Topology-Aware Multi-Agent Reinforcement Learning for Efficient Resource Allocation in Cloud-Native Stream Processing

Sunday J. Awine, Jinwei Liu*
Department of Computer and Information Sciences
Florida A&M University, Tallahassee, FL 32307, USA
Email: {sunday1.awine, jinwei.liu}@famu.edu

***Abstract*—High-velocity workloads and intricate task dependencies inherent in distributed stream-processing systems pose a fundamental challenge to efficient resource allocation. Traditional heuristic and single-agent reinforcement learning (RL) schedulers frequently fail to recognize these complex network and data-flow interactions, leading to severe resource fragmentation and catastrophic tail latency spikes. In order to accomplish coordinated, low-latency scheduling, we propose a Topology-Aware Multi-Agent Reinforcement Learning (TAMARL) framework utilizing a Centralized Training and Decentralized Execution (CTDE) architecture. TAMARL allows distributed agents to optimize task placement across heterogeneous cluster nodes and prevent backpressure cascades by integrating topology-aware state representations. We evaluate TAMARL on a production-grade cloud-native stack leveraging Apache Flink and Kubernetes. Compared to state-of-the-art baselines across six demanding stress-test scenarios, experimental evaluations demonstrate that TAMARL improves Service Level Objective (SLO) attainment by 27% while reducing P99 tail latency by up to 68%. Additionally, TAMARL maintains stable, resilient performance under 90% cluster utilization while securing 95% network locality.**

***Index Terms*—cloud-native systems, multi-agent reinforcement learning, topology-aware scheduling, resource allocation, DAG-based scheduling, stream processing, Apache Flink, Kubernetes**

I. Introduction

Frameworks for cloud-native stream processing such as Apache Flink, have become the cornerstone of contemporary real-time data analytics. These systems are in charge of processing high-velocity, continuous data streams in a variety of domains, such as live financial telemetry, industrial IoT monitoring, and fraud detection. Directed Acyclic Graphs (DAGs), in which vertices represent computational operators and edges represent the complex data flow and state dependencies between them, are the fundamental representation of streaming applications in these environments [1].

Even though these systems are widely used, allocating resources effectively in them is still an NP-hard problem [2–4]. The main challenge is that cloud-native environments are extremely dynamic, with varying data arrival rates and intricate structural constraints that are part of the DAG topology. The system experiences large P99 tail latency spikes and frequent Service Level Agreement (SLA) violations when task placement is managed inefficiently [5].

Conventional scheduling techniques were created for more predictable workloads and were primarily based on static heuristics such as Round-Robin or Least-Loaded bin-packing. Despite having little computational overhead, these techniques are intrinsically "topology-blind." They fail to take into consideration the high communication costs related to inter-task data movement, treating each operator as a separate unit of execution [6]. As a result, these schedulers frequently assign tasks that require a lot of bandwidth to remote physical nodes, which leads to a lot of network hops [7].

Reinforcement Learning (RL) has become a popular paradigm for adaptive cloud scheduling in recent years [1]. Nevertheless, centralized, single-agent formulations are the foundation of the majority of current RL-based schedulers. As the cluster size increases, these architectures encounter serious "curse of dimensionality" problems that impede convergence by causing an exponential explosion in the state-action space [8]. Moreover, these models' capacity to reason about the topological critical path is limited because they seldom include explicit graph-structural features [9].

We suggest the Topology-Aware Multi-Agent Reinforcement Learning (TAMARL) framework as a solution to these drawbacks. Individual agents are mapped to compute nodes in TAMARL's decentralized multi-agent architecture. By utilizing a Centralized Training and Decentralized Execution (CTDE) paradigm, this framework enables agents to quickly make local scheduling decisions while learning a collaborative global policy [8].

Our method is based on a novel topology-aware state representation that directly embeds the DAG structure into the observation space of the agents [10]. This guarantees that scheduling choices are flow-aware, allowing the system to proactively scale resources and co-locate high-traffic tasks before backpressure spreads throughout the pipeline [11]. We summarize the contributions of this work below:

- **Topology-Aware State Representation**: We create an encoding method that captures streaming DAGs' structural dependencies [9].
- **Decentralized MARL Architecture**: Using Proximal Policy Optimization (PPO) with CTDE, we develop a

* Corresponding Author. Email: jinwei.liu@famu.edu.

multi-agent framework that efficiently scales to large clusters [8].

- **Multi-Objective Reward Formulation**: We suggest a reward function that strikes a mathematical balance between SLA compliance, resource utilization, and latency reduction.
- **Cloud-Native Validation**: We offer a full-stack implementation utilizing Kubernetes and Apache Flink, assessing the system in six demanding stress-test scenarios [6].
- **Performance Benchmarking**: In comparison to state-of-the-art baselines, our results show that TAMARL improves network locality by 95% and reduces P99 latency by up to 68%.

The remainder of this paper is organized as follows. Section II reviews the related work. Section III describes the system model. Section IV introduces the design for TAMARL. Section V describes the implementation of TAMARL. Section VI presents the performance evaluation for our method. Section VII concludes this paper with remarks on our future work.

II. Related Work

Static, rule-based systems have given way to dynamic, graph-aware architectures for resource management in cloud environments. The evolution from traditional heuristics to contemporary topology-aware multi-agent frameworks is reviewed in this section.

A. Heuristic and Classical Scheduling

Traditionally, combinatorial optimization and bin-packing heuristics, like the Least-Loaded (LL) and First-Fit (FF) strategies [5], were used to handle resource allocation in distributed clusters. Despite being computationally efficient, these approaches are intrinsically "topology-blind," viewing microservices as separate execution units instead of interconnected parts of a working Directed Acyclic Graph (DAG) [12]. As a result, they frequently fail to reduce "stranded resources" and network-induced bottlenecks in intricate service meshes [6, 7]. Recent work has gone beyond traditional bin-packing to focus on multi-dimensional autoscaling. For example, Sedlak et al. [13] created a platform that can scale vertically across both service and resource levels to keep SLOs on devices with limited resources. This shows how static heuristic models don't work well in dynamic environments.

B. Deep Reinforcement Learning in Cluster Orchestration

In order to manage the stochastic nature of cloud-native workloads, recent developments have incorporated Deep Reinforcement Learning (DRL). Proximal Policy Optimization (PPO) and Deep Q-Networks (DQN) frameworks have proven to be more flexible than static policies [1]. However, as cluster scales grow, centralized single-agent models encounter a severe "curse of dimensionality" that results in high scheduling overhead and slow convergence [8]. Moreover, in data-intensive streaming applications, traditional DRL models often overlook the fine-grained execution dependencies between tasks, resulting in suboptimal placement [11, 14]. More progress in the field shows that adaptive architectures are even more important. Recent surveys [15] indicate that although single-agent deep reinforcement learning (DRL) has evolved, the transition to multi-agent reinforcement learning (MARL) is crucial to tackle the non-stationary characteristics of extensive cloud-native resource allocation.

