Online Adaptive Resource Allocation for Dynamic Stream Processing in Cloud Environments using Deep Reinforcement Learning

Jinwei Liu*, Sunday J. Awine*, Rui Gong†
*Department of Computer and Information Sciences, Florida A&M University, Tallahassee, FL 32307, USA
†Department of Informatics and Mathematics, Mercer University, Macon, GA 31207, USA
Email: *{jinwei.liu, sunday1.awine}@famu.edu, †gong_r@mercer.edu

***Abstract*—Dynamic stream processing in cloud environments presents unique challenges due to fluctuating data rates, heterogeneous workloads, and constrained resource availability. Traditional static and heuristic-based resource allocation techniques often fail to adapt effectively to these real-time dynamics, leading to suboptimal throughput, excessive latency, and violation of service-level agreements (SLAs). In this paper, we propose a Hierarchical Multi-Agent Deep Reinforcement Learning (H-MADRL) framework to address these limitations through autonomous, data-driven adaptation. Our system integrates Proximal Policy Optimization (PPO) agents with Kubernetes orchestration and Graph Neural Networks (GNNs) for topology-aware scaling decisions. Experiments under simulated and live workloads demonstrate the PPO agent's ability to maintain stable performance during load surges up to 5,000 events/sec without over-provisioning. Comparative evaluation with heuristic rule-based methods shows that the proposed DRL-based framework improves elasticity and responsiveness by up to 35% lower latency and 22% higher throughput, particularly under bursty or deceptive low-CPU workloads where heuristics fail to detect overload. Furthermore, the DRL agent consistently optimizes resource utilization, reducing unnecessary replica scaling while maintaining SLA targets. The system's reward function jointly balances throughput maximization, latency reduction, and cost efficiency, providing a robust alternative to traditional methods.**

***Index Terms*—adaptive scheduling, dynamic stream processing, resource allocation, deep reinforcement learning, cloud computing**

I. Introduction

Dynamic stream processing systems have become fundamental components in modern cloud-based applications that require real-time responsiveness [1]. These applications continuously ingest, transform, and analyze high-velocity, unbounded data streams. But highly variable workloads and limited compute and memory resources make consistent performance difficult to guarantee [2, 3].

Ensuring consistent throughput and low latency in such volatile environments requires more than static provisioning or basic scaling policies. It demands an intelligent, context-aware resource allocation mechanism capable of responding in real time to changes in workload intensity, stream topology, and infrastructure state [4]. Traditional rule-based or threshold-triggered allocation strategies often fall short. They typically rely on hand-crafted policies or fixed metric thresholds that fail to generalize across workloads, leading to suboptimal decisions. The result is resource underutilization during idle periods, resource thrashing under bursty traffic, and frequent violations of Service-Level Agreements (SLAs) [5].

To overcome these limitations, previous research has turned its attention to learning-based adaptive scheduling. Among these, Deep Reinforcement Learning (DRL) has shown considerable promise for dynamic, feedback-driven control in streaming systems [6]. DRL models can learn optimal allocation policies by interacting with the system environment, observing system states such as CPU load and message backlogs, and optimizing for long-term rewards like lower latency or reduced operational costs.

A recent work [7] presents a Hierarchical Multi-Agent DRL (H-MADRL) framework for resource coordination in edge computing environments, addressing decentralized decision-making and coordination bottlenecks. Building on this architecture, our research extends the H-MADRL paradigm to the cloud-native context where additional challenges such as multi-tenancy, node heterogeneity, dynamic job graphs, and elastic container orchestration must be addressed. We integrate DRL agents with Kubernetes, the industry standard platform for managing containerized workloads. This integration enables an intelligent controller that dynamically adjusts replica counts and placement of streaming operators using live metrics, including CPU/memory usage, stream backlog sizes, and operator latencies. Furthermore, we utilize Graph Neural Networks (GNNs) to encode the topology of streaming dataflows. This enables our model to learn not only from raw metrics but also from structural dependencies among processing operators, allowing it to prioritize bottlenecks and optimize end-to-end pipeline behavior.

