Multi-Agent Reinforcement Learning for Distributed Workflow Orchestration at the Tactical Edge

A. Amato, A. Morelli, M. Fogli, R. Galliera and N. Suri

MILCOM 2024 - 2024 IEEE Military Communications Conference (MILCOM), 2024

Cite this as

A. Amato, A. Morelli, M. Fogli, R. Galliera and N. Suri, "Multi-Agent Reinforcement Learning for Distributed Workflow Orchestration at the Tactical Edge," *MILCOM 2024 - 2024 IEEE Military Communications Conference (MILCOM)*, Washington, DC, USA, 2024, pp. 64-69, doi: 10.1109/MILCOM61039.2024.10773787.

Multi-Agent Reinforcement Learning for Distributed Workflow Orchestration at the Tactical Edge

Alessandro Amato*†, Alessandro Morelli†, Mattia Fogli§, Raffaele Galliera*†, Niranjan Suri*†‡
* *Department of Intelligent Systems & Robotics - The University of West Florida (UWF), Pensacola, FL, USA*
† *Florida Institute for Human and Machine Cognition (IHMC), Pensacola, FL, USA*
{aamato, amorelli, rgalliera}@ihmc.org
‡ *US Army DEVCOM Army Research Laboratory (ARL), Adelphi, MD, USA*
niranjan.suri.civ@mail.mil
§ *Department of Engineering, University of Ferrara, Italy*
mattia.fogli@unife.it

***Abstract*—The dynamic nature of tactical edge networks has led to the design of architectures that enable real-time data processing and analytics at the edge, to ensure the continuation of operations when the connection to the headquarters is unavailable. However, workflow orchestration faces unique challenges over frequently disconnected, intermittent, and limited (DIL) networks, where traditional approaches, mainly developed for cloud-like environments, lack the flexibility to react promptly to ever-changing conditions. This paper presents a novel decentralized partially observable Markov decision process (DEC-POMDP) formulation for the distributed workflow orchestration problem, where agents need to cooperate to maximize the computation efficiency while reducing the data transmission time. We propose a solution based on multi-agent reinforcement learning (MARL) that leverages graph convolutional reinforcement learning (DGN) and graph attention networks (GAT) to enable agents to share information with each other, capture the network's structural information, ensure scalability, and eliminate the needs for global knowledge of the network. Training and experiments, which compare our solution with the corresponding constraint satisfaction problem (CSP), are conducted in a simulated 2D urban scenario that mimics nodes' mobility and communications, showing promising results.**

***Index Terms*—MARL, Edge Computing, Service Orchestration, Tactical Edge, Workflow, Workflow Deployment**

I. Introduction

In recent years, tactical edge distributed applications have adopted new design and deployment patterns to enable real-time data processing and analytics even when they are disconnected from the rest of the network. This shift is a natural response to the unique challenges that applications face in disconnected, intermittent, and limited (DIL) networks, such as tactical edge networks (TENs).

Service, data pipelines, and workflow orchestration solutions designed for cloud-like scenarios typically assume that they have complete vision over the whole network and that such network is mostly static. These assumptions are broken in TENs, where continuous orchestration is required to adapt to ever-changing conditions, shifting of priorities, and potential interference from adversarial forces.

Optimal workflow orchestration solutions are available, but they are NP-hard [1] and therefore demand excessive computational resources. Heuristic solutions provide sub-optimal results within reasonable times, but often require fine-tuning for specific environments and lack the necessary flexibility to promptly react to unexpected changes in the scenario. AI-driven approaches have been proposed, as alternatives or in combination with traditional methods, but they also often require global knowledge of the network, are centralized, and scale poorly.

