AI-Assisted Synergetic Orchestration Mechanisms for Autoscaling in Computing Continuum Systems

Anastasios Zafeiropoulos, Nikos Filinis, Eleni Fotopoulou, and Symeon Papavassiliou

Abstract

The dominance of microservices-based development frameworks and the development of complex distributed applications, along with the massive production of powerful Internet of things (IoT) and edge computing devices, are paving the way toward the transition from centralized to distributed computing paradigms, where applications are deployed across the computing continuum. In the computing continuum, we consider the management of resources from the IoT to the edge to the cloud part of the infrastructure. To achieve effective management of resources and application workloads, a set of challenges have been identified. These challenges revolve around the need to introduce automation and distributed intelligence characteristics into emerging orchestration mechanisms. To treat this, the design of synergetic or cooperative orchestration mechanisms that take advantage of artificial intelligence techniques is arising as a promising approach. In this article, we describe the design and development of a synergetic orchestration mechanism, powered by multi-agent reinforcement learning (MARL), to guide the autoscaling of distributed applications that are deployed across the computing continuum. A MARL environment and a corresponding agent have been developed and applied in real and simulated environments to guide scaling actions, while indicative evaluation results are provided.

Introduction

Cloud and edge computing technologies have emerged at a high pace in the last years as the mainstream computing paradigms, covering a wide set of application domains with diverse Quality of Service (QoS) requirements. A plethora of orchestration solutions has been made available in traditional cloud and edge computing systems [1, 2]. These are mostly targeted at the specificities of the available infrastructure (e.g., cloud-native computing systems, Internet of Thins (IoT) devices management) and the application needs (e.g., very low latency). Following these evolutions, the development of distributed applications has emerged as a recent paradigm, where the application is decomposed in various parts or microservices that are deployed in different components of the computing infrastructure. This paradigm is accompanied with notable advantages such as scalability, fault tolerance, security and increased efficiency due to parallel processing.

To serve distributed applications deployed across various parts of the computing infrastructure, the existing centralized computing model is under extension toward distributed computing continuum systems [3]. With the term computing continuum, we refer to the combination of computing and communication resources and services, offered across an end-to-end infrastructure [4, 5]. The computing continuum extends the cloud computing infrastructure with edge and IoT computing devices. The objective is to support the deployment and management of distributed applications over distributed resources and serve energy-efficiency, high performance, security and privacy requirements [5]. For instance, in this way, distributed data processing and Machine Learning (ML) workflows at the edge can be better managed, considering the rapid shift foreseen for the execution of such workflows in containerized environments [4, 6].

Systems operating over the computing continuum can be considered complex systems since applications take advantage of multiple computing tiers and orchestration stacks [3]. Each orchestration stack is currently considered a silo, functioning in an egocentric way to manage deployments in a local environment, without promoting collaboration with other stacks [7]. Such an approach in orchestration has to be radically changed to achieve interoperability, distributed intelligence and seamless management of distributed applications, while in parallel accomplish the Service Level Objectives (SLOs) defined at the application level. "Ego" should be treated as an enemy and get substituted by the term synergy or cooperation among orchestration components [3].

In the work presented in this manuscript, the main contribution regards the design and development of a synergetic orchestration mechanism to support the autoscaling of distributed applications that are deployed across the computing continuum. With the term autoscaling we refer to the automated adjustment of the assigned resources to manage a workload by the various application parts, based on the dynamic configuration of the number of concurrent replicas that can support the workload [8]. Such a challenge is already identified in the literature [4], since it is not trivial to achieve efficiency in the autoscaling

Digital Object Identifier: 10.1109/MCOM.001.2200583

The authors are with National Technical University of Athens, Greece.

