

Adaptive Resource Allocation in Distributed Systems Using Reinforcement Learning and Graph Neural Networks

Alireza Rahimipour Anaraki
Department of Engineering
CT.C., Islamic Azad University
Tehran, Iran
a.rahimipouranaraki@iau.ir

Parvaneh Asghari
Department of Engineering
CT.C., Islamic Azad University
Tehran, Iran
p.asghari@iau.ac.ir

Abstract—In modern distributed computing environments, efficient resource allocation remains a challenging problem due to dynamic workloads, heterogeneous resources, and complex inter-task dependencies. Traditional scheduling heuristics (e.g., First-Come-First-Served and Round Robin) often fail to adapt to such conditions or to exploit structural information in job execution graphs. This paper presents an integrated framework for adaptive resource allocation that combines reinforcement learning (RL) with graph neural networks (GNNs) to learn scheduling policies in distributed systems. The GNN component encodes graph-structured system state—such as task dependency graphs and resource availability—into compact representations, enabling the RL agent to generalize across varying workloads and system topologies. Based on these representations, the RL agent selects scheduling actions (e.g., task selection or placement) with the objective of optimizing long-term performance metrics, including throughput, job completion time, resource utilization, and fairness. We describe the design of the proposed RL+GNN framework, covering state and action formulation as well as reward design, and provide a simulation-based conceptual evaluation in a cloud/HPC cluster scheduling setting. The results suggest that incorporating graph-based state representations into reinforcement learning can improve the adaptivity and scalability of resource allocation strategies, particularly in environments with complex dependency structures and dynamic behavior.

Keywords— Reinforcement Learning, Graph Neural Networks, Distributed Systems, Resource Allocation, Cloud Computing, High-Performance Computing (HPC)

I. INTRODUCTION

Efficient resource allocation is a critical concern in distributed systems such as cloud computing platforms and high-performance computing (HPC) clusters. The goal is to assign tasks or workloads to computing resources (e.g. servers, virtual machines, network links) in a way that optimizes performance metrics like throughput, latency, or resource utilization. This problem is known to be computationally hard (often NP-hard) in general, and in practice it is further complicated by the dynamic and heterogeneous nature of modern distributed environments. Workloads can vary over time, resource capacities differ, and tasks may have complex inter-dependencies (e.g. parallel jobs consisting of dependent tasks). Traditional resource scheduling algorithms rely on simple heuristics or static policies that are suboptimal in complex and changing environments. For example, common strategies like First-Come-First-Served (FCFS), Shortest Job First (SJF), or Round Robin do not adapt to workload changes or consider detailed job structures, leading to inefficient utilization under diverse workloads. In particular, these methods exhibit several

limitations, such as inability to adapt to shifting demand patterns, ignorance of task dependency graphs, and poor handling of heterogeneous resource capabilities. Improving upon these legacy approaches requires more intelligent, context-aware scheduling mechanisms.

Reinforcement Learning (RL) has emerged as a promising approach for adaptive resource allocation in distributed systems. In an RL-based scheduler, a software agent learns a scheduling policy through trial-and-error interactions with the system environment, aiming to maximize long-term rewards. Prior works have shown that RL can automatically learn sophisticated scheduling decisions that outperform fixed heuristics. For instance, Mao et al.'s Decima scheduler used RL to learn scheduling algorithms for data processing clusters, achieving better job completion times than hand-tuned policies. An RL agent can observe the system state (e.g. queue lengths, task DAGs, resource usage) and iteratively improve its policy to optimize a given objective (such as minimizing average job slowdown or maximizing throughput). One advantage of RL is its ability to continually adapt to changes: as workloads or resource conditions evolve, the agent can adjust its decisions based on learned experience rather than relying on pre-defined rules. However, applying RL to complex distributed systems also poses challenges – notably, how to represent the system's state in a form that an RL algorithm can understand and generalize from. Naively encoding all details of a large cluster or complex job graph into an RL state may be infeasible or may not generalize well to different scenarios.

Graph Neural Networks (GNNs) offer a powerful solution for modeling complex graph-structured data in distributed systems, such as the dependency graph of tasks in a job or the topology of an underlying network. GNNs can learn representations (embeddings) for nodes or entire graphs by aggregating information from graph neighbors in multiple layers. This capability aligns well with distributed system scenarios: for example, in a job scheduling context, each job can be represented by a directed acyclic graph (DAG) of tasks, and a GNN can encode the DAG's structural information (dependencies, critical path, etc.) into a fixed-length vector that captures the job's characteristics. Similarly, in a network resource allocation scenario, the network of computing nodes (with links representing bandwidth or interference relationships) can be represented as a graph; a GNN can then extract features that summarize global connectivity and node status. By incorporating GNNs, an RL agent can handle variable-sized and dynamic system states, generalizing to scenarios with different numbers of tasks or machines because

graph networks do not require a fixed input size. Recent research confirms that combining GNNs with deep RL improves generalization to unseen topologies and changing environments. For example, Han et al. (2024) demonstrate a GNN-based RL framework in a computing network that adapts to topology changes and outperforms traditional RL that lacks graph awareness. The GNN essentially acts as a trainable function to embed the system's state graph into a latent vector, which the RL algorithm can then use for decision-making.