This paper introduces a novel decentralized partially observable Markov decision process (DEC-POMDP) formulation for the distributed workflow orchestration problem. In this framework, agents coordinate to redeploy services across a TEN, aiming to maximize computational efficiency and minimize data transfer time. We present a preliminary study that uses multi-agent reinforcement learning (MARL) to solve the distributed workflow orchestration problem. Our MARL solution eliminates the need for global knowledge of the network by leveraging graph convolutional reinforcement learning (DGN) and graph attention networks (GAT), enabling agents to proactively coordinate by sharing information and effectively capture the network's structural information. Agents communicate solely with their neighbors, ensuring scalability and low communication overhead. To train and evaluate our agents, we created a 2D environment implementing the dynamics of our proposed DEC-POMDP, in which we accurately model nodes' mobility patterns and network communications characteristics, including packet loss, latency, and throughput.

We compared our agents against the optimal solution, obtained solving the corresponding constraint satisfaction problem (CSP), and against two ad-hoc agents that implement different heuristics. The results show that our agents exhibit promising performance and good generalization capabilities to unseen scenarios.

II. Methodology

This section introduces the service deployment environment (SDE) and the DEC-POMDP formulation for the workflow orchestration at the tactical edge problem.

We define a workflow as a graph of services or jobs working together to achieve a larger task. In this graph, each node represents a service with specific host requirements,

such as CPU cores, RAM, and storage size. The edges indicate the minimum communication requirements between these services, including bandwidth, latency, and packet loss. The process of managing a workflow's life cycle is known as workflow orchestration.

As a starting point for our research, we limited the workflow structure to a tripartite graph, Fig. 1. This structure serves as an initial step towards building multi-layer dependencies among jobs in a workflow. In the environment, we associate the first layer with data sources to be processed. The second with services performing the computation, and the last one as end-user services that receives and aggregates the previous layer's output. The objective of our agents is to cooperate in maximizing the computational efficiency of the workflow and minimizing the data transfer time from the first to the third layer by strategically moving the services in the second layer. In our setup, the aggregation happens asynchronously, without the need for all the services to be connected to the receiver at the same instant, and we assume no minimum requirement for nodes and communication links.

A. Service Deployment Environment

The SDE serves as a training and evaluation environment for our agents, modeling the computation and movement behavior of nodes, as well as the behavior of the network links between them. Specifically, the environment simulates a 2D urban scenario with N nodes (agents) participating in a TEN with heterogeneous computational resources, interacting with each other to orchestrate a workflow.

These nodes, which can be seen as C2 vehicles or UAVs performing individual missions, follow a random waypoint mobility model. At the beginning of the simulation—referred to as an episode—they are initially positioned near a base station as a starting point. From there, they move at a constant speed to a randomly selected destination. Upon reaching their destination, a new one is assigned, and this process continues until the episode ends.

The underlying network can be represented as a graph $G = (V, E(t))$, where V represents the nodes in our network and $E(t)$ represents the set of edges at time step t. An edge $(u, v) \in E$ exists if $P(u, v) \leq P_{\text{cutoff}}$, where $P(u, v)$ denotes the path loss between nodes and P_{cutoff} the cutoff value. Edges are characterized by a set of features: packet loss, bandwidth, and latency. In our settings, packet loss increases linearly with path loss, link throughput is calculated using the Mathis model [2], and latency is stable across the entire graph.

During the episode, a set of data sources $S = \{s_1, s_2, \ldots, s_n\}$, end-user services $D = \{d_1, d_2, \ldots, d_m\}$, and a workflow $J = \{j_1, j_2, \ldots, j_k\}$ are generated within the environment. Each job j_i is a tuple characterized by a data production coefficient θ within 0 and 1, a destination $d \in D$, a source $s \in S$, and a number of instructions ν. In the simulation, jobs receive an amount of data p from s and produce an output equal to $\theta * p$. In order for j_i to execute, a path between d and source node s must exist at a certain time step t.

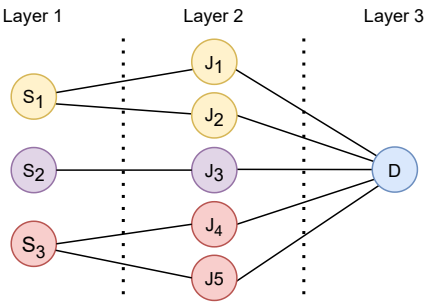


Fig. 1: **Example of a tripartite workflow:** The first layer consists of a group of data sources. The second layer contains the services. The third layer includes an end-user service that aggregates the output from the second layer.

