R. Zeng, L. Ma, and R. Yu, "Truthful auction-based resource allocation mechanisms with flexible task offloading in mobile edge computing," *IEEE Trans. Mobile Comput.*, vol. 23, no. 5, pp. 6377–6391, May 2024.

[14] Z. Sun, G. Sun, Y. Liu, J. Wang, and D. Cao, "BARGAIN-MATCH: A game theoretical approach for resource allocation and task offloading in vehicular edge computing networks," *IEEE Trans. Mobile Comput.*, vol. 23, no. 2, pp. 1655–1673, Feb. 2024.

[15] S. Sharma, S. Ghosh, M. R. Bhatnagar, and B. K. Panigrahi, "Power efficient handoff management in hybrid V2X communication: Game-theoretic approach to resource allocation with load reduction," *IEEE Trans. Mobile Comput.*, vol. 23, no. 12, pp. 14638–14655, Dec. 2024.

[16] X. Xu et al., "Game theory for distributed IoV task offloading with fuzzy neural network in edge computing," *IEEE Trans. Fuzzy Syst.*, vol. 30, no. 11, pp. 4593–4604, Nov. 2022.

[17] T. Huang, Z. Fang, Q. Tang, R. Xie, T. Chen, and F. R. Yu, "Dual-timescales optimization of task scheduling and resource slicing in satellite-terrestrial edge computing networks," *IEEE Trans. Mobile Comput.*, vol. 23, no. 12, pp. 14111–14126, Dec. 2024.

[18] J. Zheng, A. Banchs, and G. D. Veciana, "Constrained network slicing games: Achieving service guarantees and network efficiency," *IEEE/ACM Trans. Netw.*, vol. 31, no. 6, pp. 2698–2713, Dec. 2023.

[19] R. Han et al., "Dynamic network slice for bursty edge traffic," *IEEE/ACM Trans. Netw.*, vol. 32, no. 4, pp. 3142–3157, Aug. 2024.

[20] A. Asheralieva, D. Niyato, and Y. Miyanaga, "Efficient dynamic distributed resource slicing in 6G multi-access edge computing networks with online ADMM and message passing graph neural networks," *IEEE Trans. Mobile Comput.*, vol. 23, no. 4, pp. 2614–2638, Apr. 2024.

[21] Z. Chen, J. Zhang, G. Min, Z. Ning, and J. Li, "Traffic-aware lightweight hierarchical offloading toward adaptive slicing-enabled SAGIN," *IEEE J. Sel. Areas Commun.*, vol. 42, no. 12, pp. 3536–3550, Dec. 2024.

[22] C. Puligheddu, J. Ashdown, C. F. Chiasserini, and F. Restuccia, "SEM-O-RAN: Semantic and flexible O-RAN slicing for NextG edge-assisted mobile systems," in *Proc. IEEE Conf. Comput. Commun.*, 2023 , pp. 1–10.

[23] F. Mason, G. Nencioni, and A. Zanella, "Using distributed reinforcement learning for resource orchestration in a network slicing scenario," *IEEE/ACM Trans. Netw.*, vol. 31, no. 1, pp. 88–102, Feb. 2023.

[24] H. Sedghani, F. Filippini, and D. Ardagna, "SPACE4AI-D: A design-time tool for AI applications resource selection in computing continua," *IEEE Trans. Serv. Comput.*, vol. 17, no. 6, pp. 4324–4339, Nov./Dec. 2024.

[25] A. Taghinezhad-Niar and J. Taheri, "Security, reliability, cost, and energy-aware scheduling of real-time workflows in compute-continuum environments," *IEEE Trans. Cloud Comput.*, vol. 12, no. 3, pp. 954–965, Jul.–Sep. 2024.

[26] R. Sala, H. Sedghani, M. Passacantando, G. Verticale, and D. Ardagna, "AI applications resource allocation in computing continuum: A Stackelberg game approach," *IEEE Trans. Cloud Comput.*, vol. 13, no. 1, pp. 166–183, Jan.–Mar. 2025.

[27] M. Richart, J.-L. Gorricho, J. Baliosian, L. M. Contreras, A. Muñiz, and J. Serrat, "LQ-GNN: A graph neural network model for response time prediction of microservice-based applications in the computing continuum," *IEEE Trans. Parallel Distrib. Syst.*, vol. 36, no. 12, pp. 2566–2577, Dec. 2025.