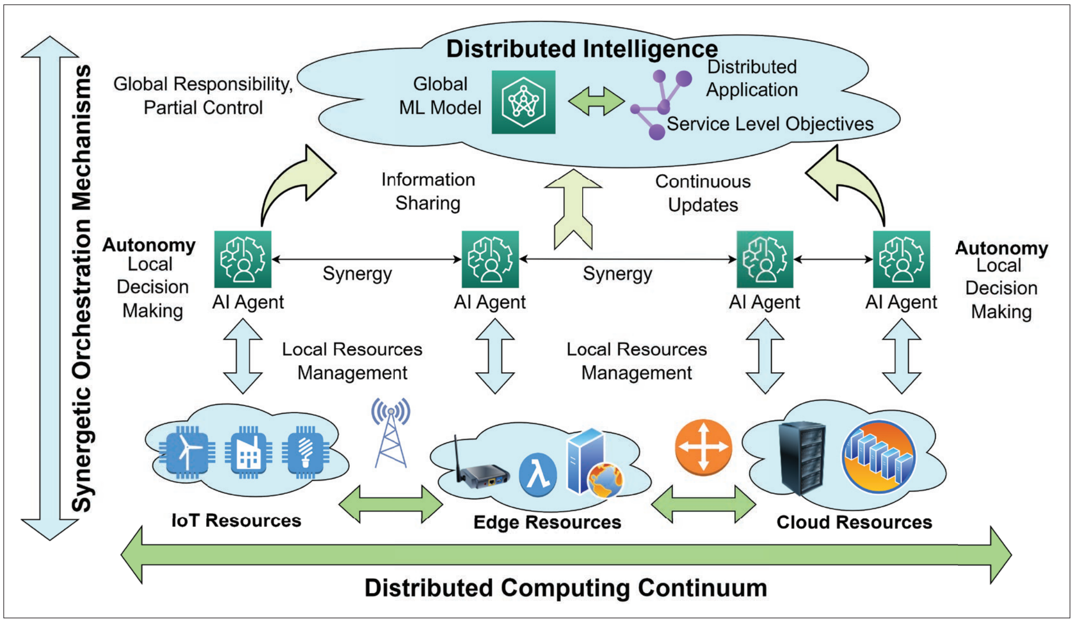

FIGURE 1. Synergetic Orchestration in the Distributed Computing Continuum.

of resources in an automated manner and under diverse workload fluctuations.

The proposed mechanism takes advantage of cooperative AI techniques and specifically Multi Agent Reinforcement Learning (MARL) [9]. A set of MARL agents collaborate to take local decisions per component of a distributed application, having as an objective the assurance of a Service Level Agreement (SLA) at the application level. A "system-of-systems" approach is considered [10] where decision making takes place at a local level (application component), while knowledge is shared at a global level (distributed application) (Fig. 1). In this setting, the overall coordination entity has the responsibility, but not the control of the reserved resources nor the knowledge and authorization for proper horizontal scaling of the various application parts. Various sub-coordination entities — the AI agents in our case — have to collaborate, exchange knowledge and take decisions in view of local and global optimization objectives.

The structure of the manuscript is as follows. In the next section we provide a short overview of emerging trends in the development of distributed applications for the computing continuum. Attention is given to the combination of multi-agent systems and reinforcement learning techniques. Following that, the proposed autoscaling mechanism is presented, and then we provide details for the development of the MARL environment and the testbed infrastructure that we have used for evaluation purposes. Evaluation results are provided following that, while the final section concludes the manuscript and highlights open research areas.

Synergetic Orchestration in the Distributed Computing Continuum

Management of Distributed Application Graphs

The development of modern cloud and edge computing applications follows a microservices-based paradigm, where distributed applications are represented in the form of an application graph. Each application graph has as nodes a set of application components (or microservices) that are independently manageable. For instance, in the case of autoscaling mechanisms, each application component can scale independently to the number of replicas considered suitable to serve the incoming workload at each point of time. Each application component implements part of the business logic of the distributed application and may have specific deployment requirements (e.g., latency sensitivity, computational intensiveness). The application components are interlinked based on the dependencies that exist among them for the consumption or exposure of software functions. In this way, an application graph with nodes and directional links among them is composed that can be deployed over hybrid infrastructure across the continuum based on various objectives or constraints.

Consideration of the application graph as a complex system during resource provisioning is required, compared to existing approaches where each microservice is managed individually. The dependencies among the microservices have to be included in the analysis part [11], introducing a need for intelligent orchestration mechanisms able to consider the trade-off between the accomplishment of local (microservice-based) and global (application-based) objectives.

Synergetic Orchestration Mechanisms

A few studies exist related to the specification of orchestration mechanisms for the computing continuum [11, 12]. Based on them, the synergetic orchestration mechanisms can be applied to guide various orchestration actions. These include the lifecycle management of applications, resources management (compute offloading, scaling, live migration), and network management (e.g., traffic engineering) [13]. In all cases, the application graph, the dependencies among its components and the defined SLOs should be considered as input in the analysis part.

One of the main objectives of the enforcement of synergetic orchestration mechanisms is

Formulation of a multi-agent system to support synergetic orchestration actions is beneficial, given that each individual AI agent has limited information or resources to achieve a global objective of the application. The AI agents can collaborate to achieve one or more joint objectives (e.g., by dynamically adjusting the allocated resources per part of the continuum), while considering the existence of good performance metrics at a local level (e.g., by monitoring the throughput per microservice).

to increase the decentralized intelligence and the autonomicity in terms of distributed resources management [12]. To achieve this, the consideration of techniques applied in the management of multi-agent systems and the joint exploitation of AI techniques has to be promoted [6]. In the case of a multi-agent system operating in the computing continuum, each AI agent should be responsible for managing part of it and collaborate in a distributed way with the rest agents to offer the desired functionality. Formulation of a multi-agent system to support synergetic orchestration actions is beneficial, given that each individual AI agent has limited information or resources to achieve a global objective of the application. The AI agents can collaborate to achieve one or more joint objectives (e.g., by dynamically adjusting the allocated resources per part of the continuum), while considering the existence of good performance metrics at a local level (e.g., by monitoring the throughput per microservice).