The computational time for a job is determined by the resources of the hosting node. When multiple jobs are hosted on the same node, they are treated as a single job, with the resulting computational time equal to the sum of their instructions divided by the computational speed of the node.

In the environment, decisions about workflow jobs are made sequentially, with one job being examined at a time by the agent where the job resides. Initially, all the jobs are assigned to the agent closest to d.

B. The MARL Approach

A Markov decision process (MDP) is a model of an agent interacting with an environment. The agent receives the state of the environment as input and produces an action as output, which, with some level of uncertainty, affects the environment's state [3]. Upon acting, the agent receives a reward, which in an MDP depends only on the state and action taken at time $t - 1$.

This concept has been further expanded into a partially observable Markov decision process (POMDP), where the agent lacks complete knowledge of the environment and instead acts based on local observations.

In a multi-agent setting, POMDP can be extended to DEC-POMDP, where multiple agents make decisions under uncertainty—such as missing communication links—sharing the same reward function [4]. A DEC-POMDP is formally characterized by the following tuple:

$$\langle \mathcal{I}, \mathcal{S}, {\mathcal{A}^i}_{i \in \mathcal{I}}, \mathcal{P}, \mathcal{R}, {\mathcal{O}^i}_{i \in \mathcal{I}}, \gamma \rangle \tag{1}$$

where $\mathcal{I}$ defines the set of agents, $\mathcal{S}$ is the global state of the environment, ${\mathcal{A}^i}_{i \in \mathcal{I}}$ represents the action space for each agent, $\mathcal{P}$ is the state transition probability, $\mathcal{R}$ is the reward function, ${\mathcal{O}^i}_{i \in \mathcal{I}}$ is the set of local observations for each agent, and γ is a discount factor.

MARL is a sub-field of reinforcement learning (RL) that exploits these concepts and studies behaviors and interactions of multiple autonomous agents to learn a decision-making policy π which guides agents' actions to maximize the reward function R.

The SDE, expands the definition of DEC-POMDP with $\mathcal{J}$ as the workflow space, ${\mathcal{M}^i}_{i\in\mathcal{I}}$ as the movement function for each agent, and $\mathcal{E}$ as the agents' connectivity model.

Therefore, the final DEC-POMDP can be described as:

$$\langle \mathcal{I}, \mathcal{S}, {\mathcal{A}^i}_{i\in\mathcal{I}}, \mathcal{P}, \mathcal{R}, {\mathcal{O}^i}_{i\in\mathcal{I}}, \mathcal{J}, {\mathcal{M}^i}_{i\in\mathcal{I}}, \mathcal{E}, \gamma \rangle \tag{2}$$

C. Observation and Action space

In the SDE each observation, Table I, includes the agent's unique ID, computational power, number of jobs currently deployed on it, the unique ID of the receiver and data source for the current job, and the predicted data transfer time from the data source to the receiver. Additionally, the agent perceives the ID, transfer time for its two best deployment options, and a flag indicating whether the current job would perform faster on one of these options. Finally, we include 5 past observations of the packet loss difference between the data source-receiver and data source-agent-receiver path for all the data sources available. This method, known as observation staking, allows the agents to perceive historical patterns in a certain time window, understand their current direction, and better react to crucial events, such as imminent link disconnections. As a result, the agents can adjust their actions by either anticipating or delaying deployments.

After receiving an observation, the agent is called upon to act. Allowing the agent to select the ID of the agent to which a job should be moved is not feasible in a large-scale scenario, where the action space would become prohibitively large. Therefore, we adopted a hybrid approach, formulating a set of heuristics for the environment and allowing the agent to select which one to apply when moving a job.

The advantages of this approach are its high scalability, as the action space is no longer directly tied to the number of agents, and the possibility for the users to incorporate domain-specific knowledge by formulating the heuristics that the agents will use.

In our context, we adopted two heuristics: greedy performance first (GPF) and greedy bandwidth first (GBF). The first one, GPF, selects the node that maximizes the computational time for the current job, while the second one, GBF, selects the node that will minimize the data transmission time.