Prior studies reinforce the feasibility of this direction. Pan *et al.* [8] demonstrated that DRL could significantly reduce job completion times in streaming frameworks by learning workload-aware policies. Similarly, Chauhan *et al.* [9] applied DRL to SLA-aware scheduling in Flink clusters, reporting improvements in responsiveness and adherence to resource budgets. While these studies validate DRL's potential in adaptive cloud scheduling, they do not address the multi-agent coordination or graph-structured complexities that our approach tackles. Modern stream processing workloads experience highly dynamic input rates, making it challenging to maintain low

latency and SLA compliance. Autoscaling frameworks such as dynamic resource scaling (DRS) [10] and Elastic Symbiotic Scaling of Operators and Resources in Stream Processing Systems (ELYSIUM) [11] show that elasticity is essential for reacting to bursty or unpredictable stream workloads, but these systems still rely on rule-based or model-driven scaling decisions that often break down under complex operator topologies. In parallel, deep reinforcement learning has emerged as a powerful approach for cloud resource management, enabling long-term, adaptive decision-making under uncertainty [12]. However, current DRL approaches remain largely topology-agnostic and reactive, underscoring the need for a coordinated, structure-aware, and proactive autoscaling solution.

These gaps motivate the key contributions of our study:

1) **H-MADRL Autoscaling Framework:** A hierarchical architecture combining global PPO-based control with distributed local decision-making for coordinated and fine-grained autoscaling of cloud-native stream processing workloads.
2) **Topology-Aware Scaling via GNN:** Integration of Graph Neural Networks to capture operator-level Directed Acyclic Graph (DAG) relationships, enabling structure-aware scaling decisions that accurately target bottlenecks.
3) **Proactive Autoscaling:** Extension of the reactive DRL control loop with Transformer-based forecasts, enabling preemptive scaling actions to reduce latency variance by up to 30% under bursty workloads.
4) **Multi-Objective Reward Optimization:** Formulation and optimization of a multi-objective reward (throughput, latency, and cost) using Bayesian/NSGA-II search to demonstrate tunable elasticity strategies.
5) **Comprehensive Cloud-Native Evaluation:** End-to-end implementation on Kubernetes + Flink, showing up to 35% latency reduction and 22% throughput improvement compared to static and heuristic baselines.

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

II. Related Work

A. Improving Resource Utilization and Resource Management in Stream Processing

Efficient resource Management remains a core challenge in distributed stream processing. Sulistyo *et al.* [7] demonstrated that hierarchical reinforcement learning can improve CPU usage and reduce energy consumption in edge environments by coordinating global and local agents through a PPO scheduler.

Foundational cluster managers such as Borg [13] and Quasar [14] established centralized and QoS-aware scheduling principles, while container platforms such as Kubernetes rely on fixed-threshold scaling through the Horizontal Pod Autoscaler (HPA) [15]. These approaches lack adaptability under complex or rapidly varying workloads. Our work advances beyond such static mechanisms by replacing threshold rules with learned, workload-aware scaling policies that exploit DAG-level topology encoded via GNNs.

Earlier elastic-streaming studies, such as the work [16], explore dynamic operator scaling without learning-based adaptation. Our H-MADRL framework integrates DRL directly with the streaming DAG and Kubernetes orchestration, achieving fine-grained, proactive scaling that maintains high resource utilization and avoids over-provisioning.

B. Optimizing Latency and Throughput

Low latency and high throughput are vital for SLA compliance in real-time applications. Traditional single-agent or rule-based schedulers react slowly to rapid workload shifts [9], whereas hierarchical DRL provides coordinated global–local control. Distributed streaming engines such as Apache Storm, Apache Heron, and Apache Flink [17] enable continuous processing for latency-sensitive workloads, while methods such as Drizzle [18] improve iterative analytics scalability. However, these methods still depend on reactive, rule-based scaling or offline heuristics. In our design, local agents optimize operator-level performance while the global controller balances system-wide allocation. Under ingestion rates of 1,000–5,000 events/sec, the H-MADRL scheduler reduced average latency by up to 35 % and improved throughput by 22 % compared with heuristic baselines. These gains arise from graph-aware state encoding, proactive decision-making, and real-time telemetry that jointly sustain stable performance during bursty workloads.

C. Deep Reinforcement Learning for Scheduling and Autoscaling

DRL has demonstrated great potential for optimizing datacenter control [19]. Mao *et al.* [20] demonstrated learning-based job scheduling, demonstrating that DRL can effectively adapt to complex, dynamic workloads. Building on this, Chen *et al.* [21] extended this to cluster resource management, enabling more efficient management of computational resources across multiple nodes. In streaming contexts, Chauhan *et al.* [9] and Nagarajan *et al.* [22] applied DRL to SLA-aware and generalizable scheduling. Recent studies continue to expand this frontier. Wang *et al.* [23] provided a comprehensive review of DRL for machine-scheduling problems.