Multi-Agent Reinforcement Learning for Synergetic Orchestration

A promising approach to develop synergetic orchestration mechanisms that take advantage of multi-agent systems and AI techniques is based on the evolution of the Reinforcement Learning (RL) algorithms toward the Multi-agent Reinforcement Learning (MARL) ones. RL is regarded as a suitable framework to represent and solve control tasks, where decision making processes have to be supported or a behavior has to be enacted. Based on the control task, the applied algorithm is responsible to decide the actions to be taken to achieve a specified goal, while a reward is provided to the agent based on the effectiveness of the applied action. RL is considered suitable for guiding orchestration mechanisms in cloud and edge computing environments [14]. The MARL algorithms extend the RL approach to a setting where multiple agents interact within an environment to achieve a specific goal [9].

Two types of classifications apply to the MARL algorithms. The first classification is based on the type of the reward, where we can have cooperative or competitive MARL. In cooperative MARL, each agent learns a policy and follows a strategy to maximize a global or own reward, considering the behavior of the rest of the agents in a common environment and toward a common goal. In competitive MARL, the agents compete with each other to maximize their own reward and outperform one another rather than cooperate toward a common goal.

The second classification is based on whether the agents get trained and act in a centralized or decentralized way. In centralized training and centralized execution (CTCE) methods, there is a central controller that receives observations from all the agents, determines the joint action to be executed, and communicates it back to the agents. However, when the number of agents increases, the computational complexity and the communication overhead in the central controller also increases. On the other hand, in the case of distributed training and decentralized execution, each agent learns and takes decisions independently. In this case, the main disadvantage is the instability and difficulty to achieve convergence due to the dynamicity that exists in the environment and the difficulty of an agent to adapt to real-time changes in the behavior of other agents.

An intermediate solution between these two approaches has been proposed in the form of centralized training and decentralized execution (CTDE) methods. CTDE combines the benefits of centralized coordination during training for effective learning with decentralized decision-making during execution for adaptability and autonomy. The agents can exploit the global state information that is collected during the training phase by all agents, acquire a more abstract and adaptable understanding of the environment and, following, be able to take their own decisions based on decentralized local policies. In this way, the trained agents can generalize their learned behaviors and apply them in real-world situations where direct centralized coordination may not be possible. In case that the agents share and utilize an extensive set of parameters during the training phase, we refer to a strategy known as "parameter sharing" that helps the agents to learn from a shared representation, promoting coordination and enabling faster convergence. This strategy is mostly applicable in cases where the agents can have identical behavior over the environment.

Autoscaling Mechanisms Based on Cooperative AI

In this manuscript, we introduce, design and promote the use of an AI-assisted autoscaling synergetic orchestration mechanism. The proposed mechanism is targeted to distributed computing continuum environments and acts as a paradigm for the development of similar mechanisms in the future. A multi-agent approach is adopted to manage the horizontal scaling of the various resources across the continuum in a holistic way, enable decentralized decision-making, and foster adaptability and scalability in complex environments.

We are building upon the work presented in [15], where an AI-assisted autoscaling mechanism is provided for a single-agent setting. Extending this work, we consider the synergy among multiple AI agents to manage the scaling decisions for all the horizontally scalable microservices that compose an application graph. In this way, a complex problem of managing the scaling actions for the various components of an application graph is decomposed into more manageable tasks distributed among agents that collaborate among each other. Each agent is responsible to guide scaling actions for a specific microservice, while the agents exchange information regarding the performance of the overall application to properly guide their actions to achieve a global objective (e.g., high performance). In the scaling decisions we consider both resource consumption metrics at a local level (per microservice) and SLOs that are related to application level metrics. The aforementioned synergy among the AI agents is implemented based on the adoption of cooperative MARL techniques. In this way, there is no need for manual specification and management of scaling thresholds since decision making is driven by the AI agents.

Below we detail how various MARL algorithms can be used to support autoscaling in distributed application graphs, then we present the conceptualization of the MARL environment considering the state, the actions and the provided reward.

Further on we describe how a specific MARL algorithm — called as QMIX — can be applied over the specific environment.

Cooperative Multi Agent Reinforcement Learning for Autoscaling

In cooperative MARL, multiple agents learn to collaborate toward a common goal. In our case, the common goal is to guide the horizontal scaling of each one of the application components (microservices) in an application graph in such a manner that it leads to optimal provision of the overall application.

Considering the classification of the MARL algorithms above, we adopt a CTDE method to combine the benefits of centralized coordination during training with decentralized and independent decision making during execution, as depicted in Fig. 2. An AI Agent is attached to each microservice and is able to undertake decisions to define scaling thresholds for this microservice. The actions refer to the configuration of the scaling threshold per microservice, while the observations combine both local parameters (e.g., average CPU usage of the microservice) and global parameters (e.g., end to end latency of the application graph). We refer to a non-stationary environment, since the scaling actions provided by the agents can change the overall state of the environment or the rewards received by the agents. For instance, scaling of one microservice of the application graph may have positive impact on the end to end latency, and, thus, affect the rewards considered by the rest agents.