Finally, we allow the agent to decide whether to maintain its current state, leading to a final one-dimensional discrete action space of 3 elements.

D. Reward Function

The reward function allows agents to evaluate the effectiveness of their actions. In this work, the objective of the agents is to minimize data transmission time while maximizing computational performance and we propose a reward function based on two main components.

The first balances the difference between the number of instructions executed in a single step across the entire workflow and the data transmission time. We multiply the latter by a constant α to control the impact of this term, preferring lower values in scenarios where the transmission time dominates the reward function, or higher values in scenarios where the transmission time should be prioritized. More formally:

$$R_p(t) = |I_{performed}(t)| - \alpha\,|T_{time}(t)| \tag{3}$$

$I_{performed}(t)$ and $T_{time}(t)$ are normalized using min-max normalization at the sub-graph and global levels, respectively. The sub-graph normalization encourages the agent to utilize the most performant node within its sub-graph without penalizing it in disadvantaged scenarios, such as when the most performant node is in a sub-graph that the agent cannot reach. It is also important to note that while decisions are made at the job level, rewards are given at the workflow level, providing the agents with a global perspective and discouraging selfish job-level behaviors.

The final element of our reward aims to steer away the agents from performing unnecessary deployments. We implement such mechanism by issuing small penalties every time a deployment is performed by an agent. Moreover, this penalty is increased when the deployment happens with no connection to the target or receiver.

The overall reward function can be formally defined as:

$$R(t) = R_p(t) - P_{deployment} \tag{4}$$

E. Graph Convolutional Reinforcement Learning

Given the dynamic nature of the environment and its graph representation, we decided to use DGN[5] in conjunction with GAT[6] as learning architecture for our agents.

In DGN, each agent's local observation is processed by a first encoding layer, a multi-layer perceptron (MLP), which transforms them into a first latent representation. Neighborhood encodings are then aggregated through a series of convolutional layers. The number of convolutional layers determines the aggregation levels and thus the number of hops over which our agent will aggregate its encoding. For instance, one convolutional layer enables aggregation with first-hop neighbors; adding another convolutional layer, as in our setup, allows aggregation with both first- and second-hop neighbors. However, it is important to note that regardless of the number of convolutional layers, our agents will communicate only with their neighbors, making this approach practical in real-world applications.

While the original paper on DGN proposes the use of multi-head dot-product attention as convolutional kernel to aggregate

Agent ID	**Comp. power**	**Num. jobs deployed**	**Recv. ID**	**Send. ID**	**Curr. Transf. time**	**GPF ID**	**GPF Transf. time**	**GPF Impr.**	**GBF ID**	**GBF Transf. time**	**GBF Impr.**	**Si ID t-5**	**Si ΔL t-5**	...	**Si ID t**	**Si ΔL t**
0	1500	3	51	50	780	5	985	1	8	630	0	51	0.5	...	51	0.1

TABLE I: Example of nodes' observation

information between agents, we instead use a multi-head graph attention network (GAT). GAT dynamically learns weights between nodes and neighbors, also aggregating edge features such as latency, bandwidth, and packet loss in the decision process. This approach has already archived important results in similar networking tasks, such as information dissemination [7] and dynamic network bridging [8].

The output of the preceding layers is concatenated and fed into a dueling Q network, which performs advantage and value estimations, ultimately producing the Q-values that determine the agent's actions.

III. Experimental Setup

In this section, we present how we set up the environment for training and evaluation.

During each episode, a workflow of 10 jobs is generated within the largest group of nodes. Each job has a number of instructions randomly selected between 10^5 and 10^6 to prevent trivial solutions where all jobs are deployed on a single node. For each job, the data production coefficient $\theta = 1$, and the amount of received data $p = \nu$.

To execute, jobs need to be connected to both a data source and a destination. In our setting, three data sources and one destination are randomly positioned within the workflow's node group. We set the same destination for all the jobs, while we randomly assign the data sources.