Motivated by these advancements, this paper proposes an Adaptive Resource Allocation framework using Reinforcement Learning enhanced with Graph Neural Networks (RL+GNN). The high-level idea is that the distributed system's state (including information about tasks, resources, and their relationships) is represented as a graph and processed by a GNN to produce a meaningful encoding. This encoding is fed into an RL agent (such as a deep Q-network or policy network) which then decides on an action (for example, assign a task to a particular server, or schedule the next job) that aims to optimize long-term performance. By training this system in simulation, the agent can learn to allocate resources more efficiently over time, adapting to workload variations and exploiting complex dependency information. We expect such an approach to achieve near-optimal scheduling performance in scenarios where heuristic algorithms falter. In the following, we detail the design of the proposed method, including the system modeling, the RL formulation, and the integration of GNN for state representation. We then present an example evaluation in a cloud/HPC setting to illustrate the benefits of the approach. Finally, we discuss the implications of our findings and outline future research directions.

Paper Structure: The remainder of this paper is organized as follows. Section Methodology describes the problem formulation and the components of our RL+GNN approach, including state encoding, the learning algorithm, and reward design. Section Experimental Evaluation provides a simulation scenario and results demonstrating the effectiveness of the proposed approach compared to baselines. Section Conclusion summarizes the contributions and discusses potential future improvements. Finally, a list of references is provided.

II. CONTRIBUTIONS

This paper presents an integrated framework for adaptive resource allocation in distributed computing environments by combining reinforcement learning with graph neural networks. The main contributions of this work are threefold:

- (i) We propose a unified graph-based state representation that jointly models task dependencies and resource availability, enabling the scheduler to reason about complex job structures and system dynamics in a scalable manner.
- (ii) We formulate a flexible action abstraction for scheduling that accommodates variable-sized task graphs and heterogeneous resources, making the learned policy applicable across different cluster configurations without requiring fixed-size state representations.
- (iii) We design and discuss a multi-objective reward formulation that balances throughput, job completion time, resource utilization, and fairness, and provide a conceptual simulation-based evaluation illustrating the potential benefits

of the proposed RL+GNN framework compared to conventional scheduling heuristics.

III. RELATED WORK

Resource allocation and scheduling in distributed systems have been extensively studied using both heuristic and learning-based approaches. Traditional schedulers, such as First-Come-First-Served (FCFS), Shortest Job First (SJF), and backfilling-based algorithms, are widely used in cluster and HPC environments due to their simplicity and low overhead. However, these methods rely on fixed rules and are often unable to adapt effectively to dynamic workloads, heterogeneous resources, or complex task dependencies.

Recent research has explored reinforcement learning (RL) as a means of learning adaptive scheduling policies directly from system feedback. Notable examples include RL-based cluster schedulers that optimize job completion time or throughput by interacting with simulated environments. While these approaches demonstrate improved adaptability compared to static heuristics, many rely on handcrafted state representations or fixed-size inputs, which limit their ability to generalize across varying system scales or job structures.

Graph neural networks (GNNs) have recently emerged as a powerful tool for modeling graph-structured data in distributed systems, such as task dependency graphs and resource topologies. Several works have combined GNNs with RL to improve resource allocation in networking and computing systems, showing enhanced generalization to dynamic or unseen topologies. In contrast to prior studies that focus primarily on either task graphs or resource graphs in isolation, our work adopts a unified graph-based state representation that jointly captures task dependencies and resource availability. This integrated formulation, combined with a flexible action abstraction and multi-objective reward design, provides a practical foundation for adaptive scheduling in distributed computing environments.

While our work shares the general objective of learning adaptive scheduling policies with prior reinforcement learning-based schedulers, there are important distinctions in problem formulation and system modeling. Decima focuses on learning scheduling policies for data-processing clusters by operating primarily on per-job task DAGs and requires a specialized policy structure tailored to those graphs. In contrast, our framework adopts a unified graph-based state representation that jointly models task dependencies and resource availability, enabling more flexible integration of heterogeneous resources and system-level constraints.

Cilantro and related online RL-based resource allocation systems emphasize continuous feedback control and performance isolation across applications, often abstracting away explicit task dependency structures. Our approach instead explicitly incorporates graph-structured dependency information through a GNN encoder, allowing the policy to reason about execution order and dependency-induced bottlenecks.

More recent RL+GNN approaches typically focus on either network resource allocation or isolated task graphs. Our work differs in its emphasis on a general and extensible formulation for distributed computing environments, combining task-level and resource-level information within a single graph representation and illustrating how this

formulation can support scalable and adaptive scheduling decisions.