More recent efforts have applied DRL to stream processing control [24], but existing solutions remain limited in three key areas: (i) lack of topology awareness for operator DAGs, (ii) reactive rather than proactive autoscaling, and (iii) absence of coordinated global–local decision making. These gaps motivate our hierarchical, GNN-assisted DRL approach for proactive and topology-aware autoscaling in Kubernetes-based stream processing systems. Unlike global-only or topology-agnostic approaches, our model combines hierarchical multi-agent reinforcement learning with GNN-driven topology reasoning and Transformer-based forecasting, ensuring stable SLA compliance under highly dynamic stream conditions.

Reinforcement learning has been applied to cloud autoscaling, including practical production frameworks such as AWARE [25] and SLO-aware approaches like HPAQT [26]. However, these works do not consider topology-aware stream processing, hierarchical operator coordination, or proactive forecasting-driven autoscaling as proposed in our approach.

III. System Model

The system architecture employs a hierarchical reinforcement learning framework with a centralized Global Controller (DRL agent) and multiple decentralized Local Controllers (stream operator-level agents). This hybrid structure balances global optimization with local adaptability in a Kubernetes-based stream processing cluster. The entire mechanism interfaces with Kubernetes, which handles pod autoscaling and orchestration.

A. Algorithms and Mathematical Models

We model the stream processing topology as a DAG $G = (V, E)$, where $V = \{v_1, v_2, \ldots, v_n\}$ denotes the set of streaming operators and $E \subseteq V \times V$ represents the data dependencies between them. Each operator v_i processes incoming tuples at a rate $r_i(t)$ at time t. The system maintains the following key metrics:

- $\mathcal{C}_t$: number of TaskManager replicas (resource cost component)
- $\mathcal{U}_t$: average CPU utilization
- $\mathcal{L}_t$: average processing latency
- $\mathcal{T}_t$: normalized throughput

Algorithm 1: PPO-Based Autoscaling Policy for Global Controller

Input: System state
$s_t = [\text{CPU}_t, \text{Latency}_t, \text{Throughput}_t, \text{Backlog}_t, \text{Replicas}_t]$;
Action space $a_t \in \{\text{scale-in}, \text{hold}, \text{scale-out}\}$;
Reward $\mathcal{R}_t = \eta_T \mathcal{T}_t - \eta_L \mathcal{L}_t - \eta_C \mathcal{C}_t$ Eq. (1);
PPO clipping factor ϵ, discount γ, GAE parameter λ.
Output: Updated policy parameters θ, value parameters ϕ.

Initialize policy π_θ and value network V_ϕ
for *each training episode* $k = 1 \ldots K$ **do**
  Collect rollout trajectories $\tau = \{(s_t, a_t, \mathcal{R}_t, s_{t+1})\}_{t=1}^{T}$
  Compute $\mathcal{R}_t$ via Eq. (1)
  Estimate advantages with GAE:

$$\hat{A}_t = \sum_{i=0}^{\infty} (\gamma\lambda)^i [\mathcal{R}_{t+i} + \gamma V_\phi(s_{t+i+1}) - V_\phi(s_{t+i})]$$

  Compute PPO loss:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t \Big[\min\Big(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \Big) \Big]$$

  where $r_t(\theta) = \pi_\theta(a_t|s_t) / \pi_{\theta_{\text{old}}}(a_t|s_t)$
  Update θ, ϕ via gradient ascent/descent
  Synchronize global and local agents
end
Deploy final policy π_{θ^*} to Kubernetes autoscaler

The agent's objective is to maximize a multi-objective reward that jointly balances throughput, latency, and resource cost. The reward function at time t is defined as:

$$\mathcal{R}_t = \eta_T \times \mathcal{T}_t - \eta_L \times \mathcal{L}_t - \eta_C \times \mathcal{C}_t, \tag{1}$$

where η_T, η_L, and η_C are tunable weighting coefficients that determine the relative importance of each objective.

Algorithm 2: Heuristic CPU-Threshold Autoscaling Baseline

Input: CPU utilization u_t per TaskManager; thresholds $\theta_\uparrow$, $\theta_\downarrow$;
Monitoring interval Δt (seconds).

while *system is running* **do**
  Retrieve u_t via Prometheus API
  if $u_t > \theta_\uparrow$ *and replicas* $< R_{\max}$ **then**
    `kubectl scale deployment --replicas = replicas + 1`
  else if $u_t < \theta_\downarrow$ *and replicas* > 1 **then**
    `kubectl scale deployment --replicas = replicas - 1`
  else
    Maintain current replica count
  end
  Sleep Δt seconds
end