Nodes in the simulation are categorized into one of three possible instances representing different computing capabilities: small (2500 instructions/step), medium (5000 instructions/step), and high (7500 instructions/step).

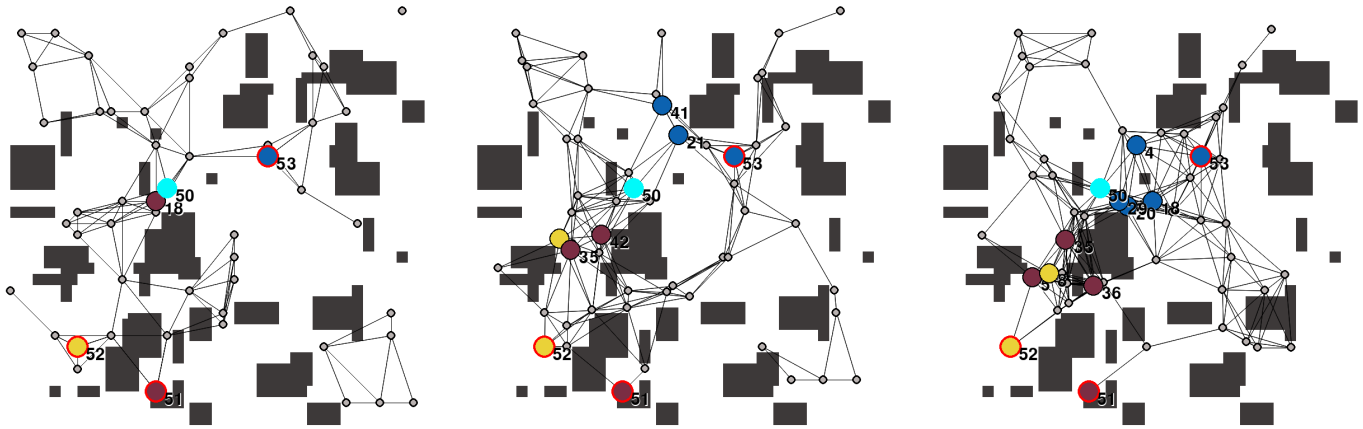
The environment supports two different path loss propagation models: Free Space [9] and Okumura-Hata [10]. While we experimented with both models, the Okumura-Hata model required extensive fine-tuning, without which the agents struggled to understand link behavior. Consequently, we used the Free Space model for both training and evaluation. To avoid over-connected scenarios, we selected a path loss cutoff value of 105 dBm based on an empirical analysis. Packet loss for links is set to range from 0 to 0.1, increasing linearly with the path loss value. The link's throughput follows the Mathis model [2], with an initial bandwidth of 1500 Kbit/s for all the links in the simulation. Latency is set to 10ms for each link.

MARL was trained over 300,000 episodes, each limited to 200 steps.

IV. Results and Discussion

In this section, we discuss the results obtained from evaluating our policy across 80 randomly generated scenarios, each 200 steps long.

We compare our agents with an optimal CSP that maximizes the difference between the number of instructions executed and the data transfer time during each step. The CSP operates in a centralized manner, with a global knowledge of the network and control over the entire workflow, deploying all jobs simultaneously. The maximum computational time for the CSP to compute its solution is set to 3 seconds per step, ensuring that the solution is optimal within this time frame. Additionally, Table II includes the results achieved by our agents when using the GPF and GBF heuristics instead of the learned policy, demonstrating their ability to switch between the two. The discussion is based on six metrics: service availability, number of deployments, average per-job completion percentage, average per-job transmission time, latency, and the time required to compute the solution. To demonstrate the generalization capability of our agents, although training was conducted with 50 agents and 10 services per workflow,



(a) (b) (c)

Fig. 2: **Workflow distribution during a MARL episode.** Data sources are outlined in red and color-coded: Yellow (52 - bottom left), Bordeaux (51 - bottom center), and Blue (53 - top right). Each service is assigned to a data source and a receiver. The receiver is the same for all services (50 - center - light blue). The service color determines the data source to which it must be connected. **Fig. 2a** shows all services initially deployed on the closest node (18). **Fig. 2b** illustrates the movement of services between the sensors and the receiver. **Fig. 2c** shows final positions optimized for computational efficiency.

the evaluation also includes scenarios with 50 agents and 20 services, 100 agents with 10 services, and 100 agents with 20 services. All results were computed on a machine with an Intel Core i9-13900F, 32 GB RAM, and an RTX 4090 GPU.