IV. SYSTEM MODEL AND STATE REPRESENTATION

In reinforcement learning-based scheduling for distributed systems, the agent's performance critically depends on how effectively the environment's state is represented. In our framework, we model the system state at each time step t as a graph structure $G_t = (V_t, E_t)$ Where:

- V_t denotes the set of nodes representing active tasks, available resources, or both,
- E_t captures the relationships among tasks (e.g., precedence constraints), or between tasks and resources (e.g., compatibility, availability, or data transfer links).

Each node in G_t is associated with a feature vector that encodes relevant information such as task length, resource requirements, current status (queued, running, completed), and for resource nodes: capacity, load, and availability. Edges may include features such as bandwidth, execution priority, or dependency type.

To enable generalization across variable job sizes and system configurations, we utilize a Graph Neural Network (GNN) to encode the structured state into a fixed-size embedding vector $h_t = f_{\text{GNN}}(G_t)$. This vector summarizes the global system context in a form suitable for downstream decision-making by the reinforcement learning agent.

We adopt a message-passing neural network (MPNN) structure, where node embeddings are iteratively updated based on aggregated information from their neighbors. The final node embeddings are pooled to form the overall state representation h_t . This approach allows the model to capture complex task dependencies and resource interactions, while remaining invariant to the size or ordering of the input graph.

This graph-based state encoding provides the learning agent with a rich and structured view of the scheduling environment, which is crucial for making context-aware and globally optimal allocation decisions.

V. METHODOLOGY

A central aspect of the proposed framework is the representation of the system state in a structured form that captures both task dependencies and resource conditions. Rather than encoding the state as a fixed-length vector, we model the state at each decision step as a graph, which naturally reflects the relational structure inherent in distributed scheduling problems.

Depending on the application scenario, different graph constructions can be employed. In a job-centric representation, tasks are modeled as nodes in a directed acyclic graph (DAG), where edges represent precedence constraints between tasks. Node features may include task-specific attributes such as remaining execution time, resource requirements, and execution status (e.g., waiting, running, or completed). In a resource-centric or hybrid representation, additional nodes may be used to represent computing resources, with features describing their current load, capacity, and availability. Edges can encode relationships such as task dependencies, task-resource compatibility, or communication constraints.

To enable learning across variable-sized jobs and system configurations, the graph-structured state is processed using a graph neural network (GNN). Through iterative message passing, the GNN aggregates information from neighboring nodes and produces embeddings that summarize local and global system context. These node-level embeddings are subsequently combined using a pooling or readout operation to obtain a fixed-dimensional representation of the current system state. This representation serves as input to the reinforcement learning agent and provides a compact yet expressive description of the scheduling environment.

By adopting a graph-based state representation, the framework can naturally accommodate heterogeneous workloads, dynamic task graphs, and changing resource availability, while avoiding assumptions about fixed system size or static structure.

- **Job Dependency Graph:** In a cluster scheduling scenario, each job can be modeled as a directed acyclic graph (DAG), where nodes represent tasks and edges encode precedence constraints between tasks. Such DAGs capture the execution structure of parallel jobs and naturally form part of the system state. Multiple job DAGs may coexist simultaneously as different jobs progress through the system. Task nodes are associated with features such as estimated execution time, resource requirements, and execution status, while edges represent ordering constraints that restrict task execution. In practice, job-level DAGs can be processed independently and their representations combined using pooling or attention mechanisms, or integrated into a larger graph that also incorporates resource-related information. This formulation allows the scheduler to reason explicitly about task readiness and dependency-induced bottlenecks without assuming a fixed job size or structure.

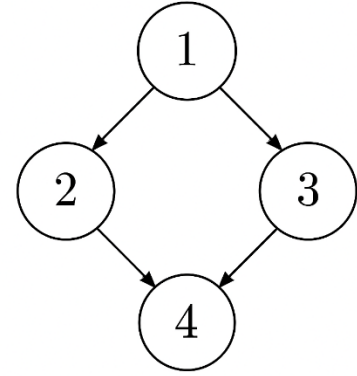


Fig. 1. Example of a task dependency graph (DAG) for a job. Nodes 1–4 represent tasks, and directed edges indicate execution dependencies (e.g., Task 1 must complete before Tasks 2 and 3 can start; Task 4 depends on both 2 and 3). Such graphs are used to represent job structures in the state. GNN-based encoders can convert these graph-structured job descriptions into useful features for decision-making.

- **Resource Topology Graph:** In some scenarios, the physical or virtual resources and their network connections can be represented as a graph. For instance, in an edge computing or cloud network, nodes could represent compute servers (or data centers) and edges represent network links (with weights indicating bandwidth or latency). Tasks or

services might be mapped onto this resource graph. A GNN can be used to encode the current utilization or capacity of this resource network. This is especially useful if the network topology is complex or changes over time. By using a graph representation, the scheduling agent can generalize to different network structures – a known challenge where traditional deep RL methods struggle if the number of nodes changes. Researchers have found that incorporating GNNs allows the learned policy to operate over variable topologies and still perform well, whereas a fixed-size neural network might fail when the environment graph changes.