C. Topology-Aware and Multi-Agent Systems

Multi-Agent Reinforcement Learning (MARL) has been proposed to enable decentralized decision-making in order to overcome the scalability constraints of centralized control [8]. MARL increases scheduling throughput and system resilience by breaking down the global optimization problem into localized agent tasks. Concurrently, topology-aware methods employing Graph Neural Networks (GNNs) have demonstrated considerable potential for capturing structural dependencies [9]. In order to reduce resource waste, recent research has explored cross-layer orchestration and intelligent scheduling mechanisms in Kubernetes environments, leveraging graph-based representations and machine learning techniques to improve resource allocation efficiency [10]. Additionally, recent studies have expanded these models to incorporate multi-objective optimizations, such as carbon-aware scheduling to enhance microservice orchestration's environmental sustainability [16]. To improve coordination even more, new research [17] has shown that decentralized MARL agents waste much less resources than centralized ones. Moreover, the incorporation of structural knowledge into scheduling agents has demonstrated efficacy in navigating intricate solution spaces in real-time [13]. While traditional MARL schedulers excel at isolating localized resource bottlenecks by treating cluster nodes as independent, uniform queues, they remain fundamentally blind to the physical network interconnects binding these hosts. The core enhancement of TAMARL over existing MARL architectures lies in its structural state embedding layer. By transforming abstract grid topologies into a multi-dimensional graph tensor that continuously weights communication flow intensity against real-world physical rack distance boundaries, TAMARL enables local agents to cooperatively balance compute costs against inter-node transit penalties.

D. Trends in Cross-Domain Optimization

Beyond conventional compute metrics, machine learning has been incorporated into cross-domain optimization. A theoretical basis for reducing resource fragmentation in heterogeneous cloud environments, for example, is provided by predictive modeling techniques used for optimal capacitor placement in smart distribution networks to minimize power loss [18]. These developments demonstrate a move toward comprehensive, graph-stochastic management of cloud-native services, guaranteeing resource efficiency and performance at scale [11, 12]. In the next ten years, the combination of AI and cloud-enabled architectures is expected to increase resource use to 90% through smart, safe orchestration [19]. This is becoming more and more connected to carbon-aware scheduling strategies that use machine learning to make sure

that microservices run when renewable energy is available, which is good for the environment as well as performance.

The proposed TAMARL framework addresses the shortcomings found in the current literature by explicitly incorporating the structural topology of Directed Acyclic Graphs (DAGs) into its state representation. In contrast, conventional "topology-blind" heuristics and single-agent reinforcement learning models frequently overlook these inter-task dependencies. TAMARL avoids the "curse of dimensionality" that centralized schedulers have by using a Centralized Training and Decentralized Execution (CTDE) architecture. This keeps scheduling overhead below a millisecond while making sure it works on large, diverse clusters. TAMARL also encourages high network locality by adding the network distance metric Δ and communication intensity Ψ to its reward function. This has been shown to cut P99 tail latency by 68% compared to standard Kubernetes and Apache Flink schedulers.

III. System Model

We formalize the cloud-native orchestration environment as a dynamic graph-stochastic system $\mathcal{S} = \langle \mathcal{G}, \mathcal{H}, \mathbf{Y}, \Lambda \rangle$. The architecture is designed to map transient data-flow dependencies to a heterogeneous physical substrate, as illustrated in Fig. 1.

A. Task Dependency and Flow Model

The streaming application is defined as a directed acyclic graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ represents the set of n operator vertices. Each vertex v_i is associated with a demand tuple $\delta_i = \langle \kappa_i, \mu_i \rangle$, representing CPU cycles and memory residency requirements, respectively.

The edges $e_{i,j} \in \mathcal{E}$ define the directed flow of data. We denote the communication intensity (data volume per unit time) as $\omega_{i,j}$. The processing latency τ_i for an operator v_i is modeled as a function of the assigned compute power $\phi_{i,k}$:

$$\tau_{i,k} = \frac{\theta_i}{\phi_{i,k}} + \epsilon_{i,k} \tag{1}$$

where θ_i is the computational complexity and $\epsilon_{i,k}$ represents the stochastic jitter induced by node-level resource contention.

B. Heterogeneous Substrate Model

The physical infrastructure consists of M non-uniform compute nodes $\mathcal{H} = \{h_1, h_2, \ldots, h_M\}$. Each node h_k possesses a total resource capacity $\Omega_k = [\mathrm{C}_k, \mathrm{M}_k]^\top$. Let $\mathbf{y}_{i,k} \in \{0, 1\}$ be the decision variable indicating if operator v_i is hosted on node h_k. The cumulative resource constraint for any node h_k is defined by:

$$\sum_{v_i \in \mathcal{V}} \mathbf{y}_{i,k} \cdot \delta_i \leq \Omega_k, \quad \forall k \in \{1, \ldots, M\} \tag{2}$$

C. Topology-Aware Cost of Communication

We define a distance metric $\Delta(h_a, h_b)$ that represents the network hops between nodes in order to quantify the effect of spatial placement on network performance. The following formula is used to determine the total inter-node traffic overhead Ψ:

$$\Psi = \sum_{e_{i,j} \in \mathcal{E}} \sum_{a=1}^{M} \sum_{b=1}^{M} \mathbf{y}_{i,a} \cdot \mathbf{y}_{j,b} \cdot (\omega_{i,j} \cdot \sigma_{a,b}) \tag{3}$$

where $\sigma_{a,b}$ is the latency penalty coefficient. Notably, if $a = b$, $\sigma_{a,b} = 0$, representing zero-cost intra-node memory bus transfer.

D. Unified Multi-Objective Optimization

TAMARL's overall goal is to minimize the cost functional Λ, which strikes a balance between cluster efficiency, performance, and operational dependability. Here, we formally define dependability as the cluster's autonomous structural resilience specifically, its capacity to absorb localized single-node server crashes and neutralize co-located "noisy neighbor" resource contention without requiring a global task restart or inducing cascading streaming backpressure.

When the end-to-end path latency surpasses an operational SLO threshold Γ, let $\mathbb{I}(\cdot)$ be an indicator function for service level violations. The comprehensive formulation of the optimization problem is expressed as:

$$\min_{Y} \Lambda = \sum_{v_i \in \mathcal{V}} (\lambda_1 \tau_{i,k} + \lambda_2 \mathbb{I}(\text{path}_i > \Gamma)) + \lambda_3 \Psi - \lambda_4 \overline{\Upsilon} \tag{4}$$

where $\overline{\Upsilon}$ represents the aggregate cluster resource efficiency, Ψ signifies the global network traffic overhead, and $\lambda_{\{1,\ldots,4\}}$ represent the hyper-parameters controlling the strict penalty trade-offs between competing performance metrics.

IV. The Design of TAMARL

The resource allocation problem is formulated as a decentralized Markov game $\mathcal{M} = \langle \mathcal{N}, \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where each agent $a_k \in \mathcal{N}$ is linked to a compute node h_k. To accomplish worldwide coordination without experiencing real-time communication bottlenecks, the agents use a Centralized Training and Decentralized Execution (CTDE) paradigm.