Starting with service availability, which measures the percentage number of steps in which our services were both connected to the data source and the receiver, our agents make near-optimal decisions, with a worst result in the 50/20 scenario being 5.35% less available than the optimal.

The number of deployments shows how our agents significantly outperform the optimal solution, reducing deployments by an average of 70.29%. This result is expected given the deployment penalty in the reward function and highlights the agent's ability to anticipate or delay deployment, proactively responding to the scenario.

The average job completion provides information about the computational efficiency of the solution. Our agents are on average 42.45% less efficient than the optimal CSP solution. The gap observed between the GPF and GBF heuristics illustrates the challenge of balancing computational efficiency with network constraints.

The average data transmission time metric reflects the time required for data to flow from the source to our agents and then to the receiver. Here, our agents trade off some computational efficiency to outperform the optimal solution, achieving an average improvement of 4.11%. Correspondingly, in the latency metric, MARL outperforms the CSP, being 53.38% faster.

Finally, MARL takes an average of 0.05 seconds to perform a step in the environment, compared to 3.28 seconds for the CSP. However, it is important to note that while MARL may require multiple iterations to reach a final solution, the CSP guarantees the optimum in a single step.

Fig. 2 shows a full workflow orchestration with MARL throughout an episode.

V. Related Work

Extensive research has been conducted on the topic of orchestration in data centers and clusters. However, the edge remains a challenging scenario due to its heterogeneous and dynamic nature.

Similarities with the edge can be seen in fog computing. In de Brito et al. [11], the authors proposed an architecture for fog-enabled infrastructures based on two components: the fog orchestrator (FO) and the fog orchestration agent (FOA). In their architecture, each fog node has an instance of the FOA, which performs operations such as monitoring and resource management, controlling the life status of the local node and the services within it. A FO sees a group of FOAs as a centralized infrastructure and manages the lifecycle of an orchestration across multiple fog nodes. A FOA can seamlessly switch roles and act as a FO, thus controlling a group of FOAs itself.

While the FOA can resemble our MARL's agents, the main difference lies in allowing all the nodes participating in the architecture to act as FOs in a distributed decision-making process. However, their paper primarily focuses on