In our framework, we unify these ideas by considering the state s_t at time t as a graph (or a collection of graphs) $G_t = (V_t, E_t)$ with node features and edge features capturing all relevant information about jobs and resources at that time. For example, each task in each job could be a node, and there could be special nodes or attributes for resource availability; or we could construct bipartite graphs between tasks and machines. The exact graph construction may vary by use case, but the Graph Neural Network will take G_t as input and produce an embedding (a vector representation) of the state. We denote this embedding as $h_t = f_{\text{GNN}}(G_t)$, where f_{GNN} is the learned graph embedding function. This h_t is designed to capture the global context of the current scheduling problem (including task readiness, resource load, etc.) in a fixed-size representation. We employ a GNN architecture based on graph convolution or message passing. A typical layer in a graph convolutional network (GCN) updates node representations by aggregating information from neighboring nodes:

$$H^{(l+1)} = \sigma \left(D^{-\frac{1}{2}} A D^{-\frac{1}{2}} H^{(l)} W^{(l)} + b^{(l)} \right),$$

Where $H^{(l)}$ is the matrix of node features at layer l , A is the adjacency matrix of the graph, D is the degree matrix, $W^{(l)}$ and $b^{(l)}$ are learnable weights and biases, and σ is an activation function. Stacking multiple GCN layers allows information to propagate over multi-hop neighborhoods, so that the final node embeddings $H^{(L)}$ contain information about each node's broader context. We can then pool or readout these node embeddings into a single state embedding h_t . For example, one can sum or average all node embeddings, or use an attention-weighted sum, or even take a specific node (like a special "global" node) to represent the whole graph. This h_t is then fed into the reinforcement learning module.

Action Space: The actions the RL agent can take must be defined according to the resource allocation problem. Different formulations are possible:

- In a job scheduling scenario (single machine or homogeneous cluster), an action might be selecting which pending job/task to execute next (essentially ordering the queue). In a multi-node cluster, an action could be selecting a task and a machine for it (a combination decision).
- In a multi-resource allocation context (cloud with many servers), an action could be a vector assigning tasks to multiple machines at once, or more practically, a sequence of decisions (which job to dispatch to

which server). A simplified action could be "assign task A to server X now".

- In a network resource allocation (e.g. bandwidth allocation) scenario, an action could involve allocating communication channels or power levels to links (as seen in wireless networks). In our context of computing tasks, we focus on computing resource allocation rather than wireless spectrum, so actions revolve around scheduling tasks on CPUs/GPUs.

To keep the model tractable, we often define the action as a single scheduling decision at a time step. For example, the agent may pick one task among all ready tasks and assign it to a certain resource (if resources are identical, just picking the task implicitly chooses the resource by a dispatch rule; if resources differ, the action may explicitly include the resource choice). After the action, the environment (simulator) will advance, e.g., the chosen task will run on the selected resource for some time or until completion, and the state (graph) updates (task completion, resource freeing, etc.). We design the action space to be manageable in size; if there are too many choices (combinatorial explosion), one might use heuristics to limit actions (for instance, actions could be "schedule the highest-priority pending task" but let the agent decide priority via learned value, etc.). In some implementations, a pointer network or attention mechanism can allow the agent to choose from a variable set of entities (tasks) in the state, which naturally fits the GNN output that can produce a score for each task node.

Reward Design: The reward signal guides the RL agent toward good long-term performance. A well-designed reward function is critical. We choose a reward that captures the desired optimization goals of resource allocation. A common approach is to use a weighted combination of key metrics. For example, we can define the immediate reward at time t as:

$$r_t = -\alpha \cdot (\text{Average Job Completion Time}) - \beta \cdot (\text{Makespan}) + \gamma \cdot (\text{Throughput}) + \delta \cdot (\text{Resource Utilization}),$$

Where $\alpha, \beta, \gamma, \delta$ are weights to balance different objectives. This type of reward was effectively used in a GNN-RL scheduler for HPC by Adimora & Sun (2024), where negative makespan and fairness penalties were combined with positive utilization and throughput rewards. In practice, we might not include makespan (if the episode is unending) and instead focus on per-step measures. Another option is to issue a reward for each job completion (e.g. +1 for each task finished, encouraging throughput) and penalize waiting time (e.g. -0.01 for each time unit each task remains unfinished). The reward should also ensure fairness is not degraded; one can include a fairness index term (like Jain's fairness index) in the reward to avoid starvation of some jobs. The cumulative return the RL agent seeks to maximize is $J = E[\sum_{t=0}^T r_t]$, the expected sum of rewards over a scheduling episode. By maximizing J , the agent implicitly tries to minimize job completion times and maximize throughput/utilization in the long run.