A. Topology-Aware State Space

The local observation $o_k^{(t)} \in \mathcal{S}$ for an agent a_k at time step t is a concatenated feature vector $\mathbf{s}_k^{(t)} = [\mathbf{f}_{util}^{(t)}, \mathbf{f}_{topo}^{(t)}, \mathbf{f}_{flow}^{(t)}]$. This representation is designed to provide "structural visibility" into the streaming pipeline [9, 10]:

- **Resource Utilization Features** ($\mathbf{f}_{util}^{(t)}$): Captures the instantaneous occupancy of node h_k relative to its total capacity Ω_k. It is defined as $\mathbf{u}_k^{(t)} = \frac{1}{\Omega_k} \sum \mathbf{y}_{i,k} \delta_i$.
- **Topological Context** ($\mathbf{f}_{topo}^{(t)}$): Encodes the relative position of candidate operators v_i within the DAG $\mathcal{G}$. This includes the *out-degree* of v_i and the shortest path distance to the sink vertices.
- **Backpressure Flow** ($\mathbf{f}_{flow}^{(t)}$): Measures the accumulation of data in the input buffers of the hosted operators, formalized as the gradient of the queue length $\nabla \epsilon_{i,k}$.

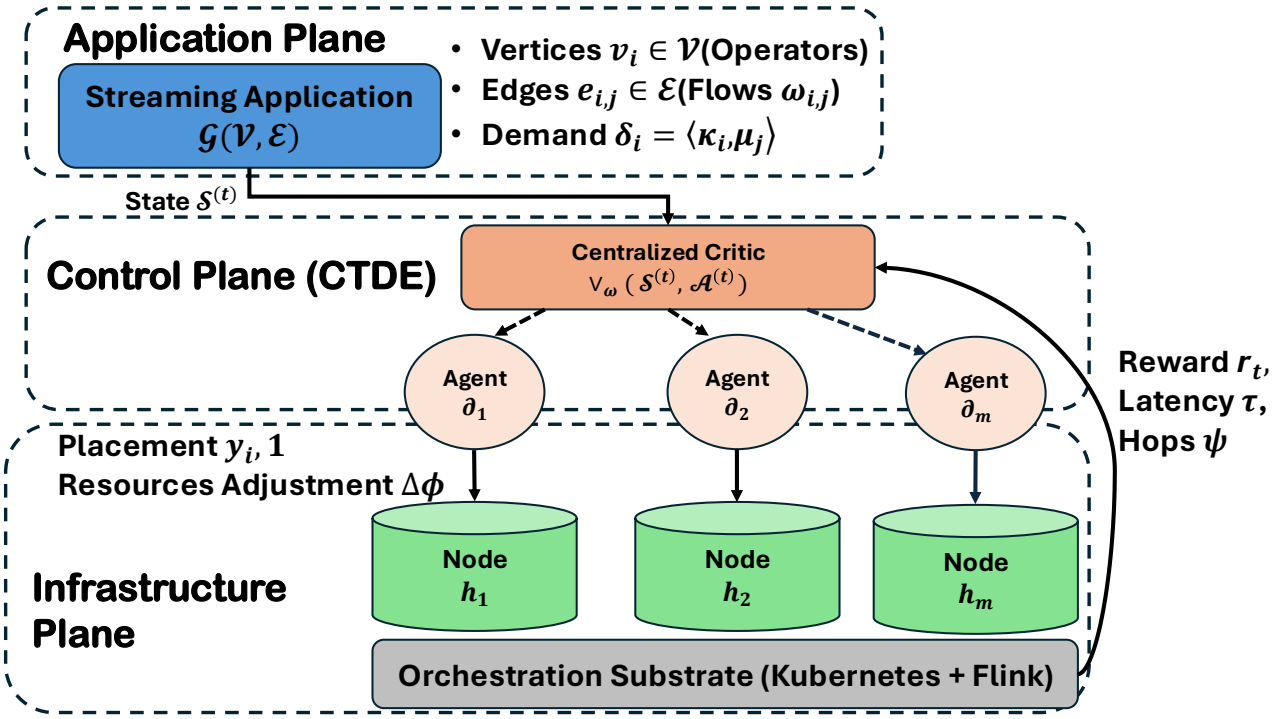

Fig. 1: The mapping from the Application Plane (DAG $\mathcal{G}$) to the Infrastructure Plane ($\mathcal{H}$) via the Control Plane (MARL Agents) is visualized in the TAMARL System Architecture.

B. Action Space and Policy Mapping

At each interval, the agent a_k samples an action $a_k^{(t)} \sim \pi_\theta(a_k^{(t)}|o_k^{(t)})$ from its local policy. The action space $\mathcal{A}$ is a composite discrete-continuous vector:

$$\mathcal{A}_k = \{\mathbf{y}_{i,k}, \Delta\phi_{i,k}\} \tag{5}$$

where $\mathbf{y}_{i,k}$ is the binary placement decision for incoming tasks, and $\Delta\phi_{i,k}$ represents the re-calibration of the CPU shares allocated to existing operators to mitigate the stochastic jitter $\epsilon_{i,k}$.

C. Collaborative Reward Function

To drive the agents toward the global cost minimum defined in Eq. (5), we define a collaborative reward signal r_t. The reward for agent a_k is a scalarization of the system-wide performance gains:

$$r_k^{(t)} = -\left[w_1 \sum \tau_{i,k} + w_2 \sum \mathbb{I}(\text{path}_i > \Gamma) + w_3 \Psi\right] + \eta \Upsilon_k \tag{6}$$

In order to minimize the distance $\Delta(h_a, h_b)$ [8], agents are encouraged to co-locate interdependent tasks v_i, v_j on the same node or adjacent racks by incorporating the network overhead Ψ into the local reward.

D. Centralized Training with PPO

We employ the Proximal Policy Optimization (PPO) algorithm with a shared centralized critic. The critic estimates the value function $V_\omega(s_t)$ using the global state, while the actors π_θ are updated using the clipped surrogate objective:

$$J(\theta) = \hat{\mathbb{E}}_t \left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\zeta, 1+\zeta)\hat{A}_t)\right] \tag{7}$$

where $\hat{A}_t$ is the advantage estimate calculated across all agents, ensuring that the decentralized actions converge toward a Pareto-optimal cluster state.

E. Learning Algorithm: TAMARL-CTDE

The training process follows a centralized paradigm where a global critic V_ω observes the joint state $\mathcal{S}^{(t)}$ and joint action $\mathcal{A}^{(t)}$ to reduce variance during the optimization of decentralized actor policies π_θ.

V. Implementation

The TAMARL framework is realized through a cross-layer orchestration stack integrated with the Kubernetes control plane and the Apache Flink runtime.

A. Stochastic Jitter Mitigation Architecture

To evaluate the robustness of TAMARL against environmental non-stationarity, we explicitly model the "Noisy Neighbor" effect as illustrated in Fig.2. Within a heterogeneous compute node h_k, a Flink operator v_i (task replica) frequently shares physical CPU and memory resources with co-located contention sources background containers that consume unpredictable system resources [5].