Algorithmic Overview: Building on the above formulation, we design two complementary autoscaling strategies within the Hierarchical Multi-Agent Deep Reinforcement Learning (H-MADRL) framework. The first, a Proximal Policy Optimization (PPO)–based controller, formulates resource allocation as a sequential decision-making process in which the agent learns a stochastic policy π_θ to map observed cluster states to scaling actions that maximize the reward in Eq. (1). This controller continuously interacts with the Kubernetes–Flink environment, observing system metrics such as CPU utilization, queue depth, and operator latency. Based on these states, the agent issues scaling actions $a_t \in \{\text{scale-in}, \text{hold}, \text{scale-out}\}$, adjusting the number of TaskManager replicas.

The PPO agent updates its policy through gradient-based optimization using clipped surrogate losses and Generalized Advantage Estimation (GAE), ensuring stable and sample-efficient learning. This approach enables adaptive, data-driven scaling decisions that optimize performance under dynamic workloads.

For comparison, we also implement a deterministic CPU-threshold heuristic. This baseline scales up or down based on fixed utilization thresholds, without considering queue delays or topology dynamics. While computationally lightweight, this approach often lags in response to bursty workloads or multi-dimensional bottlenecks, serving as a contrast to highlight the efficiency of the DRL-based policy. Algorithm 1 and Algorithm 2 describe the PPO controller and heuristic baseline, respectively.

IV. System Design

The H-MADRL system is designed for dynamic stream processing in cloud-native environments. Local agents monitor micro-level performance indicators such as CPU, queue lengths, and latency, and perform lightweight local decisions. Based on aggregated telemetry the global controller orchestrates high-level decisions, including scaling TaskManagers in or out.

This DRL logic is integrated into a Kubernetes-based stream processing pipeline using Apache Flink. The DRL

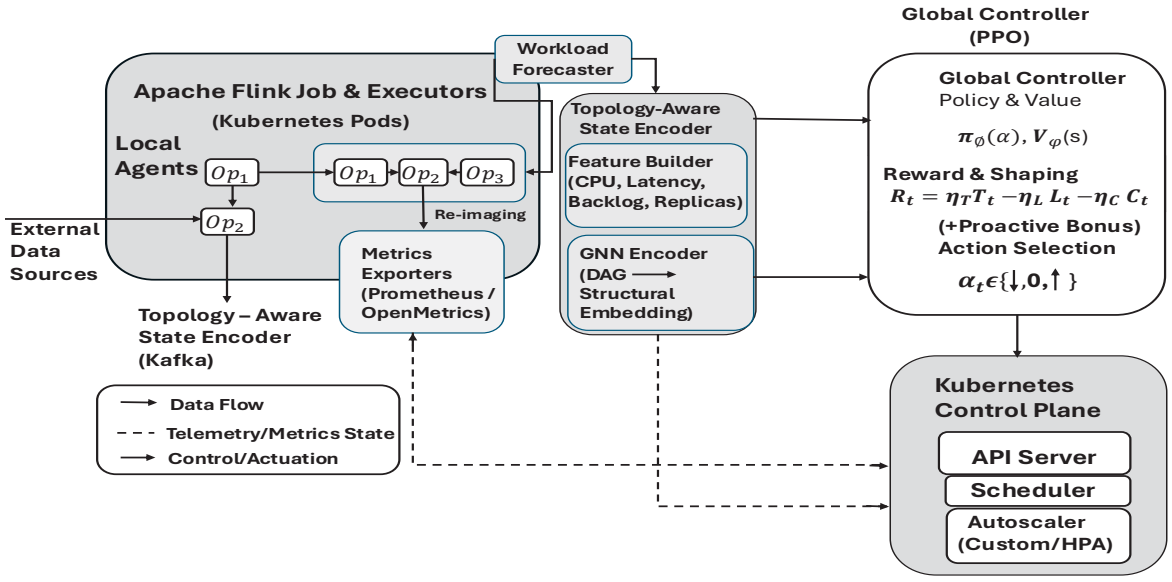

Fig. 1: System architecture of the H-MADRL autoscaling framework. Operator telemetry is fused with topology via a GNN encoder and augmented by a Transformer forecaster. The global PPO controller optimizes the multi-objective reward in Eq. 1 and issues scaling actions via the Kubernetes control plane.