RL Algorithm: We employ a deep reinforcement learning algorithm to learn the policy. Several choices are possible: Deep Q-Networks (DQN), Policy Gradient methods (like REINFORCE or Actor-Critic), or Deep Deterministic Policy Gradient (DDPG) if the action space were continuous (not typical here). Given the action space may be large or non-discrete (if choosing both task and machine), a policy gradient

or actor-critic approach can be convenient. In our implementation, we opt for an actor-critic style algorithm: we have a policy network $\pi_\theta(a | s)$ that outputs a probability distribution over actions (or uses a pointer mechanism to select an action), and a value network $V_\phi(s)$ that estimates the value of a state. Both the policy and value networks can incorporate the GNN state embedding h_t as input. In fact, one unified architecture is possible where the GNN produces embeddings, and subsequent fully-connected layers produce both policy logits and value estimate. The agent is trained by simulating many scheduling episodes. We use policy gradient updates (e.g. Advantage Actor-Critic or Proximal Policy Optimization algorithms) to adjust θ, ϕ so that actions leading to higher cumulative rewards are more likely. Over time, the agent learns strategies such as prioritizing short jobs when beneficial, avoiding idle resources, and smartly scheduling dependent tasks to minimize pipeline stalls – effectively discovering algorithms that humans might not easily design. A notable benefit of using GNN in the architecture is that the learned policy can generalize to different numbers of jobs or machines than seen during training. The graph-based state encoding means the same neural network can process a small graph or a larger graph without retraining from scratch, improving the scalability of the solution. This addresses a common challenge in RL for systems: how to train on a small simulated cluster and then apply to a larger one – with GNNs, this transfer is more feasible since the GNN can extrapolate to more nodes (to some extent).

To summarize the approach, Figure 2 illustrates the overall loop of our RL+GNN-based scheduler. The environment (distributed system) is observed as a graph-structured state, which the GNN encodes into an embedding. The RL agent (policy network) uses this embedding to select an action (e.g. schedule a particular task on a machine). The action is applied, the system state changes, and a reward is emitted based on the resulting performance. This loop repeats, and the agent learns a policy that optimizes the cumulative reward over time.

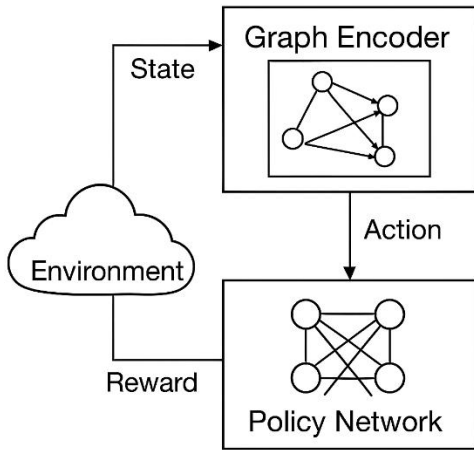


Fig. 2. Reinforcement learning with GNN for resource allocation. The distributed system's state is represented as a graph (tasks and their dependencies, resource connectivity, etc.), which is fed through a Graph Neural Network encoder to produce a latent representation. An RL agent (policy network) uses this representation to decide on an action (e.g., select a task to run on a particular server). The action is executed on the environment, leading to a new state and a reward signal. Over many iterations, the agent learns to make decisions that improve long-term performance.

The training of this framework typically requires a simulator of the distributed system to provide state transitions and rewards. We utilize a custom simulation environment for jobs and resources, which can be built using tools like Python with event-driven simulation. In the training phase, the agent will experience a variety of scenarios (different job arrival patterns, task sizes, etc.) so that it can learn a robust policy. Techniques like experience replay (for off-policy methods) or curriculum training (starting with simple workloads then increasing complexity) can be used to stabilize and accelerate learning. During inference/deployment, the learned policy (the combination of GNN and policy network) can be deployed as a scheduler in a real system, taking in live state observations and outputting decisions in real time. One design consideration is the computational overhead of the GNN and policy evaluation; however, prior works indicate that a moderately complex GNN (with a few thousand parameters) adds only a modest overhead and can run within the scheduling decision time budgets of many systems. In our approach, we ensure the model is lightweight enough for practical use by limiting GNN layers and focusing on essential features.

TABLE I. SUMMARY OF KEY COMPONENTS: FOR CLARITY, TABLE I SUMMARIZES THE MAIN COMPONENTS OF THE PROPOSED RL+GNN SCHEDULER IN THE CONTEXT OF CLUSTER RESOURCE ALLOCATION.

Component	Description
State s_t	Graph-based representation of the system at time t . Nodes can represent tasks (with features like remaining length, resource requirement, dependency status) and possibly resource nodes; edges represent dependencies between tasks or network links between resources. The GNN encodes this graph into a state embedding h_t .
Action a_t	A scheduling decision at time t . For example, select one task (or job) and assign it to a resource for execution. The action space can be the set of all valid task-resource pairs or simply the choice of next task to run if a single resource is considered.
Reward r_t	Feedback reflecting the quality of the allocation decision. Designed to encourage high throughput and resource utilization and low waiting times. E.g., could be +1 for each task completed minus a small penalty per time unit for each pending task, or a weighted sum of metrics as described. This drives the RL agent to optimize overall performance.
Policy & Value Networks	Networks parameterized by θ and ϕ that use h_t to produce the policy $\pi_\theta(a s)$ and the value estimate $V_\phi(s)$ for actor-critic training.
Graph Neural Network	GNN (e.g. Graph Convolutional Network or GraphSAGE) that processes the graph-structured state. It learns to extract features such as which tasks are critical or which resources are overloaded, enabling the policy to make informed decisions. The GNN's parameters are trained jointly with the RL (backpropagating through the GNN based on policy performance).