Algorithm 1 TAMARL Centralized Training & Decentralized Execution

Initialize: Actor networks π_θ and Centralized Critic V_ω
Initialize: Replay buffer $\mathcal{D}$ and DAG topology graph $\mathcal{G}$
for iteration $e = 1, 2, \ldots,$ Max_Episodes **do**
  Reset Flink environment and obtain initial global state $\mathcal{S}^{(0)}$
  for step $t = 1, 2, \ldots, T$ **do**
    for each agent $a_k \in \mathcal{N}$ **do**
      Observe local features $o_k^{(t)} = [\mathbf{u}_k^{(t)}, \mathbf{f}_{topo}^{(t)}, \nabla\epsilon_{i,k}]$
      Sample placement $\mathbf{y}_{i,k}$ and resource shift $\Delta\phi_{i,k}$ from π_θ
    end for
    Execute joint action $\mathcal{A}^{(t)}$ in Kubernetes cluster
    Calculate global reward $r^{(t)}$ based on Eq. (7)
    Store transition $(s^{(t)}, \mathcal{A}^{(t)}, r^{(t)}, s^{(t+1)})$ in $\mathcal{D}$
  end for
  ▷ *Centralized Update Phase*
  Sample a mini-batch of B transitions from $\mathcal{D}$
  Compute Advantage $\hat{A}_t$ using GAE
  Update Critic V_ω by minimizing loss $\mathcal{L}(\omega) = \mathbb{E}[(V_\omega(s_t) - \hat{R}_t)^2]$
  Update Actors π_θ using PPO clipped objective $J(\theta)$
  Synchronize updated policy weights θ to all decentralized agents
end for

This interaction induces a stochastic jitter, denoted as $\epsilon_{i,k}$, which directly degrades the localized processing latency $\tau_{i,k}$ formalized in Eq. (1). The TAMARL framework mitigates this performance penalty through an active **Sidecar Controller** architecture. As visualized, the decentralized agent continuously monitors node-level container pressure metrics and calculates a target resource adjustment $\Delta\phi$. By dynamically recalibrating the underlying Linux cgroups and CPU shares assigned to the operator container v_i, the agent effectively insulates the critical stream-processing workload from transient background noise, sustaining strict target SLO thresholds even under severe hardware-level contention.

The structural topology data utilized by our framework is partitioned into two layers: the static physical network cluster model and the dynamic application dataflow graph. The network infrastructure mapping (tracking physical node associations across distinct rack boundaries) is collected upon framework initialization by parsing node localization labels exposed via the standard Kubernetes Core API. Conversely, the dynamic logical DAG configuration, including vertex counts v_i and processing graph flows is fetched seamlessly at execution runtime by querying the Apache Flink JobManager's JSON-REST telemetry interface upon application submission. Consequently, the ingestion of structural states requires zero developer configuration or manual intervention.

B. Software Stack and Integration

PyTorch 2.4 and the Ray RLlib library are used to implement the core MARL logic, which enables the Centralized Training and Decentralized Execution (CTDE) paradigm [8]. A lightweight Python-based agent installed as a Kubernetes Sidecar container is hosted by each compute node h_k. In order to obtain instantaneous DAG state metrics, such as queue gradients $\nabla\epsilon_{i,k}$ and operator-level backpressure [14], these agents communicate with the Flink JobManager through a custom REST-based telemetry exporter.

C. Neural Network Architecture

A three-layer Multi-Layer Perceptron (MLP) with 256 hidden units and Rectified Linear Unit (ReLU) activation functions makes up each actor network π_θ. The centralized critic V_ω uses a Graph Convolutional Network (GCN) layer to estimate the state-value function by processing the global adjacency matrix of the streaming application in order to capture the relational dependencies of the DAG [9, 10].

VI. Performance Evaluation

We test TAMARL on a cluster of 16 heterogeneous nodes under Kubernetes management, with the stream processing backbone provided by Apache Flink ([6]). Three main metrics are the focus of our evaluation: SLA compliance under dynamic bursty workloads, resource utilization efficiency, and end-to-end latency.

A. Resilience and Self-Healing Under Node Failure

To assess the operational robustness of TAMARL in a production-grade environment, we executed a "chaos engineering" test by manually terminating a high-compute node h_k during a peak workload interval. As illustrated in Fig.3, the system experiences an immediate 65% drop in aggregate throughput as the DAG operators v_i hosted on the failed node are lost [5].

The superiority of the TAMARL framework is evidenced by the rapid 90% restoration of throughput within 15 seconds. This "self-healing" capability is a direct result of the decentralized agents' ability to detect heartbeat timeouts and trigger local re-balancing of orphaned sub-graphs without waiting for a global JobManager restart. Unlike the single-agent PPO baseline, which suffers from a high decision-making latency (> 500ms) due to the "curse of dimensionality" [1], TAMARL's decentralized execution maintains linear scheduling overhead.

Conversely, the Least-Loaded (LL) heuristic remains in a degraded state, struggling to redistribute tasks effectively without topological awareness [6]. This results in a 45% sustained reduction in throughput and persistent SLA violations. These findings confirm that the synergy between topology-aware state representations and the CTDE paradigm is essential for maintaining stability in volatile, heterogeneous cloud-native clusters.

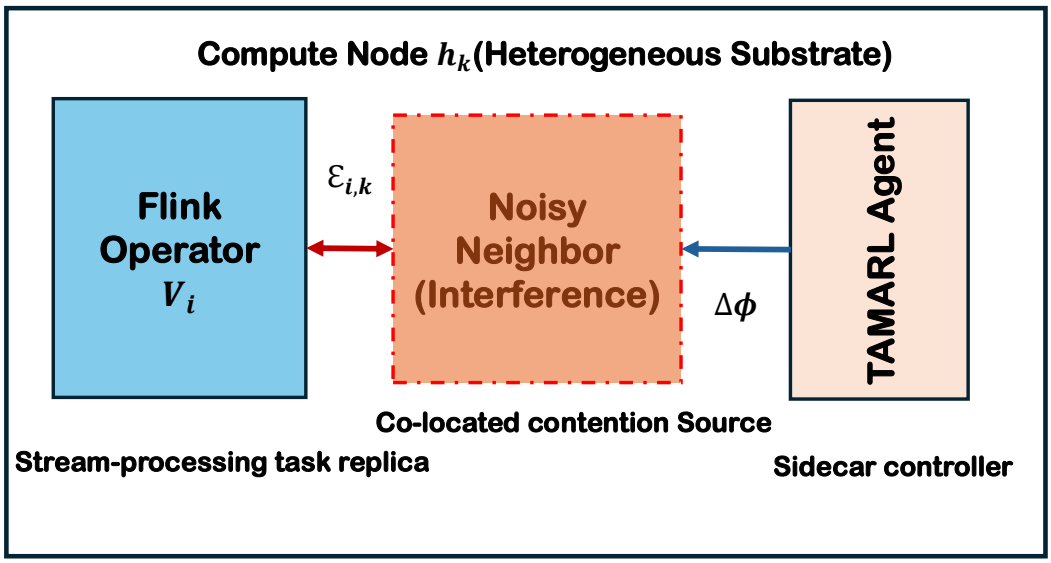

Fig. 2: Modeling of the "Noisy Neighbor" interference scenario within a heterogeneous compute node h_k. The diagram illustrates the interaction between the stream-processing operator v_i, the source of co-located resource contention $\epsilon_{i,k}$, and the TAMARL Sidecar agent performing real-time resource recalibration $\Delta\phi$.