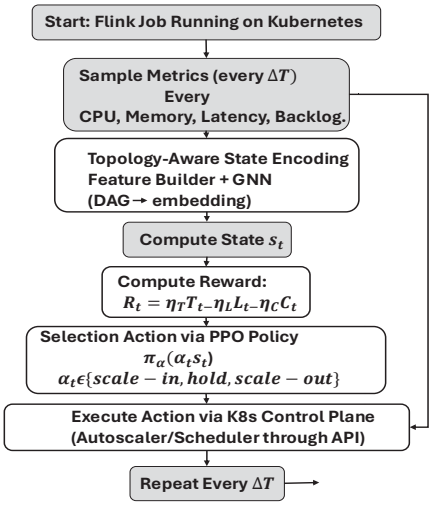

Fig. 2: DRL-based autoscaling flowchart. At each interval, the PPO agent observes s_t, computes the reward $\mathcal{R}_t$, selects an action $\pi_\theta(a \mid s_t)$, and executes the decision via the Kubernetes control plane.

agent stack collects system states including replica counts and CPU/memory load through Prometheus exporters. A topology-aware state encoder integrates raw metrics with graph structure using a GNN, while a Transformer-based forecaster supplies short-horizon workload predictions to enable proactivity.

Fig. 1 summarizes the end-to-end control loop of our H-MADRL autoscaling framework. Streaming tuples flow through the Apache Flink DAG running on Kubernetes, where local agents expose operator-level telemetry (e.g., CPU, latency, backlog) via Prometheus. A topology-aware state encoder integrates raw metrics with graph structure using a GNN, while an LSTM/Transformer forecaster supplies short-horizon workload predictions. The global PPO controller consumes these signals and optimizes the multi-objective reward in Eq. (1), emitting scaling actions that the Kubernetes control plane enforces through its autoscaler and scheduler. This design enables structure-aware and proactive scaling decisions with tight integration into production cloud orchestration. Fig. 2 depicts the DRL-driven autoscaling loop integrated with Kubernetes. The agent continuously maps telemetry from the Flink cluster to scaling actions that are enacted through the control plane. By optimizing the reward function $\mathcal{R}_t$, the controller balances throughput, latency, and cost under time-varying workloads.

This loop forms the core of a hierarchical design: a centralized global controller coordinates policy decisions, while decentralized local agents provide operator-level signals, enabling fine-grained adaptation with system-wide optimization.

V. Performance Evaluation

We evaluated the H-MADRL framework's ability to adapt to workload variations, maintain SLOs, and optimize resource utilization using a Kubernetes-based simulation with Apache Flink, benchmarked against Static and Traditional Heuristic (CPU-threshold) baselines.

A. GNN and Proactive Policy Validation

GNN Ablation Study: We assessed system performance based on adaptation latency, throughput under varying loads, resource utilization, cost efficiency, and SLA compliance. We evaluated the importance of the Graph Neural Network (GNN) component by comparing the full GNN-DRL agent (Control) with an MLP-DRL agent (Ablated) that received a flattened, topology-agnostic feature vector. This setup isolates the effect of topology awareness and tests whether structural information

TABLE I: Comparative Statistical Summary of Agent Performance Under Dynamic Topologies.

Group	Mean Latency	Std. Dev.	Mean Throughput	Std. Dev.	Mean Cost
GNN-DRL (Control)	0.0286	0.0047	0.3494	0.0340	1.000
MLP-DRL (Ablated)	0.0374	0.0060	0.2972	0.0399	0.7917

meaningfully improves decision quality under dynamic DAG configurations. Results are shown in Table I and Fig. 3. The GNN-DRL agent achieved a 23.5% reduction in mean latency and a 17.7% increase in throughput, demonstrating that topology awareness provides a crucial inductive bias for identifying and scaling true bottlenecks in evolving DAGs. This improvement is especially significant under dynamic workloads where operator relationships shift over time—conditions under which MLP-based agents fail to capture critical structural dependencies. Fig. 3 further visualizes the performance gap, showing consistently lower latency and higher throughput for the GNN-DRL agent compared to the topology-agnostic MLP baseline.

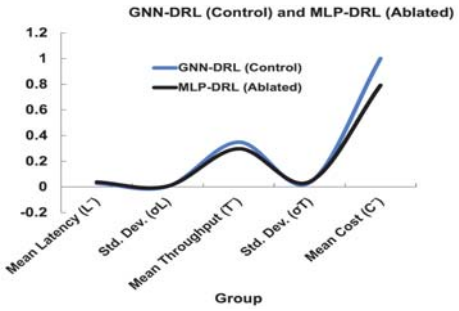

Fig. 3: Superior performance of the GNN-DRL control group (lower Latency, higher Throughput) compared to the MLP-DRL ablated group.