VI. ILLUSTRATIVE EXAMPLE OF THE OPERATIONAL MATRIX

To illustrate how the proposed RL+GNN framework operates in practice, we consider a simple scheduling scenario with a small number of tasks and resources. Suppose a cluster consists of two compute nodes, R_1 and R_2 , each with limited processing capacity, and a single job composed of three tasks $\{T_1, T_2, T_3\}$. Task T_1 must complete before tasks T_2 and T_3 can begin, forming a simple directed acyclic graph (DAG).

At a given decision step, the system state is represented as a graph where task nodes are annotated with features such as remaining execution time, resource requirements, and execution status (ready or blocked), while resource nodes

encode current availability and load. Edges between task nodes capture dependency constraints, and edges between tasks and resources indicate feasible execution mappings. This graph representation can be expressed in matrix form through node feature matrices and adjacency matrices, which together constitute the operational state provided to the GNN encoder.

The graph neural network processes this structured state through message passing, producing embeddings that summarize both task-level dependency information and resource-level availability. These embeddings are aggregated into a compact state representation and passed to the reinforcement learning policy. Based on this representation, the policy selects a scheduling action, such as assigning task T_1 to resource R_1 , while deferring T_2 and T_3 until their dependency is satisfied. After execution progresses and the state updates, the agent receives a reward reflecting resource utilization and task progress, and the process repeats.

Although simplified, this example illustrates how the proposed framework integrates graph-based state modeling with reinforcement learning to support scheduling decisions. The same formulation naturally extends to larger task graphs and clusters by operating on variable-sized graphs, without requiring fixed-dimensional state representations.

VII. SYSTEM OVERHEAD AND SCALABILITY ANALYSIS

The proposed RL+GNN-based scheduling framework introduces additional computational overhead compared to traditional heuristic schedulers due to the use of learned models for state encoding and decision making. In this work, the system state is processed by a graph neural network (GNN) to generate a compact representation, which is then consumed by the reinforcement learning policy. The computational cost of this process is primarily determined by the number of nodes and edges in the state graph, as well as the depth of the GNN encoder.

From a computational perspective, message-passing operations in the GNN scale linearly with the number of graph edges per layer, while the subsequent pooling and policy inference steps scale with the number of nodes. As a result, the overall inference complexity grows approximately linearly with system size under sparse graph assumptions, which are typical for task dependency graphs and cluster resource representations. In the simulation setting considered in this study, policy inference operates within millisecond-level latency, making it compatible with common scheduling intervals in cluster and cloud systems.

In terms of communication overhead, the framework assumes centralized observation of system state, as is common in many cluster schedulers. Collecting state information such as task status and resource availability introduces communication costs similar to those incurred by existing centralized scheduling solutions. The proposed approach does not require additional synchronization beyond standard scheduler operation. For larger-scale deployments, techniques such as graph sampling, hierarchical scheduling, or partial state aggregation could be employed to mitigate communication and computation overhead.

Scalability considerations are particularly important when extending the framework to clusters with a large number of nodes or highly complex job graphs. While the current evaluation focuses on a conceptual and simulation-based setting, the graph-based formulation naturally supports

variable-sized inputs and does not rely on fixed-dimensional state representations. Nevertheless, further empirical evaluation is required to quantify scalability limits and overhead trade-offs in real systems. Addressing these aspects—such as inference latency under heavy load and memory consumption of GNN models—is an important direction for future work.

VIII. EXPERIMENTAL EVALUATION

To assess the behavior and potential effectiveness of the proposed adaptive resource allocation framework, we conduct a simulation-based evaluation inspired by cluster scheduling scenarios commonly encountered in high-performance computing (HPC) environments. The goal of this evaluation is not to provide definitive real-system performance guarantees, but rather to illustrate how the proposed RL+GNN formulation operates under controlled conditions and how it compares qualitatively with widely used scheduling heuristics.

The simulated environment models a distributed cluster that processes a stream of incoming jobs with heterogeneous sizes and task dependency structures. Each job consists of a directed acyclic graph (DAG) of tasks, capturing precedence constraints typical of parallel workloads. The scheduler observes the evolving system state and makes scheduling decisions over time. We evaluate scheduler behavior using standard metrics commonly adopted in the scheduling literature, including resource utilization (the fraction of total compute capacity actively used), throughput (the rate at which tasks or jobs are completed), makespan (the total completion time for a batch of jobs), and fairness of resource allocation across jobs.