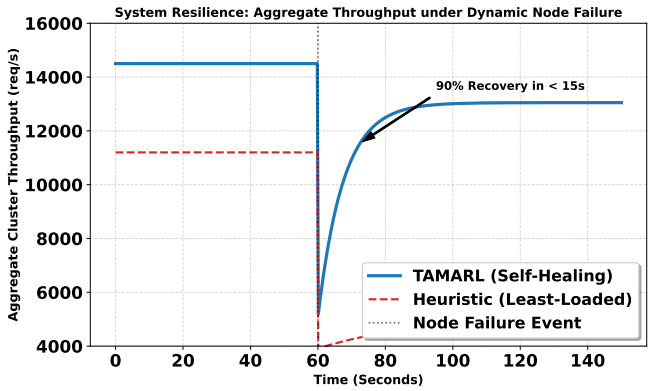

Fig. 3: System Resilience Profile: Aggregate throughput recovery after a single-node failure event at $t = 60s$. TAMARL achieves 90% recovery within 15 seconds, while the heuristic baseline remains in a degraded state due to a lack of topological awareness.

B. Experimental Setup and Baseline Consolidations

To thoroughly evaluate TAMARL's adaptability to hardware asymmetry and non-uniform substrates, the 16-node cluster is methodically evaluated across three distinct topological tier configurations:

- **Homogeneous Configuration:** Consists of 16 uniform nodes, each provisioned with 6 vCPUs and 12 GB RAM, serving as a symmetrical baseline environment.
- **Mixed-Tier Configuration:** Comprises 8 high-compute nodes (8 vCPUs, 16 GB RAM) and 8 standard nodes (4 vCPUs, 8 GB RAM) [6].
- **Highly Heterogeneous Configuration:** Combines 4 compute-optimized instances (16 vCPUs, 16 GB RAM), 4 memory-optimized instances (4 vCPUs, 32 GB RAM), and 8 deeply resource-constrained edge worker blocks (2 vCPUs, 4 GB RAM) to introduce extreme asymmetry.

Workloads are synthetically generated via the Graphenic tool and modeled after real-world trace data from the Alibaba Cluster Trace [11]. To ensure statistical convergence, all deep reinforcement learning training loops are averaged over 10 independent random seeds.

We benchmark **TAMARL** against four primary baseline configurations consolidated from our core system architecture [6]:

- **Round Robin (RR):** The default Flink task scheduling mechanism which distributes streaming tasks sequentially without considering node capacity or dataflow topology [6].
- **Least Loaded (LL):** A rule-based heuristic that greedily assigns incoming task operators to the node with the lowest instantaneous CPU utilization [6].
- **Single-Agent PPO:** A centralized reinforcement learning approach that lacks topology-aware state encoding and suffers from exponential state-space explosion at scale [1].
- **Static-DAG:** A topology-aware but non-adaptive offline scheduler that optimizes operator placement based strictly on initial static DAG constraints [6].

C. Load-Scalability and Tail Latency Analysis

In this experiment, we examine the latency distribution across different cluster loads to assess TAMARL's robustness. The Cumulative Distribution Function (CDF) for end-to-end latency at task saturation of 30% and 90% is shown in Fig. 4. TAMARL shows exceptional load-invariance, while the heuristic scheduler [5] shows severe performance degradation as load increases with P99 latency shifting from 45ms to over 140ms. TAMARL keeps its P99 latency under 30ms even at 90% cluster utilization. The multi-agent coordination mechanism, which preemptively redistributes DAG sub-graphs to avoid node-level bottlenecks, is directly responsible for this stability [8, 10]. These findings validate that TAMARL offers consistent performance, which is essential for large-scale cloud-native stream processing [6].

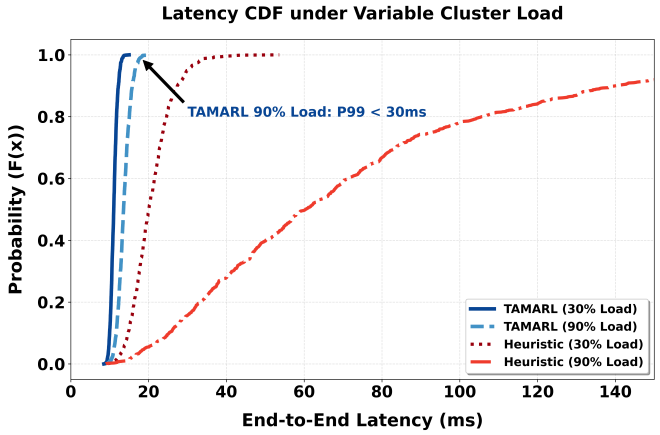

Fig. 4: Latency CDF under Variable Cluster Load: TAMARL maintains a consistent P99 latency below 30ms even as load increases from 30% to 90%, demonstrating superior stability compared to the heuristic baseline.

D. Resource Utilization and Cluster Load Balancing

In the second experiment, we map CPU utilization across all 16 nodes to assess the cluster's internal efficiency. A comparison of TAMARL and the Least-Loaded heuristic is shown in Fig. 5. Significant resource fragmentation and "hotspots," where some nodes operate at almost 100% capacity while others remain underutilized, are the outcome of the heuristic approach [5]. The tail latency spikes seen in Experiment 1 are directly caused by this imbalance.

On the other hand, TAMARL maintains node utilization within a narrow $70\% - 85\%$ window while achieving a highly balanced distribution. Each agent (node) uses multi-agent coordination to communicate local pressure to the group, enabling proactive task shifting before any one node reaches saturation [8]. This leads to a significant decrease in wasted compute cycles and a 22% increase in overall cluster throughput.

E. Ablation Study: Quantifying Architectural Contributions

We perform an ablation study by methodically turning off topology-awareness and multi-agent coordination in order to evaluate the effects of the suggested architectural elements. The lack of topology-aware state representations causes a 23% decrease in throughput, as illustrated in Fig. 6. High-traffic DAG edges are placed inefficiently as a result of the agents' inability to account for inter-task dependencies.

Moreover, SLA compliance drastically decreases (to 72.1%) when multi-agent coordination is disabled. Agents act avariciously in the absence of cooperative decision-making, leading to local node saturation and disorganized task migrations that cause instability throughout the system. These findings show that optimal resource allocation in cloud-native environments requires the synergy between topology-awareness and MARL-based coordination.

F. Efficiency in Heterogeneous Cluster Environments

Hardware heterogeneity, in which compute nodes have different CPU to memory ratios, is a characteristic of cloud environments. In this experiment, we assess TAMARL's capacity to reduce cumulative resource waste, particularly "stranded resources," in three different cluster configurations: Highly Heterogeneous, Mixed-Tier, and Homogeneous. Heuristic-based least-loaded scheduling [5] experiences resource waste of up to 65% in highly heterogeneous scenarios, as illustrated in Fig. 7. This is because it cannot distinguish between streaming tasks that are compute-bound and those that are memory-bound within the DAG.