In the centralized training phase, each AI agent participates in centralized data collection and coordination, undertakes actions and maintains its local RL model. Part of the collected information is shared among the AI agents toward the development of a global RL model. The exact type and level of shared information, and the sharing mechanism (e.g., centralized entity that evaluates the joint actions of the agents, shared replay buffer accessible to all agents) depends on the MARL algorithm. Such information may include parameters for the trained ML models, such as weights and biases of neural network layers, parameters used to estimate state-action values, and parameters for configuring the exploration phase of a MARL algorithm, as well as data for the undertaken observations and actions. The extensive sharing of information regarding the parameters used by the AI agents for the training phase is beneficial in particular in scenarios where the agents can act in a homogeneous way. In the case of autoscaling, this should happen for agents that manage application components with similar profiles in terms of consumption of resources.

Once the global ML model is trained, each agent updates accordingly its local RL model and can use it for decision making during the decentralized execution. In this phase, each agent undertakes its actions based on its own observations and the learned policy, without explicit coordination with other agents.

MARL Environment Conceptualization

Aligned with the aforementioned reasoning and observations, the conceptualization of an application autoscaling mechanism as a multi-agent problem is

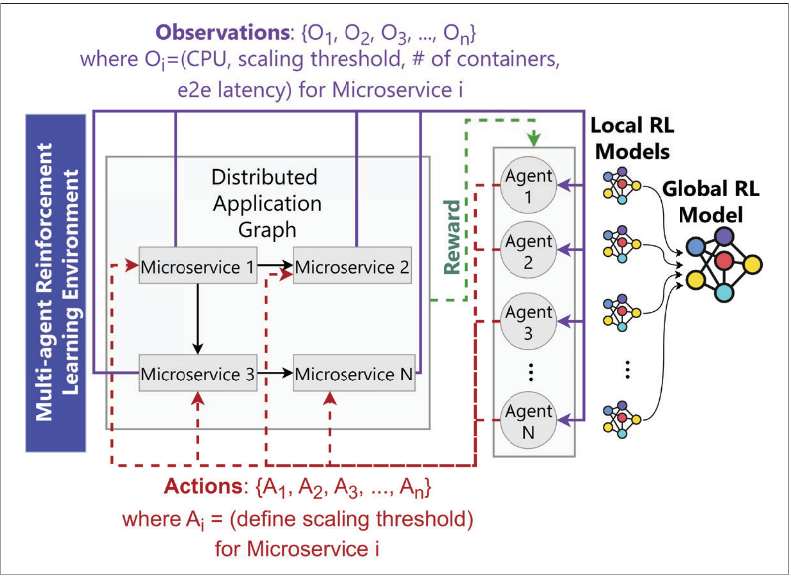
Observations: {O_1, O_2, O_3, ..., O_n}
where O_i=(CPU, scaling threshold, # of containers, e2e latency) for Microservice i
Multi-agent Reinforcement Learning Environment
Distributed Application Graph
Microservice 1
Microservice 2
Microservice 3
Microservice N
Reward
Agent 1
Agent 2
Agent 3
Agent N
Local RL Models
Global RL Model
Actions: {A_1, A_2, A_3, ..., A_n}
where A_i = (define scaling threshold) for Microservice i

FIGURE 2. MARL Agents for Autoscaling Management.

modeled as a cooperative multi-agent environment with a continuous state and discrete actions space.

The state space of the MARL environment consists of a combination of metrics that include resource usage metrics per microservice (CPU usage, number of deployed instances), QoS metrics for the considered application (end-to-end latency), and metrics related to configuration aspects of the scaling mechanisms (scaling threshold) per microservice. Specifically, each AI agent monitors the average percentage of the CPU usage of the active instances of a microservice. It also sets the CPU usage threshold for triggering a scaling action and monitors the number of deployed instances for each microservice. Regarding the QoS metrics, we monitor the end-to-end latency for serving a request by the application. The end-to-end latency usually is associated with the execution of processes across various microservices of an application and, thus, is considered as a metric measured for the entire application graph (global level), compared to the previously referred metrics that are measured per microservice (local level). A continuous state space is considered.

The actions space consists of a discrete set of actions for managing the scaling thresholds, where each threshold refers to a specific microservice of the distributed application. The possible actions of an agent are to raise the current CPU threshold by 20 percent, to keep it at the same level or to lower it by 20 percent. Depending on the observed utilization of CPU resources and the end-to-end latency, each agent receives a reward for the enforced action. The reward is a floating number in the range (0, 100) and is maximized when the utilization of resources is minimum and the latency is below a certain threshold. Actually, we penalize the addition of further containers for a specific microservice, as well as the existence of latency values that are more than 80 percent of the specified latency in the SLA. The penalty is larger in case of overpassing the defined value for the SLA latency. The goal of each agent is to achieve optimal usage of resources in the part of the computing continuum that it operates (local