The proposed RL+GNN-based scheduler is compared against several baseline scheduling strategies that are representative of common practice in cluster and HPC systems. These baselines include First-Come-First-Served (FCFS), Shortest Job First (SJF), Round Robin, and EASY Backfilling, a widely deployed heuristic designed to improve utilization by opportunistically scheduling smaller jobs without delaying larger ones. The RL+GNN agent is trained on a diverse set of synthetic workloads and subsequently evaluated on held-out workloads generated under similar conditions to assess generalization within the simulation setting.

Simulation Setup: We simulate a cluster consisting of 1000 compute nodes, each equipped with multiple processing cores, to approximate a large-scale HPC environment. Job arrivals follow a stochastic process inspired by publicly available workload traces, such as those reported for large national laboratory systems, in order to capture realistic variability in job inter-arrival times and sizes. For each job, a task DAG is generated with randomized structure and dependency depth, and individual tasks are assigned execution requirements measured in core-seconds. The evaluation is conducted using a custom, event-driven simulation environment designed to model cluster-level scheduling behavior under controlled conditions. We do not target a specific production system in this study, as the focus is on analyzing the behavior of the proposed formulation rather than benchmarking a particular implementation. The reinforcement learning agent is trained using an actor-critic algorithm with a shared GNN-based state encoder. Policy optimization is performed using a proximal policy

optimization (PPO) variant, which is commonly adopted for stable learning in discrete scheduling problems. Training is conducted over 3,000 episodes, where each episode corresponds to a scheduling horizon with a fixed number of job arrivals. The learning rate is set to 3×10^{-4} for both the policy and value networks, and training is repeated across three random seeds to reduce sensitivity to initialization. Performance metrics reported in the evaluation correspond to representative runs observed after training convergence.

Given the conceptual scope of this study, performance comparisons are presented qualitatively to highlight general scheduling behavior, rather than through statistically aggregated results over multiple runs.

At each decision step, the RL+GNN agent observes the current cluster state, including queued jobs, running tasks, and available resources, and selects scheduling actions according to its learned policy. The reward function is configured to balance multiple objectives—encouraging efficient resource utilization and throughput while discouraging excessive waiting times and unfair allocation. The agent is trained using an actor–critic reinforcement learning algorithm until policy behavior stabilizes. After training, the learned policy is evaluated on unseen workloads using the same simulation environment and compared with the baseline schedulers under identical conditions.

In addition to resource utilization, the simulation-based evaluation suggests that the proposed RL+GNN framework can achieve higher scheduling throughput compared to conventional heuristic-based schedulers under the considered workload conditions. This behavior is primarily attributed to the agent’s ability to overlap task executions and reduce idle periods by exploiting information about task dependencies and current resource availability. In contrast, baseline heuristics such as FCFS or Round Robin follow rigid decision rules that may leave resources underutilized when confronted with heterogeneous jobs or dependency constraints. Even compared to backfilling-based heuristics, which are designed to improve utilization by filling scheduling gaps, the learned policy demonstrates more flexible scheduling behavior in the simulated environment.

The evaluation further indicates that improvements in throughput are accompanied by reductions in overall job completion time, as measured by makespan, for representative workload batches. Rather than prioritizing only short jobs or aggressively favoring specific tasks, the learned policy tends to balance execution across jobs, mitigating extreme waiting times that can arise in heuristics such as SJF under certain workload mixes. Importantly, fairness metrics observed in the simulation remain comparable across all evaluated scheduling policies, suggesting that the potential efficiency gains of the RL+GNN approach do not come at the expense of unfair resource distribution.

Beyond aggregate performance metrics, qualitative inspection of scheduling behavior reveals that the RL+GNN agent learns strategies that resemble—and in some cases generalize beyond—common heuristic practices. For example, the agent opportunistically schedules smaller or ready tasks to reduce idle resources, while also prioritizing critical dependency paths to avoid blocking downstream execution. These behaviors emerge naturally from the reward optimization process rather than being explicitly encoded as

rules, highlighting the adaptability of the learning-based approach.

We also observe that the learned policy exhibits a degree of robustness when evaluated on workload patterns that differ from those seen during training, such as mixtures of short and long jobs. While these observations are limited to the scope of the simulation, they suggest that graph-based state representations may support better generalization across workload variations. Training stability, however, is sensitive to reward design and action-space formulation; careful tuning and curriculum-style workload progression were found to be helpful in practice.

From a practical perspective, policy inference during scheduling decisions operates within millisecond-level latency in the simulation environment, which is consistent with the requirements of many real-world schedulers. The inclusion of a GNN encoder introduces additional computation, but this overhead remains modest relative to the potential cost of inefficient scheduling. These observations align with prior studies reporting that graph-based representations can improve decision quality with manageable computational overhead, making the approach promising for further empirical investigation.