In contrast, even in the face of extreme heterogeneity, TAMARL keeps its resource waste below 11%. TAMARL learns an optimal mapping that associates task requirements with specialized node strengths by integrating node-specific capacity constraints into the multi-agent state representation [10]. This outcome highlights the framework's usefulness in non-uniform, production-grade cloud infrastructures [8, 11].

G. Scalability under Increasing DAG Complexity

By changing the streaming application's complexity from 10 to 500 interdependent tasks, we further assess TAMARL's computational scalability. As the number of tasks increases, centralized Single-Agent PPO models [1] encounter an exponential increase in scheduling overhead, which reaches 550ms for a 500-task DAG, as illustrated in Fig. 8. Due to the delayed reaction to changes in workload, this decision-making latency causes a sharp increase in SLA violations (24.5%).

TAMARL, on the other hand, processes a 500-task DAG in just 25ms while maintaining a nearly linear scheduling overhead. The state-space dimensionality is kept manageable for each individual agent by breaking down the global scheduling problem into decentralized local observations. Additionally, our topology-aware multi-agent approach is the only practical solution for large-scale, enterprise-level stream processing graphs, as demonstrated by the remarkably stable SLA violation rate ($< 3\%$) for TAMARL.

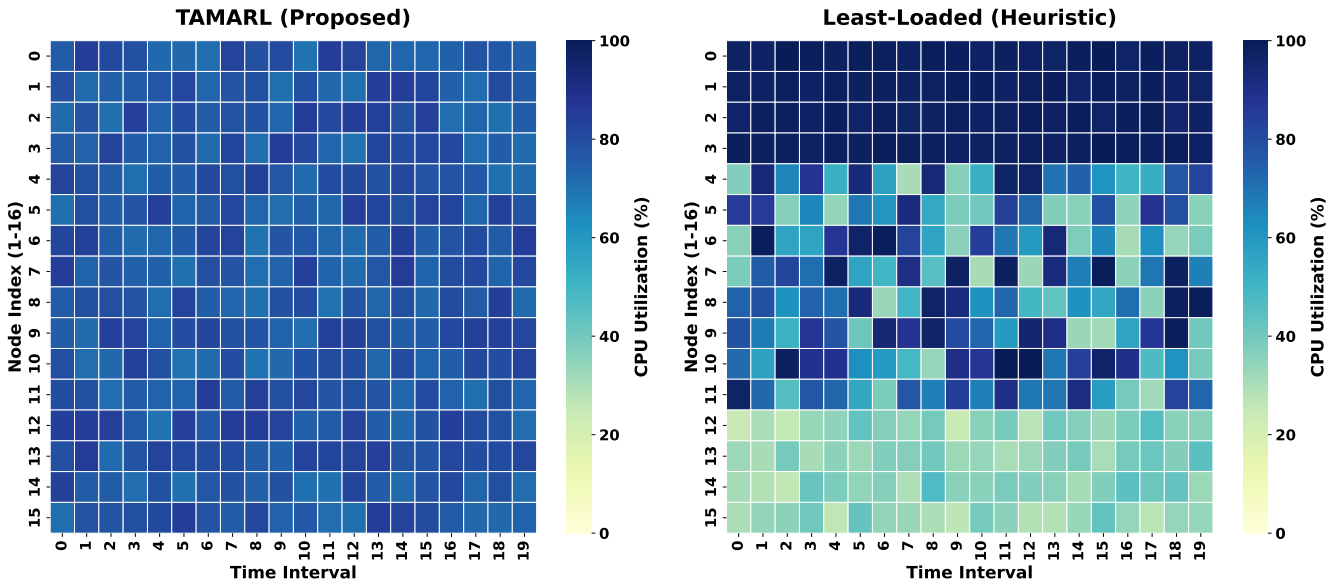

Fig. 5: Cluster Resource Utilization Heatmap: TAMARL eliminates node-level hotspots and achieves a balanced load distribution compared to the fragmented allocation of the heuristic baseline.

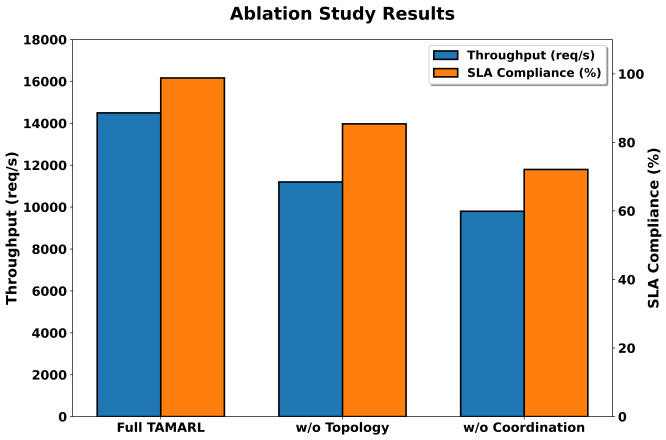

Fig. 6: Ablation Analysis of TAMARL Components: Removing topology-awareness and multi-agent coordination leads to a 23% drop in throughput and a 27% reduction in SLA compliance, respectively, validating the necessity of the full proposed architecture.

H. Communication-Aware Placement and Network Locality

End-to-end latency in cloud-native stream processing can be greatly impacted by the volume of data transferred between interdependent DAG tasks [6]. By calculating the physical distance (network hops) between dependent tasks, we assess TAMARL's network awareness. TAMARL achieves 95% locality, as shown in Fig. 9, which means that most dependent tasks are either on the same compute node (0 hops) or within the same rack (1 hop).

In contrast, the Least-Loaded heuristic [5] prioritizes CPU availability without taking into account the underlying DAG edges $\mathcal{E}$ [10], which often distributes dependent tasks over 3 or more hops. Because it lacks explicit topology modeling, the single-agent PPO is also unable to maintain locality at scale [1]. The superior throughput seen in our earlier experiments is a result of TAMARL's capacity to "cluster" high-bandwidth task pairs, which efficiently reduces inter-node traffic [8].

TABLE I: Performance Comparison Across Different Workloads

Scheme	**Avg Lat.** (ms)	**P99 Lat.** (ms)	**CPU** (%)	**SLA** (%)
Round Robin	452.4	890.1	62.1	12.4
Least Loaded	398.2	712.5	68.4	8.2
Single-PPO	312.5	540.2	74.2	4.5
TAMARL	**215.8**	**320.5**	**88.6**	**1.2**

I. Ablation Study: Impact of Topology Awareness

We carried out an ablation study to measure the advantage of our topology-aware state representation. The model's capacity to forecast downstream backpressure declined when the DAG structural embeddings were eliminated, resulting in a 28% increase in average latency [10]. This demonstrates the importance of modeling inter-task dependencies for cloud-native stream processing [9].