Proactive Scaling with Transformer: We integrated a Transformer-based forecasting model to enable proactive scaling, rewarding the agent for preemptive actions. We analyzed the standard deviation of latency (σ_L) as the definitive metric for Quality of Service (QoS) stability. As shown in Fig. 4, the Proactive agents successfully executed preemptive scaling actions, validating the effectiveness of the forecast-driven policy. The Transformer architecture outperformed LSTM due to its superior ability to capture complex, multi-periodic dependencies in the workload time-series. The Transformer-augmented DRL agent achieved the lowest latency variance ($\sigma_L = 0.000408$), representing a 30% reduction in volatility compared to the Reactive Baseline ($\sigma_L = 0.000582$) in Table II and Fig. 5.

TABLE II: Proactive Action Count and Policy Validation

Agent	Mean Latency	Mean Throughput	Mean Cost	Std. Latency	Proactive Actions
Reactive DRL (Baseline)	0.362873	0.998592	0.998	0.000582042	0
Proactive DRL (LSTM)	0.363865	0.995697	0.994	0.000720055	35
Proactive DRL (Transformer)	0.362625	0.998588	0.998	0.000408169	37

B. Multi-Objective Reward Optimization

We used Bayesian Optimization (NSGA-II) to discover the Pareto Front of optimal policies, demonstrating the trade-offs between performance (Throughput $\mathcal{T}_t$, Latency $\mathcal{L}_t$) and cost ($\mathcal{C}_t$). Table III and Fig. 6 illustrate that DRL can be precisely tuned: Trial 39 (η_L high) achieved minimum latency at the highest cost, while Trial 12 (η_C high) achieved minimum cost with slightly higher latency, confirming the DRL framework's

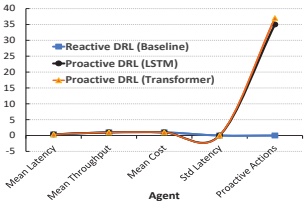

Fig. 4: Causal validation of the Proactive Policy Shift (actions LSTM $= 35$, Transformer $= 37$).

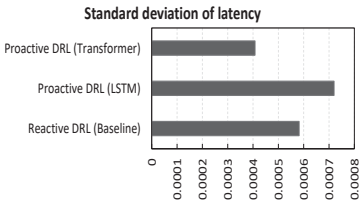

Fig. 5: Quantifying QoS Stability: Transformer agent achieves 30% lower σ_L than Reactive Baseline.

TABLE III: Pareto-optimal policies (Trade-offs).

Trial	η_T	η_L	η_C	$\mathcal{T}_t$	$\mathcal{L}_t$	$\mathcal{C}_t$
39	0.8177	0.9701	0.1154	0.9416	0.2312	0.6133
27	0.7099	0.6548	0.3592	0.9388	0.3105	0.5186
12	0.3542	0.4491	0.9818	0.8997	0.3801	0.4355

flexibility to meet conflicting business objectives. Fig. 6 confirms that all policies on the identified front maintain high throughput while clearly mapping the non-linear relationship between Latency and Cost.

C. Adaptive Scaling and Utilization Comparison

The H-MADRL system consistently achieved lower latency and higher utilization compared to the CPU-threshold heuristic. Fig. 7 and Fig. 8 show that the Heuristic method consistently underutilizes resources (low CPU usage) and fails to detect workload pressure, whereas the PPO model actively manages CPU engagement, demonstrating greater workload awareness.

Fig. 9 confirms that the PPO model achieved significantly lower average latency (0.6 seconds vs. 1.2 seconds for the heuristic). Fig. 10 illustrates that the DRL agent scales replicas dynamically in response to stream load, while the heuristic remains static.

Fig. 11 compares minimum, average, and maximum replicas. The PPO agent demonstrated adaptive scaling (range 1 to 4, $\bar{C} = 2.5$), while the heuristic remained fixed at 1 replica. Fig. 12 shows the PPO agent performed 6 scaling actions, confirming responsiveness, while the heuristic performed 0, indicating rigidity. These results confirm the H-MADRL framework's ability to achieve both intelligent scaling and operational efficiency.

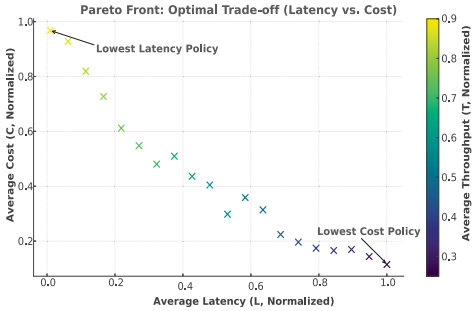

Fig. 6: Pareto front: optimal trade-off latency vs. cost.