[28] G. Yu, P. Chen, Z. Zheng, J. Zhang, X. Li, and Z. He, "FaaSDeliver: Cost-efficient and QoS-aware function delivery in computing continuum," *IEEE Trans. Serv. Comput.*, vol. 16, no. 5, pp. 3332–3347, Sep.–Oct. 2023.
[29] F. Li, Y. Hao, S. Yang, and P. Zhao, "OACR^2: Online admission control and resource reservation for 5G slice networks with deep reinforcement learning," *IEEE Trans. Mobile Comput.*, vol. 24, no. 8, pp. 7360–7376, Aug. 2025.
[30] H. Shen, Y. Tian, T. Wang, and G. Bai, "Slicing-based task offloading in space-air-ground integrated vehicular networks," *IEEE Trans. Mobile Comput.*, vol. 23, no. 5, pp. 4009–4024, May 2024.
[31] M. Dai, G. Sun, H. Yu, and D. Niyato, "Maximize the long-term average revenue of network slice provider via admission control among heterogeneous slices," *IEEE/ACM Trans. Netw.*, vol. 32, no. 1, pp. 745–760, Feb. 2024.
[32] M. N. Dazhi, H. Al-Hraishawi, M. R. B. Shankar, S. Chatzinotas, and J. Grotz, "Joint NTN slicing and admission control for infrastructure-as-a-service: A deep learning aided multi-objective optimization," *IEEE Trans. Cogn. Commun. Netw.*, vol. 11, no. 2, pp. 1297–1315, Apr. 2025.
[33] B. Hu, J. Du, J. Zhang, and X. Chu, "Computation offloading and resource allocation in mixed cloud/vehicular-fog computing systems," *IEEE Trans. Mobile Comput.*, vol. 24, no. 9, pp. 8612–8624, Sep. 2025.
[34] V. C. Pujol, P. K. Donta, A. Morichetta, I. Murturi, and S. Dustdar, "Edge intelligence—Research opportunities for distributed computing continuum systems," *IEEE Internet Comput.*, vol. 27, no. 4, pp. 53–74, Jul./Aug. 2023.
[35] Z. He, Y. Sun, B. Wang, S. Li, and B. Zhang, "CPU–GPU heterogeneous computation offloading and resource allocation scheme for industrial Internet of Things," *IEEE Internet Things J.*, vol. 11, no. 6, pp. 11152–11164, Mar. 2024.
[36] S. Liu et al., "Dependent task scheduling and offloading for minimizing deadline violation ratio in mobile edge computing networks," *IEEE J. Sel. Areas Commun.*, vol. 41, no. 2, pp. 538–554, Feb. 2023.
[37] L. Zhang, W. Zheng, K. Zheng, H. Zhu, C. Li, and M. Guo, "Bayesian-driven automated scaling in stream computing with multiple QoS targets," *IEEE Trans. Parallel Distrib. Syst.*, vol. 35, no. 7, pp. 1251–1267, Jul. 2024.
[38] Z. Nezami, K. Zamanifar, K. Djemame, and E. Pournaras, "Decentralized edge-to-cloud load balancing: Service placement for the Internet of Things," *IEEE Access*, vol. 9, pp. 64983–65000, 2021.
[39] Z. Nezami, E. M. Chaniotakis, and E. Pournaras, "When computing follows vehicles: Decentralized mobility-aware resource allocation for edge-to-cloud continuum," *IEEE Internet Things J.*, vol. 12, no. 13, pp. 23324–23340, Jul. 2025.

Shujia Niu (Student Member, IEEE) received the BS degree from Southwest Jiaotong University, China. She is currently working toward the master's degree with the School of Software, Yunnan University, Kunming, China, under the supervision of Professor Zhenli He. Her research focuses on distributed computing.

Zhenli He (Senior Member, IEEE) received the BS degree in software engineering and the MS and PhD degrees in software engineering and systems analysis and integration from Yunnan University, Kunming, China, in 2010, 2012, and 2015, respectively. From 2019 to 2021, he was a Postdoctoral Researcher with Hunan University, Changsha, China. He is currently an associate professor and the head with the Department of Software Engineering, School of Software, Yunnan University. His research interests include edge computing, energy-efficient computing, heterogeneous computing, and edge-AI techniques. Dr. He was selected as a Young Talent of the Yunnan Provincial "Xingdian Talent Support Program". He was the recipient of the Hongyun Gardener Award for teaching excellence.

Jixian Zhang received the MS and PhD degrees in computer science from the University of Electronic Science and Technology of China, in 2006 and 2010, respectively. He is currently an associate professor with the School of Computer Science and Engineering, Yunnan University. He has authored or coauthored 50+ articles in peer-reviewed journals and conference proceedings, including *IEEE Transactions on Mobile Computing*, *IEEE Transactions on Network and Service Management*, *IEEE Transactions on Network Science and Engineering*, FGCS, JOGC, *Cluster Computing*, and *Cloud Computing*. His research interests include cloud computing, edge computing, and mechanism design.

Cheng Xie (Senior Member, IEEE) received the BS degree in software engineering from the Minzu University of China, Beijing, China, in 2009, and the MS and PhD degrees in software engineering from Shanghai Jiao Tong University, Shanghai, China, in 2012 and 2017, respectively. From 2015 to 2016, he was a visiting scholar with Data and Web Science Group, University of Mannheim, Mannheim, Germany. He is currently a professor with the School of Software, Yunnan University, Kunming, China. His research interests include graph learning, knowledge graphs, and industrial informatics. Prof. Xie was the recipient of the Shanghai Science and Technology Progress Award (Second Prize) in 2014, Yunnan Science Award (Second Prize) in 2022, Youth Talent Project of China Association for Science and Technology, High-level Talent Program of Yunnan in 2018, and Donglu Young Scholars in 2021.

Keqin Li (Fellow, IEEE) received the BS degree in computer science from Tsinghua University, in 1985, and the PhD degree in computer science from the University of Houston, in 1990. He is currently a SUNY distinguished professor with the State University of New York and the National distinguished professor with Hunan University (China). He has authored or co-authored more than 1280 journal articles, book chapters, and refereed conference papers. He holds nearly 80 patents announced or authorized by the Chinese National Intellectual Property Administration. He is 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 #3) based on a composite indicator of the Scopus citation database. He is listed in Scilit Top Cited Scholars and among the top 0.02% out of more than 20 million scholars worldwide based on top-cited publications in the last ten years. He is listed in ScholarGPS Highly Ranked Scholars and among the top 0.002% out of more than 30 million scholars worldwide based on a composite score of three ranking metrics for research productivity, impact, and quality in the recent five years. He is a member of Academia Europaea, member of the European Academy of Sciences and Arts, fellow of the American Association for the Advancement of Science, member of the USA National Academy of Artificial Intelligence, fellow of the International Academy of Artificial Intelligence Sciences, fellow of the International Artificial Intelligence Industry Alliance, member of the World Academy of Sciences, fellow of the World Academy of Engineering & Technology, fellow of the Institute of Electrical and Electronics Engineers, fellow of the Asia Computational Intelligence Society, and member of the SUNY Distinguished Academy.