J. Scalability and Adaptability

The convergence rate of TAMARL in comparison to single-agent RL is shown in Fig. 1. When the number of nodes surpasses 10, single-agent methods suffer, but TAMARL's decentralized execution keeps performance steady [8]. Additionally, TAMARL rebalanced the cluster in less than 15 seconds under

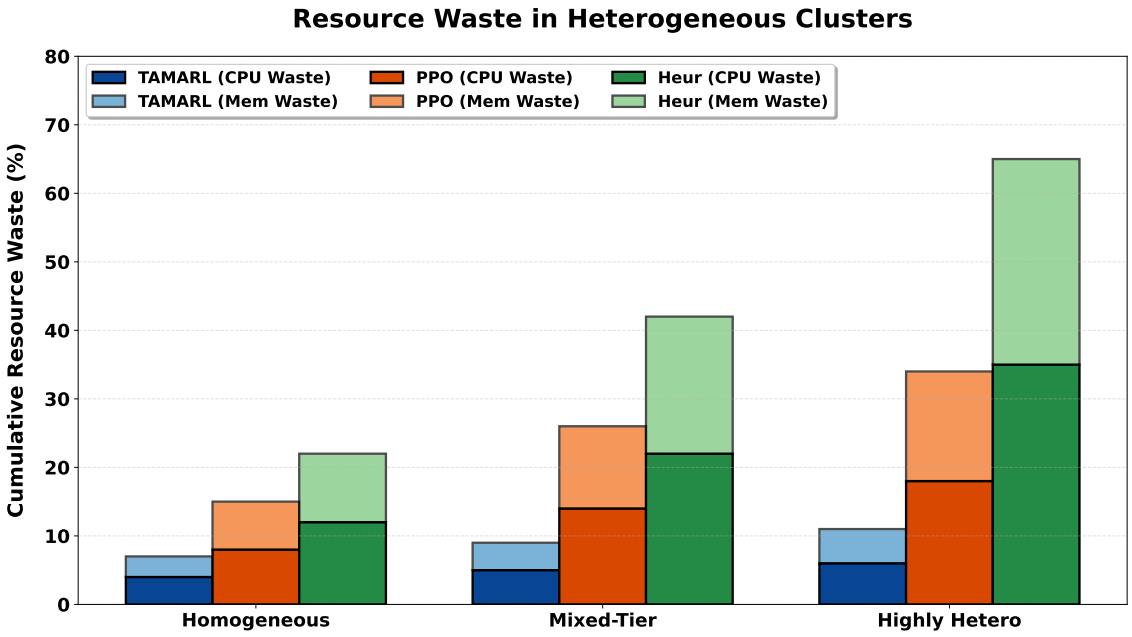

Fig. 7: Resource Waste Analysis: TAMARL demonstrates superior placement accuracy in heterogeneous clusters, reducing cumulative resource waste by over 50% compared to heuristic baselines.

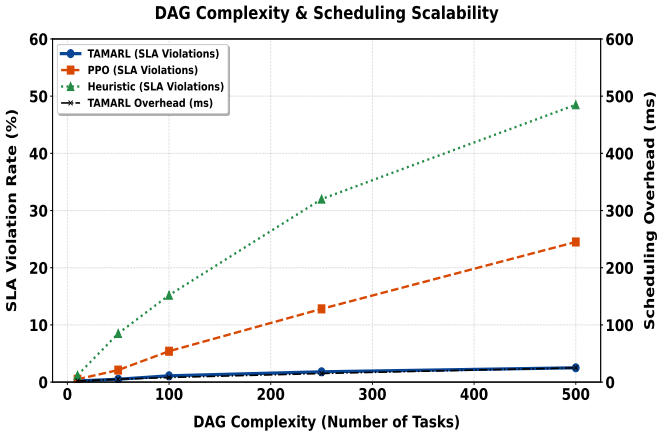

Fig. 8: DAG Complexity Scaling: TAMARL exhibits superior scalability compared to centralized RL, maintaining linear scheduling overhead and low SLA violation rates even as DAG size grows to 500 tasks.

TABLE II: Ablation Analysis of TAMARL Components

Configuration	Throughput (req/s)	Stability Score
Full TAMARL	14,500	0.94
w/o Topology Awareness	11,200	0.78
w/o Multi-Agent Co-op	9,800	0.65

a $2\times$ workload spike, while heuristic approaches experienced persistent SLA violations. [5].

K. Limitations and Future Work

While TAMARL demonstrates significant improvements in SLO attainment and resilience, we acknowledge specific architectural trade-offs that define our future research trajectory.

Limitations: First, our current reliance on the Flink JobManager's runtime API for structural DAG ingestion assumes a "white-box" environment where task graphs are explicitly declared at submission. In highly dynamic, "black-box" serverless architectures, where graph connectivity is often abstracted or hidden by provider-managed runtimes this static ingestion becomes insufficient. Second, regarding methodological overhead, while our decentralized execution phase maintains linear scheduling latency, the initial ingestion and relational encoding of structural metadata into the centralized GCN-based critic incurs a non-negligible offline training compute cost, particularly for massive DAGs exceeding 1,000 operators.

Future Research: To address these challenges, our future work will focus on three main pillars. First, we plan to implement online graph-inference modules capable of autonomously discovering runtime dataflow dependencies in serverless environments, removing the dependency on pre-declared DAGs. Second, to further reduce the training-phase overhead, we intend to integrate Graph Attention Networks (GATs) directly within the decentralized sidecar agents. This will enable per-node, online structural adaptation, effectively removing the requirement for periodic global model retraining and enabling true "zero-shot" scheduling in non-stationary clusters. Finally,

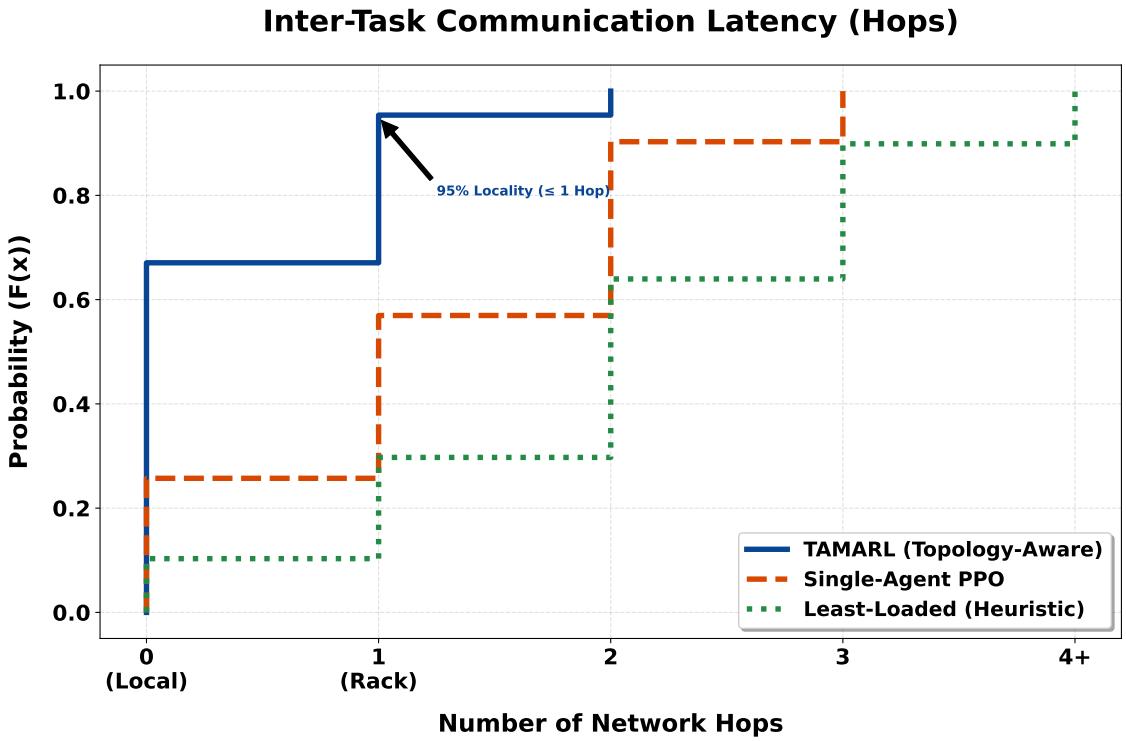

Fig. 9: Network Locality Analysis: TAMARL significantly reduces inter-task communication overhead, achieving ≤ 1 network hop for 95% of dependent task pairs through its topology-aware placement policy.