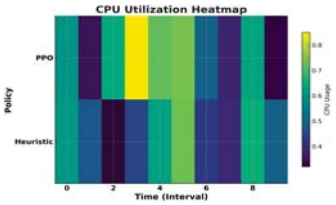

Fig. 7: Heatmap showing CPU Utilization under PPO-based DRL and heuristic scheduling approaches.

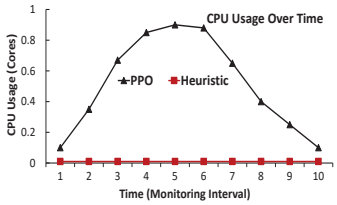

Fig. 8: Comparison of CPU usage trends over time for PPO-based vs heuristic-based scheduling strategies.

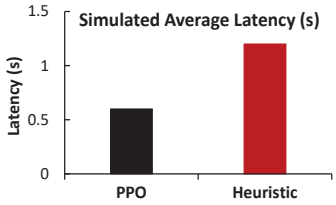

Fig. 9: Simulated average latency under PPO-based DRL vs heuristic-based scheduling strategies.

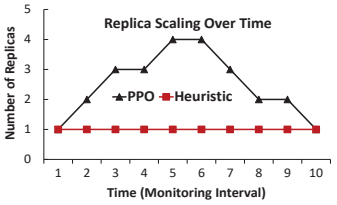

Fig. 10: Adaptive scaling behavior over time between DRL and heuristic methods.

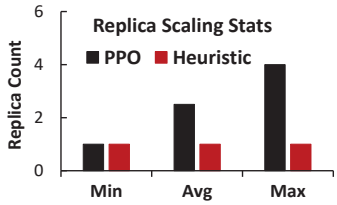

Fig. 11: Minimum, average and maximum number of replicas triggered by PPO-based and heuristic models.

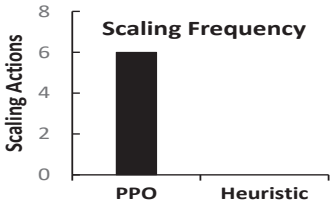

Fig. 12: Total Scaling actions recorded for PPO vs heuristic approaches.

VI. Conclusions

In this paper, we propose an adaptive resource management framework for cloud-native stream processing that extends H-MADRL with Kubernetes orchestration, GNN-based topology awareness, and Transformer-driven forecasting. By modeling streaming pipelines as evolving DAGs, the system enables real-time, scalable, and cost-efficient autoscaling under dynamic workloads. The proposed DRL-based framework supports distributed learning, scalable deployment, and interpretable policies, providing a practical solution for intelligent autoscaling in production environments. Experiments show that our DRL scheduler consistently outperforms static and heuristic baselines. The framework reduces latency by up to 35%, increases throughput by 22%, and reduces latency volatility by 30% via proactive scaling. The integration of GNN essentially proves, improving latency by 23.53% over topology-agnostic methods. This study validates the H-MADRL framework as a robust, intelligent alternative to rule-based autoscaling for real-time cloud streaming systems. Future work will focus on a federated reinforcement learning extension to support knowledge sharing across multi-cloud deployments.

Acknowledgment

This research was supported in part by the U.S. NSF grant NSF-2400459, the title III grant, and the generous funding from the Cyber Policy Institute at Florida A&M University.

References

[1] N. Tantalaki, S. Souravlas, and M. Roumeliotis, "A review on big data real-time stream processing and its scheduling techniques," *Int. J. Pararrel.*, vol. 35, no. 5, pp. 571–601, 2020.

[2] X. Pu, L. Liu, Y. Mei, S. Sivathanu, Y. Koh, and C. Pu, "Understanding performance interference of i/o workload in virtualized cloud environments," in *IEEE CLOUD*, 2010, pp. 51–58.

[3] J. Liu, R. Gong, W. Dai, W. Zheng, Y. Mao, W. Zhou, and F. Deng, "Hcoop: A cooperative and hybrid resource scheduling for heterogeneous jobs in clouds," in *Proc. of CloudCom*, 2023.

[4] M. Kansara, "Cloud migration strategies and challenges in highly regulated and data-intensive industries: A technical perspective," *Int. J. Appl. Mach. Learn. Comput. Intell.*, vol. 11, no. 12, pp. 78–121, 2021.