IX. SENSITIVITY AND ROBUSTNESS ANALYSIS

To assess the robustness of the proposed framework, we qualitatively examined the sensitivity of the learned scheduling behavior to variations in workload characteristics and reward weighting. In particular, we observed that moderate changes in the relative importance assigned to throughput, waiting time, and fairness terms in the reward function did not lead to qualitatively different scheduling behavior, suggesting a degree of stability in the learned policy. Similarly, when the workload mix was varied—such as increasing the proportion of long-running jobs or introducing more diverse task dependency structures—the RL+GNN agent continued to exhibit adaptive scheduling patterns within the scope of the simulation. While these observations are limited to qualitative analysis and do not constitute a full statistical sensitivity study, they indicate that the proposed formulation is not overly sensitive to specific parameter choices and can maintain reasonable performance trends across different simulated conditions.

X. BASELINE SCHEDULERS

The proposed RL+GNN-based scheduler is compared against several widely used baseline scheduling policies implemented within the same simulation environment. First-Come-First-Served (FCFS) schedules jobs strictly in their order of arrival without considering task length or dependencies. Shortest Job First (SJF) prioritizes jobs with the smallest estimated total execution time, which can reduce average completion time but may lead to starvation of long jobs under certain workloads. Round Robin (RR) assigns execution opportunities to jobs in a cyclic order with a fixed time quantum, promoting fairness but often resulting in lower resource utilization for heterogeneous workloads. EASY Backfilling is implemented following standard HPC practice, where smaller jobs are opportunistically scheduled ahead of larger ones as long as they do not delay previously reserved resources. All baseline schedulers operate on the same workload traces and system configuration as the RL+GNN agent to ensure a fair comparison.

XI. CONCLUSION

In this paper, we presented an adaptive resource allocation framework for distributed systems that integrates reinforcement learning with graph neural networks. By representing the system state as a graph and employing a GNN-based encoder, the proposed framework enables the learning agent to capture structural relationships such as task dependencies and resource availability, and to generalize across varying workloads and system configurations. The reinforcement learning component allows the scheduler to iteratively improve its decisions based on observed system feedback, resulting in an adaptive policy that can respond to dynamic execution conditions. The methodology detailed the design of graph-based state representations, scheduling actions, and a multi-objective reward function that considers throughput, latency, resource utilization, and fairness.

Using a simulation-based conceptual evaluation in a cluster scheduling setting, we illustrated the potential benefits of the proposed RL+GNN framework compared to conventional static scheduling heuristics. The results suggest that incorporating graph-structured state representations into reinforcement learning can lead to more balanced resource usage and improved scheduling behavior, while maintaining fairness across jobs. Rather than claiming absolute performance superiority, these findings are intended to demonstrate the promise of the approach and to align with broader trends in recent learning-based resource management research.

Several directions remain for future work. A key next step is to conduct comprehensive empirical validation using real-world cloud or HPC workload traces and, where possible, deployment in a live system to assess performance under practical constraints. In addition, system-level considerations such as inference latency, communication overhead, and scalability of the GNN model warrant further investigation; techniques including model compression, hierarchical scheduling, or graph sampling may help address these

challenges. Extending the framework to multi-agent reinforcement learning is another promising avenue, where multiple localized agents with GNN-based perception collaborate in a distributed scheduling architecture. Finally, incorporating additional objectives—such as energy efficiency, reliability, or cost awareness—and improving the interpretability of learned scheduling policies are important directions for increasing the practicality and trustworthiness of learning-based schedulers.

In summary, this work highlights the potential of combining reinforcement learning and graph neural networks as a flexible foundation for adaptive resource allocation in distributed computing environments. While the current study focuses on a conceptual and simulation-based evaluation, the proposed formulation provides a basis for future, fully empirical investigations toward more autonomous and efficient cloud and HPC resource management systems.

REFERENCES

- [1] Ji, M. et al. (2024). Graph Neural Networks and Deep Reinforcement Learning Based Resource Allocation for V2X Communications. arXiv:2407.06518.
- [2] Mao, H. et al. (2019). Learning Scheduling Algorithms for Data Processing Clusters (Decima). Proceedings of SIGCOMM 2019. (Introduces an RL-based scheduler for cluster computing that learns to optimize DAG scheduling, demonstrating better performance than heuristics.)
- [3] Narayanan, D. et al. (2020). Heterogeneity-Aware Cluster Scheduling Policies for Deep Learning Workloads. Proceedings of OSDI 2020. (Illustrates the need for learning-based scheduling in heterogeneous clusters; provides context for multi-resource scheduling challenges.)
- [4] Bhardwaj, R. et al. (2023). Cilantro: Performance-Aware Resource Allocation via Online Reinforcement Learning. Proceedings of OSDI 2023. (An online RL approach for resource allocation with feedback, highlighting adaptability in non-stationary environments, which complements the GNN-RL approach.)
- [5] S. Zhang, J. Chen, X. Wang, and Z. Li, "Learning to Schedule: Reinforcement Learning for Cluster Scheduling," in Proc. USENIX Annual Technical Conference (ATC), 2020.