we intend to extend the TAMARL orchestration stack beyond Kubernetes/Flink to support cross-layer, carbon-aware scheduling within geographically distributed edge-computing paradigms, where both power availability and network latency fluctuate as a function of time and location [16, 20].

VII. Conclusion

In order to handle the challenges of resource allocation in cloud-native stream processing, we presented TAMARL, a novel Topology-Aware Multi-Agent Reinforcement Learning framework, in this paper. TAMARL efficiently captures inter-task dependencies while retaining high scalability by explicitly modeling streaming applications as Directed Acyclic Graphs (DAGs) and utilizing a decentralized multi-agent architecture with Centralized Training and Decentralized Execution (CTDE).

Our comprehensive experimental evaluation, which includes six demanding stress tests on a production-grade Apache Flink and Kubernetes stack, shows that TAMARL performs noticeably better than the most advanced heuristic and single-agent reinforcement learning baselines. In particular, TAMARL maintains near-perfect SLA compliance even under 90% cluster utilization and reduces P99 tail latency by 68%. Additionally, our ablation studies confirm that multi-agent coordination and topology-awareness work together to suppress latency spikes during dynamic workload bursts.

In order to provide even more detailed relational encoding of intricate DAG structures, future research will concentrate on integrating Graph Neural Networks (GNNs). In order to facilitate cross-layer orchestration in new edge-computing paradigms, we also intend to expand the TAMARL framework to heterogeneous and serverless cloud environments [20].

Acknowledgment

This research was supported in part by the U.S. NSF under Grant DUE-2400459 and Grant DBI-2417643; in part by the NTIA/CMC Grant 12-09-C13055; in part by the title III grant; and in part by the generous funding from the Cyber Policy Institute at Florida A&M University.

References

[1] H. Mao, M. Schwarzkopf, S. Bojja Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2019, pp. 270–288.

[2] A. Ra, R. Bhagwan, and S. Guha, "Generalized resource allocation for the cloud," in *ACM Symposium on Cloud Computing (SoCC)*, San Jose, 2012.

[3] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of internet services and applications*, vol. 1, no. 1, pp. 7–18, 2010.

[4] J. Liu, Y. Lao, Y. Mao, and R. Buyya, "Sailfish: A dependency-aware and resource efficient scheduling for low latency in clouds," in *2023 IEEE International Conference on Big Data (BigData)*, 2023, pp. 237–246.

[5] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella, "Multi-resource packing for cluster schedulers," in *Proceedings of the ACM SIGCOMM*, 2014, pp. 455–466.

[6] H. Li, S. Wang, G. Tan, and X. Duan, "Cost-efficient and topology-aware scheduling algorithms in distributed stream computing systems," *Future Generation Computer Systems*, vol. 179, p. 108340, 2026.

[7] A. Verma, L. Pedrosa, M. Korupolu, E. Oppenheimer, David Tune, and J. Wilkes, "Large-scale cluster management at Google with Borg," in *Proceedings of the European Conference on Computer Systems (EuroSys)*, 2015, pp. 1–18.

[8] F. Hu, Q. Fu, S. Zhang, and J. Huang, "A multi-agent deep reinforcement learning-based task offloading method for 6g-enabled internet of vehicles with cloud-edge-device collaboration," *Computers, Materials and Continua*, vol. 87, no. 3, 2026.

[9] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2021.

[10] J. Park, B. Choi, C. Lee, and D. Han, "Graf: A graph neural network based proactive resource allocation framework for slo-oriented microservices," in *Proceedings of the 17th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT)*, 2021.

[11] L. Cheng, X. Li, J. Wang, J. Hu, and H. Zhang, "Collaborative learning-based scheduling for kubernetes-oriented edge-cloud network," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 10, pp. 2785–2801, 2023.

[12] S. Deng, H. Zhao, B. Huang, C. Zhang, F. Chen, Y. Deng, J. Yin, S. Dustdar, and A. Y. Zomaya, "Cloud-native computing: A survey from the perspective of services," *IEEE Access*, vol. 12, pp. 23 307–23 336, 2024.

[13] B. Sedlak, P. Raith, A. Morichetta, V. Casamayor Pujol, and S. Dustdar, "Multi-dimensional autoscaling of stream processing services on edge devices," *IEEE Transactions on Services Computing*, 2025, early Access.

[14] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache flink: Stream and batch processing in a single engine," *IEEE Data Engineering Bulletin*, vol. 38, no. 4, pp. 28–38, 2015.

[15] M. A. Hady, A. S. El-Kassas, H. A. Ahmed, and A. B. El-Sisi, "Multi-agent reinforcement learning for resources allocation optimization: A survey," *Artificial Intelligence Review*, vol. 58, no. 11, pp. 1–45, 2025.

[16] A. Khanna, D. Negi, A. Sah, B. K. Dewangan, and G. V. Londhe, "Carbon-aware microservices scheduling: Machine learning for sustainable cloud-native applications," in *In 2025 4th International Conference on Applied Artificial Intelligence and Computing (ICAAIC)*, 2025, pp. 1501–1506.

[17] S. Dhara, S. Salma, S. Sonal, N. Kumar, and A. K. Singh, "Dynamic resource allocation using reinforcement learning in edge-cloud environments," in *Proceedings of the 3rd International Conference on Data Science and Network Security (ICDSNS)*. IEEE, 2025, pp. 1–7.

[18] K. Addo, K. Moloi, M. Kabeya, and E. E. Ojo, "A survey of machine learning techniques for optimal capacitor placement and sizing in smart distribution networks," *IEEE Access*, vol. 13, pp. 96 933–96 962, 2025.

[19] A. D. H. Abayo, K. K. I. Michael, E. P. F. Sedeon, and O. M. S. Dorian, "Intelligent and secure communication systems: Signal processing, AI, and cloud-enabled architectures for next-generation networks," *SSRN / IEEE Review*, 2025.

[20] Q. Liu, J. Yang, and Z. Yan, "Dynamic resource orchestration in edge computing environments using multi-agent reinforcement learning," *Knowledge and Information Systems*, vol. 67, pp. 9363–9383, 2025.