	Num. of agents	Num. of jobs	CSP	GPF	MARL	GBF
↑ Availability (%)	50	10	87.48 ± 18.71	*85.2 ± 18.13*	87.42 ± 17.24	**88.05 ± 16.59**
		20	**89.45 ± 20.28**	*82.2 ± 17.98*	84.1 ± 17.95	86.35 ± 18.39
	100	10	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**
		20	**96.82 ± 6.75**	*95.48 ± 10.58*	96.78 ± 6.97	96.78 ± 6.97
↓ Number of deployment	50	10	*193.8 ± 14.96*	**21.7 ± 4.57**	49.85 ± 12.82	66.05 ± 13.39
		20	*198.95 ± 4.7*	**53.95 ± 6.65**	71.2 ± 12.26	100.65 ± 18.46
	100	10	*199.8 ± 0.52*	**17.95 ± 4.17**	42.95 ± 12.45	56.0 ± 18.11
		20	*200.0 ± 0.0*	**44.85 ± 6.17**	71.65 ± 13.96	94.85 ± 23.22
↑ Avg. job completion (%)	50	10	**393.4 ± 102.83**	303.22 ± 68.48	209.76 ± 61.26	*86.74 ± 25.14*
		20	**313.39 ± 54.02**	230.75 ± 46.57	150.47 ± 31.03	*49.5 ± 13.53*
	100	10	**367.97 ± 59.44**	346.71 ± 57.22	245.64 ± 81.79	*72.2 ± 23.36*
		20	**378.56 ± 81.34**	338.43 ± 73.67	235.11 ± 61.66	*55.6 ± 20.33*
↓ Avg. transmission time (s)	50	10	1060.5 ± 232.31	*1078.49 ± 143.03*	1037.81 ± 170.55	**1017.37 ± 176.68**
		20	*1153.17 ± 166.53*	1126.08 ± 114.72	1101.27 ± 111.56	**1085.8 ± 111.32**
	100	10	*1162.9 ± 135.19*	1149.6 ± 131.73	1113.39 ± 125.11	**1097.09 ± 121.99**
		20	*1166.73 ± 121.03*	1140.91 ± 117.97	1109.99 ± 113.36	**1095.49 ± 113.95**
↓ Latency (ms)	50	10	*53.99 ± 9.24*	47.36 ± 7.12	36.98 ± 6.75	**30.34 ± 5.6**
		20	*62.41 ± 10.42*	48.48 ± 8.76	37.09 ± 6.97	**30.23 ± 6.44**
	100	10	*57.8 ± 7.31*	49.65 ± 8.16	40.19 ± 8.83	**34.51 ± 7.23**
		20	*69.12 ± 7.11*	55.02 ± 6.6	44.46 ± 5.9	**38.28 ± 5.79**
↓ Time to compute a step (s)	50	10	*3.08 ± 0.23*	**0.02 ± 0.0**	**0.02 ± 0.0**	**0.02 ± 0.0**
		20	*3.2 ± 0.08*	**0.02 ± 0.0**	**0.02 ± 0.0**	**0.02 ± 0.0**
	100	10	*3.34 ± 0.15*	**0.09 ± 0.0**	0.09 ± 0.01	**0.09 ± 0.0**
		20	*3.5 ± 0.25*	**0.09 ± 0.0**	0.1 ± 0.01	**0.09 ± 0.0**

TABLE II: Comparison of CSP, GPF, MARL, and GBF approaches. **Bold** values represent the best results, while *italic* values indicate the worst. The symbols ↑ and ↓ denote whether higher or lower values are preferable, respectively

the architectural details, while our primary concern is strictly related to service orchestration.

In Bastiaansen et al. [12] and [13], the concept of federation and trust is introduced, with multiple cloud partners sharing or utilizing heterogeneous edge computational resources to enable deployment of a workflow at the tactical edge. In the presented architecture, each partner deploys within its cloud four different components: federated resource discovery (FRD) which provides information about the current resources available in the federation, federated service deployment (FSD) which performs service deployment, trust resource management (TRM) collecting security intra-cloud metrics, and the federated adaptive orchestrator (FAO) in charge of computing workflow deployment plans across the federation. Currently, the FAO relies on a CSP that optimizes over trust values provided by the TRM component. While this approach provides an optimal resource allocation, the FAO requires a complete view of the network to compute its solution, and based on the number of clouds and constraints within the services, may require time.

Here, our approach would provide an alternative to the FAO, removing the need for a comprehensive view of the network and enhancing the overall solution scalability. A comparison has been proposed in Section IV.

In recent years, multiple machine learning (ML) approaches have been proposed for network and service monitoring. An interesting method is presented by Zeydan et al. [14], where the authors introduce a data-driven, ML-based node selection strategy for mobile operators. Their ML platform collects metrics from Kubernetes and OpenDaylight to rank and select the most promising nodes, enhancing data connection and restoration capabilities in the event of failures.

We see their approach complementary to ours, providing agents with metrics to evaluate where to deploy a service.

While current literature lacks examples of MARL approaches in the context of distributed workflow orchestration, some solutions have been proposed by abstracting the problem to a job-shop scheduling scenario. Wang et al. [15] approach it by assigning each job to a specific agent, allowing jobs to select where to be scheduled. In their context, each agent can perform $S + 3$ actions, where S is the number of machines involved in the process.

This can't apply to our use case, where the number of services can be greater than the number of nodes, and the policy convergence would be highly compromised by the action space proportional to the number of nodes. A solution to this has been proposed in Section II.