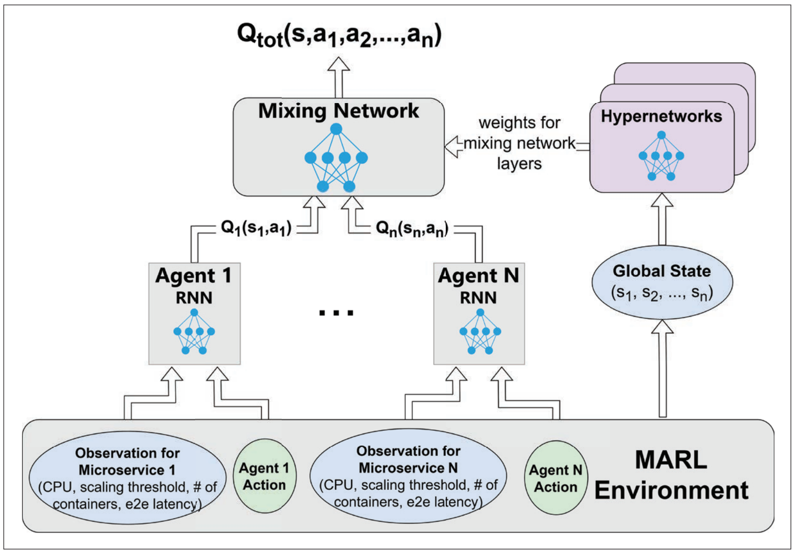
$Q_{tot}(s,a_1,a_2,\ldots,a_n)$
Mixing Network
weights for mixing network layers
Hypernetworks
$Q_1(s_1,a_1)$
$Q_n(s_n,a_n)$
Agent 1 RNN
Agent N RNN
Global State ($s_1, s_2, \ldots, s_n$)
Observation for Microservice 1 (CPU, scaling threshold, # of containers, e2e latency)
Agent 1 Action
Observation for Microservice N (CPU, scaling threshold, # of containers, e2e latency)
Agent N Action
MARL Environment

FIGURE 3. QMIX Algorithm Overview.

level), while maintaining an acceptable service provision level (global level).

QMIX Cooperative MARL Algorithm

The developed agent is based on the QMIX (Q-learning with Mixing) cooperative MARL algorithm [9]. QMIX is a learning algorithm that allows multiple agents to work together toward a common goal in a decentralized manner. QMIX is based on Q-learning, which is a popular RL algorithm used to teach agents how to make decisions in complex environments. In Q-learning, an agent learns to take actions that result in the highest expected reward. This is achieved based on the value function — referred to as Q-value function and denoted as Q(s,a) — that is a critical concept used to evaluate how good a particular state or state-action pair is for an agent in an environment. It represents the expected cumulative reward an agent can obtain from taking action a in state s, and then following a specific policy thereafter.

In the context of multi-agent systems, Q-learning is not considered as a wise choice since it treats each agent independently and does not consider the joint actions of the agents. This makes the application of Q-learning inappropriate for non-stationary environments where the action of one agent may significantly affect the other agents. Furthermore, Q-learning is considered computationally infeasible or inefficient in cases where the number of the state-action pairs becomes excessively large. The QMIX algorithm is a deep RL algorithm that overcomes these challenges. It enables the collaboration among agents to learn a centralized value function that captures the interdependencies among their individual actions, facilitating their effective coordination to achieve common goals. However, during execution, agents can follow decentralized policies. This balance between centralized learning and decentralized execution helps the agents to effectively coordinate their actions while maintaining scalability.

During the centralized training phase (Fig. 3), each agent maintains a recurrent neural network (RNN) to learn its individual Q-value function. The RNN represents the local RL model of an agent in a MARL algorithm. The learning is based on its local set of observations (CPU, scaling threshold, number of pods, latency) and the last individual scaling action. The selected actions by all the individual agents are used to determine the joint action that is taken by the QMIX algorithm. The joint action is passed through a mixing network that is a feed-forward neural network that represents the global RL model of a MARL algorithm. The mixing network takes as input the individual Q-value functions and mixes them monotonically to produce a centralized value function (expected joint reward) for all the agents (Qtot(s, a1, a2, ..., an)). The weights of the mixing network have to be non-negative. This constraint guarantees that the centralized value function is increasing monotonically based on the individual Q-values, simplifying the coordination among the agents and ensuring that the improvements in the individual policies of the agents lead to improvements in the joint policy.

To learn the weights of the mixing network, QMIX employs a technique called hypernetworks. A hypernetwork is a type of neural network that generates the parameters for another neural network. In the case of QMIX, hypernetworks are used to dynamically generate the non-negative weights of the mixing network based on the input state of the MARL environment. Each hypernetwork consists of a single linear layer, followed by an absolute value activation function so that it can follow the monotonicity constraint. The activation function used in our case regards a Rectified Linear Unit (ReLU) function that introduces non-linearity to the model, enabling it to learn complex patterns and representations in the data.

During the decentralized execution phase, each agent uses its individual Q-value function to independently select its own action based on its local microservice state and the last action that was undertaken. The actions of all agents are then combined to form the joint action of the system. The joint action is passed through the mixing network to estimate the centralized value function (expected joint reward). During this phase, the weights of the mixing network are fixed.