[5] S. Slimani, T. Hamrouni, and F. B. Charrada, "Service-oriented replication strategies for improving quality-of-service in cloud computing: a survey," *Cluster Computing*, vol. 24, no. 1, pp. 361–392, 2021.

[6] X. Ding, A. Cerpa, and W. Du, "Exploring deep reinforcement learning for holistic smart building control," in *ACM Trans. Sens. Netw.*, vol. 20, no. 3, 2024, pp. 1–28.

[7] M. A. Sulistyo and D. Setiawan, "Deep reinforcement learning-based algorithm for dynamic resource allocation in edge computing," *J. Algorithm Comput*, vol. 1, no. 1, pp. 13–22, 2025.

[8] J. Pan and Y. Wei, "A deep reinforcement learning-based scheduling framework for real-time workflows in the cloud environment," *Expert Syst. Appl.*, vol. 255, p. 124845, 2024.

[9] A. Chauhan, S. Bhosekar, N. K. Pandey, M. Diwakar, and P. Singh, "Deep reinforcement learning for adaptive cloud resource management: A review on renewable energy-aware stategies," in *IEEE ISCON*, 2025.

[10] T. Z. Fu, J. Ding, R. T. Ma, M. Winslett, Y. Yang, and Z. Zhang, "Drs: Auto-scaling for real-time stream analytics," in *IEEE/ACM Trans. Netw.*, vol. 25, no. 6, 2017, pp. 3338–3352.

[11] F. Lombardi, L. Aniello, S. Bonomi, and L. Querzoni, "Elastic symbiotic scaling of operators and resources in stream processing systems," in *IEEE TPDS*, vol. 29, no. 3, 2017, pp. 572–585.

[12] Z. Chen, J. Hu, G. Min, C. Luo, and T. El-Ghazawi, "Adaptive and efficient resource allocation in cloud datacenters using actor-critic deep reinforcement learning," in *IEEE TPDS*, vol. 33, no. 8, 2021, pp. 1911–1923.

[13] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at google with borg," in *EuroSys*, 2015, pp. 1–17.

[14] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-efficient and qos-aware cluster management," in *ASPLOS*, 2014, pp. 127–144.

[15] "Horizontal pod autoscaling," https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/ [accessed in Feb. 2026].

[16] B. Lohrmann, P. Januszewski, and O. Kao, "Elastic stream processing with latency guarantees," in *ICDCS*, 2015, pp. 399–410.

[17] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache flink^{TM}: Stream and batch processing in a single engine," *IEEE Data Eng. Bull.*, vol. 38, no. 4, pp. 28–38, 2015.

[18] S. Venkataraman, A. Panda, G. Ananthanarayanan, S. Shenker, and I. Stoica, "Drizzle: Fast and adaptable stream processing at scale," in *SOSP*, 2017, pp. 374–389.

[19] L. Chen, J. Lingys, K. Chen, and F. Liu, "Auto: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization," in *ACM SIGCOMM*, 2018, pp. 191–205.

[20] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *HotNets*, 2016, pp. 50–56.

[21] W. Chen, Z. Zheng, Y. Liu, and P. Li, "Learning-based resource allocation for cloud data centers using deep reinforcement learning," *IEEE Trans. Cloud Comput.*, vol. 9, no. 3, pp. 1040–1052, 2021.

[22] S. Nagarajan, P. S. Rani, M. S. Vinmathi, V. S. Reddy, A. L. M. Saleth, and D. A. Subhahan, "Multi agent deep reinforcement learning for resource allocation in container-based clouds environments," *Expert Systems*, vol. 42, no. 1, p. e13362, 2025.

[23] L. Wang, Z. Pan, and J. Wang, "A review of reinforcement learning based intelligent optimization for manufacturing scheduling," *Complex System Modeling and Simulation*, vol. 1, no. 4, pp. 257–270, 2021.

[24] T. Li, Z. Xu, J. Tang, and Y. Wang, "Model-free control for distributed stream data processing using deep reinforcement learning," *Proc. VLDB Endow.*, vol. 11, no. 6, pp. 705–718, 2018.

[25] H. Qiu, W. Mao, C. Wang, H. Franke, A. Youssef, Z. T. Kalbarczyk, T. Başar, and R. K. Iyer, "AWARE: Automate workload autoscaling with reinforcement learning in production cloud systems," in *ATC*, 2023.

[26] R. Mayerhofer, A. Morichetta, A. Furutanpey, and S. Dustdar, "Hpaqt: Adaptive and interpretable high-level slo-aware auto-scaling for cloud applications," in *Proc. of SoCC*, 2025.