VI. Conclusions and Future research direction

This paper proposed a DEC-POMDP formulation for the distributed workflow orchestration problem and conducted a preliminary study using MARL to solve it. We compared our multi-agent solution with the corresponding CSP and demonstrated the ability of MARL to learn decentralized policies and achieve sub-optimal results. Future research will focus on improving the environment by allowing for a more complex workflow formulation, enhancing job definition, and modeling how bandwidth is allocated when multiple jobs communicate over the same link. Moreover, we plan to integrate our solution in an emulated network environment, leveraging a custom in-house network emulator [16] and the federated architecture proposed in [13].

Acknowledgment

The authors would like to acknowledge the NATO RTG IST-193 "Edge Computing at the Tactical Edge" for inspiring this paper.

References

[1] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni *et al.*, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *Journal of Parallel and Distributed Computing*, vol. 61, no. 6, pp. 810–837, 2001.

[2] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, "The macroscopic behavior of the tcp congestion avoidance algorithm," *SIGCOMM Comput. Commun. Rev.*, vol. 27, no. 3, p. 67–82, jul 1997.

[3] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, no. 1, pp. 99–134, 1998.

[4] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein, "The complexity of decentralized control of markov decision processes," *Mathematics of Operations Research*, vol. 27, no. 4, pp. 819–840, 2002.

[5] J. Jiang, C. Dun, T. Huang, and Z. Lu, "Graph convolutional reinforcement learning," 2020. [Online]. Available: https://arxiv.org/abs/1810.09202

[6] S. Brody, U. Alon, and E. Yahav, "How attentive are graph attention networks?" 2022. [Online]. Available: https://arxiv.org/abs/2105.14491

[7] R. Galliera, K. B. Venable, M. Bassani, and N. Suri, "Collaborative information dissemination with graph-based multi-agent reinforcement learning," 2024. [Online]. Available: https://arxiv.org/abs/2308.16198

[8] R. Galliera, T. Möhlenhof, A. Amato, D. Duran *et al.*, "Distributed autonomous swarm formation for dynamic network bridging," 2024. [Online]. Available: https://arxiv.org/abs/2404.01557

[9] H. Friis, "A note on a simple transmission formula," *Proceedings of the IRE*, vol. 34, no. 5, pp. 254–256, 1946.

[10] M. Hata, "Empirical formula for propagation loss in land mobile radio services," *IEEE Transactions on Vehicular Technology*, vol. 29, no. 3, pp. 317–325, 1980.

[11] M. S. de Brito, S. Hoque, T. Magedanz, R. Steinke, A. Willner *et al.*, "A service orchestration architecture for fog-enabled infrastructures," in *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, 2017, pp. 127–132.

[12] H. Bastiaansen, J. v. d. Geest, C. v. d. Broek, T. Kudla, A. Isenor *et al.*, "Federated control of distributed multi-partner cloud resources for adaptive c2 in disadvantaged networks," *IEEE Communications Magazine*, vol. 58, no. 8, pp. 21–27, 2020.

[13] A. Amato, H. Bastiaansen, W. Datema, M. Fogli, R. Fronteddu *et al.*, "A performance cost/benefit analysis of adaptive computing in the tactical edge," in *2024 International Conference on Military Communication and Information Systems (ICMCIS)*, 2024, pp. 1–8.

[14] E. Zeydan, J. Mangues-Bafalluy, and Y. Turk, "Intelligent service orchestration in edge cloud networks," *IEEE Network*, vol. 35, no. 6, pp. 126–132, 2021.

[15] X. Wang, L. Zhang, T. Lin, C. Zhao, K. Wang, and Z. Chen, "Solving job scheduling problems in a resource preemption environment with multi-agent reinforcement learning," *Robotics and Computer-Integrated Manufacturing*, vol. 77, p. 102324, 2022.

[16] A. Amato, R. Fronteddu, and N. Suri, "Dynamically creating tactical network emulation scenarios using unity and emane," in *MILCOM 2023 - 2023 IEEE Military Communications Conference (MILCOM)*, 2023, pp. 201–206.