Multi Agent Reinforcement Learning Environment and Testbed Infrastructure

The OpenAI Gym framework has been used for modeling and managing the MARL environment, as an open source Python library for developing and comparing RL algorithms. The extended Python MARL framework (EPyMARL) has been used for the development of AI agents based on the QMIX algorithm.

To realize a set of experiments, a testbed infrastructure has been set up. The scaling mechanisms are implemented based on the Kubernetes Horizontal Pod Autoscaler (HPA). The Kubernetes HPA adjusts the number of replicas of each microservice based on the guidance provided by the QMIX algorithm. A Kubernetes cluster has been deployed, consisting of three nodes.

A sample distributed application has been developed that consists of three microservices and supports various graph analysis functions. The first microservice gets as input a number, creates two graphs with the provided number as their size, and provides a union of the created graphs. The second microservice receives the unified graph and performs link prediction, where new edges are added to the graph. The third microservice receives the previous graph and calculates the degree centralities for all the nodes. With the term degree centrality, we refer to the number of edges that each node participates. Per external request, the three microservices are executed in a sequential manner. The NGINX Ingress Controller has been used as a load balancer for each microservice. Stress testing of the application has taken place with the Vegeta HTTP load testing tool. A variable workload is used where the load follows a gradual increase and decrease pattern that is applied periodically. A monitoring infrastructure is deployed for getting time series data for resource usage and application metrics. The testbed infrastructure used for experimentation is illustrated in Fig. 4.

To speed up the training process for the agents, we have utilized a benchmarking and simulation-based approach, as the deployment of distributed applications and scaling actions can be time-consuming at each time-step. The approach involves deploying the aforementioned distributed application graph in a benchmarking environment, where agents with random policies collect historical traces over a set of epochs. The time period for each epoch is set to six minutes. This period includes the time required for sending the variable workload, applying any scaling action and examining the potential impact in the monitored metrics. The collected data is then enriched by applying probability distributions, and used to train agents with QMIX-based policies, resulting in a significant reduction in training time. No time constraint is applicable in the training of the agents, since the training is based on the produced data from the simulation process.

Evaluation Results

Performance evaluation results have been produced based on the deployment of the distributed application, the stress testing of the application with a variable workload and the training of the QMIX agents. We have examined the performance in terms of convergence capabilities of the QMIX agents, the learning rate and the achieved efficiency. With the term convergence capability, we refer to the capability to learn a policy that approaches an optimal policy that maximizes the cumulative reward over time. The proposed autoscaling solution is compared with a rule-based scaling approach where the scaling is triggered upon the overpassing of static thresholds for the CPU usage. Specifically, two threshold values are considered for the CPU usage, namely 60 percent and 80 percent.

In Fig. 5 we present the results for the efficiency of the QMIX algorithm. The left y-axis refers to the average percentage of the CPU utilization of the active replicas (instances) for all the microservices, as well as the provided reward by the MARL mechanisms per epoch. The right y-axis refers to the average number of active replicas per microservice, as well as the average number of latency violations per epoch. A good learning rate is noticed, since the provided reward is increased and stabilized at a level of around 75 percent. The usage of resources is also improved, since the agent tends to use fewer containers with higher average CPU usage, while simultaneously guaranteeing the achievement of the imposed SLA that

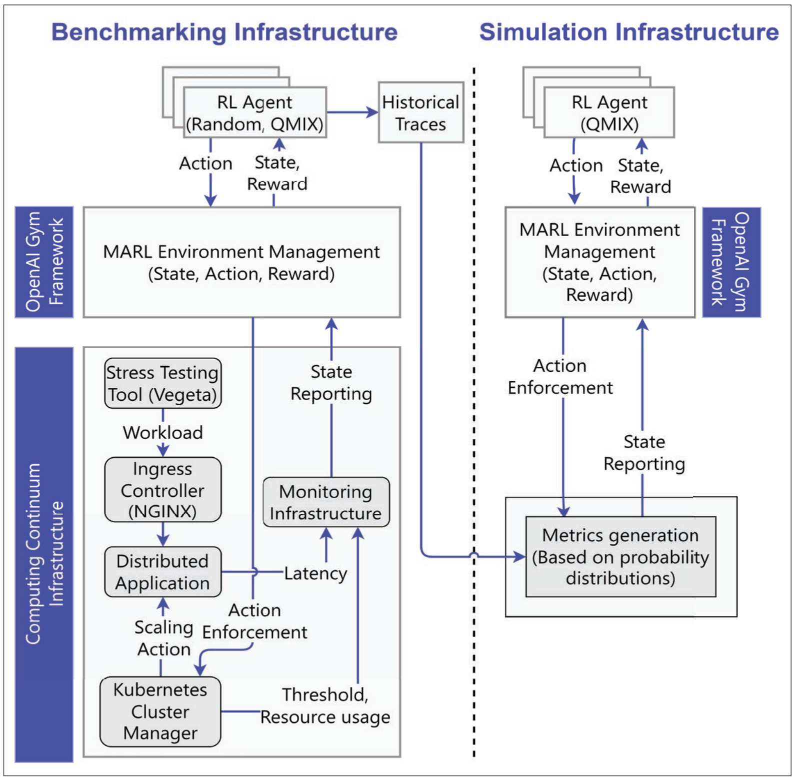
Benchmarking Infrastructure
Simulation Infrastructure
RL Agent (Random, QMIX)
Historical Traces
Action
State, Reward
OpenAI Gym Framework
MARL Environment Management (State, Action, Reward)
Computing Continuum Infrastructure
Stress Testing Tool (Vegeta)
Workload
Ingress Controller (NGINX)
Distributed Application
Scaling Action
Kubernetes Cluster Manager
State Reporting
Monitoring Infrastructure
Latency
Action Enforcement
Threshold, Resource usage
RL Agent (QMIX)
Action
State, Reward
MARL Environment Management (State, Action, Reward)
OpenAI Gym Framework
Action Enforcement
State Reporting
Metrics generation (Based on probability distributions)

FIGURE 4. Testbed and Experimentation Setup [15].

is set to 70ms. A reduction of the identified violations upon the achievement of convergence in the behavior of the agent is observed as well. In Fig. 6, we present the results for the efficiency in the case of application of the rule-based approach with the definition of static thresholds. The same metrics are depicted except for the values for the reward, since in this case, we do not apply an RL mechanism for guiding the scaling actions.

By comparing these results, we notice that the QMIX algorithm significantly outperforms the rule-based approach. In the case of the QMIX algorithm, the SLA violations are drastically reduced, since the average SLA violations per epoch are 0.3 compared to 1.49 and 1.24 in the case of static thresholds for values 60 percent and 80 percent respectively. With regards to the consumed computational resources, QMIX also shows similar or better behavior, given that the average replicas used per microservice is 3.08 compared to 3.58 and 3 in the case of static thresholds for values 60 percent and 80 percent respectively. In the case where the static threshold is set to 60 percent, more dynamicity is noticed in the number of active replicas during an epoch, compared to the case where the threshold is set to 80 percent. This makes it more challenging to achieve consistent compliance with the SLA due to oscillation effects, and potentially leads to a slightly higher average number of SLA violations. However, QMIX seems to also tackle this challenge, since the dynamicity that is introduced in the environment and the consideration of actions that can optimally manage it was considered during the centralized training phase of the QMIX algorithm. In this way, the AI agents are trained to properly react to the introduced dynamicity and undertake actions that can guarantee the assurance of the desired SLA.

Conclusions and Future Work

The development of synergetic orchestration mechanisms is promising for tackling challenges related to the optimal provisioning of distributed applications. In this manuscript, we introduced a synergetic orchestration mechanism for the autoscaling of resources in the computing continuum, considering the scaling needs of all the microservices within an application graph. A MARL environment and a relevant agent have been made available, while a set of evaluation results are generated. The produced results confirm the high potential that exists in the development of synergetic orchestration mechanisms for the computing continuum.

Several open research challenges are identified, including the need to generalize the proposed solution to any type of distributed application, the consideration of application graphs with larger number of horizontally scalable microservices, and the development of further synergetic orchestration mechanisms targeted to scheduling and cybersecurity management.

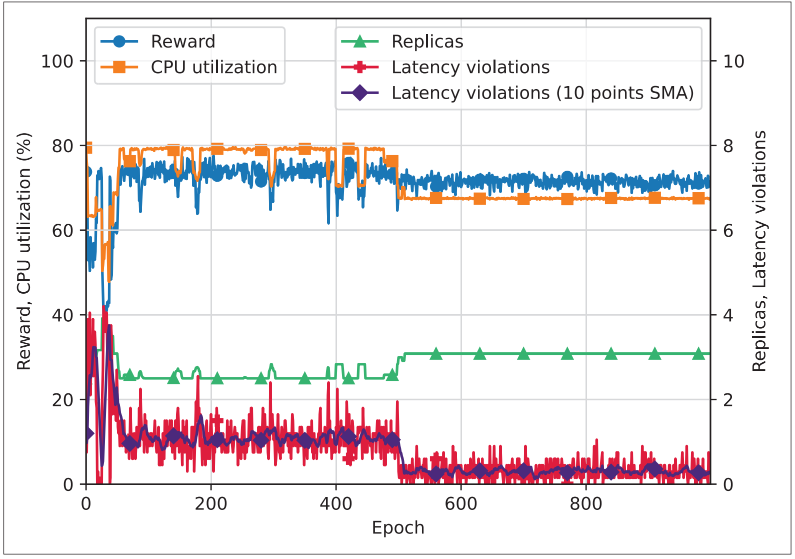

FIGURE 5. QMIX evaluation results.

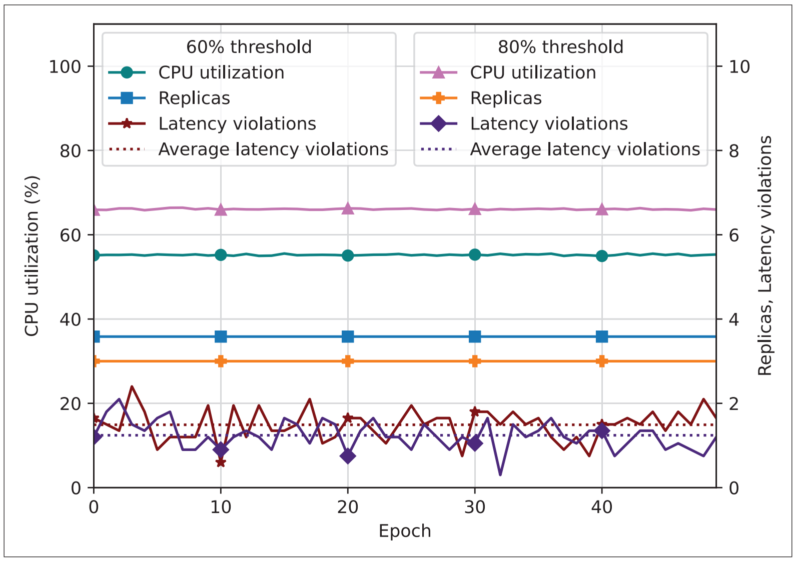

FIGURE 6. Evaluation results with static thresholds (60 % and 80 %).

Acknowledgment

This work has been partly funded from the European Union's Horizon Europe research and innovation programme under project NEPHELE (grant agreement No 101070487).

References

[1] O. Tomarchio, D. Calcaterra, and G. D. Modica, "Cloud Resource Orchestration in the Multi-Cloud Landscape: A Systematic Review of Existing Frameworks," *J. Cloud Computing*, vol. 9, no. 1, p. 49, Sept. 2020.

[2] B. Costa *et al.*, "Orchestration in Fog Computing: A Comprehensive Survey," *ACM Comput. Surv.*, vol. 55, no. 2, Jan. 2022.

[3] S. Dustdar *et al.*, "On Distributed Computing Continuum Systems," *IEEE Trans. Knowledge and Data Engineering*, 2022, pp. 1–1.

[4] D. Rosendo *et al.*, "Distributed Intelligence on the Edge-to-Cloud Continuum: A Systematic Literature Review," *J. Parallel and Distributed Computing*, vol. 166, 2022, pp. 71–94.

[5] D. Kimovski *et al.*, "Cloud, Fog, or Edge: Where to Compute?" *IEEE Internet Computing*, vol. 25, no. 4, 2021, pp. 30–36.

[6] S. Iftikhar *et al.*, "Ai-Based Fog and Edge Computing: A Systematic Review, Taxonomy and Future Directions," *Internet of Things*, vol. 21, 2023, p. 100674.

[7] J. Ménétrey *et al.*, "Webassembly as a Common Layer for the Cloud-Edge Continuum," *Proc. 2nd Workshop Flexible Resource and Application Management on the Edge, ser. FRAME '22*, New York, USA: Association for Computing Machinery, 2022, p. 3–8.

[8] K. Rzadca *et al.*, "Autopilot: Workload Autoscaling at Google," *Proc. Fifteenth European Conf. Computer Systems, ser. EuroSys '20*, New York, NY, USA: Association for Computing Machinery, 2020.

[9] K. Zhang, Z. Yang, and T. Başar, *Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms*, Cham: Springer International Publishing, 2021, pp. 321–84.

[10] C.B. Nielsen *et al.*, "Systems of Systems Engineering: Basic Concepts, Model-Based Techniques, and Research Directions," *ACM Comput. Surv.*, vol. 48, no. 2, Sept. 2015.

[11] K. Fu *et al.*, "Adaptive Resource Efficient Microservice Deployment in Cloud-Edge Continuum," *IEEE Trans. Parallel and Distributed Systems*, vol. 33, no. 8, 2022, pp. 1825–40.

[12] V.C. Pujol *et al.*, "Edge Intelligence — Research Opportunities for Distributed Computing Continuum Systems," *IEEE Internet Computing*, vol. 27, no. 4, 2023, pp. 53–74.

[13] I. Cohen *et al.*, "Dynamic Service Provisioning in the Edge-Cloud Continuum With Bounded Resources," *IEEE/ACM Trans. Networking*, vol. 31, no. 6, 2023, pp. 3096–3111.

[14] Y. Garí *et al.*, "Reinforcement Learning-Based Application Autoscaling in the Cloud: A Survey," *Engineering Applications of Artificial Intelligence*, vol. 102, 2021, p. 104288.

[15] A. Zafeiropoulos *et al.*, "Reinforcement Learning-Assisted Autoscaling Mechanisms for Serverless Computing Platforms," *Simulation Modelling Practice and Theory*, vol. 116, 2022, p. 102461.

Biographies

Anastasios Zafeiropoulos [M] received the Diploma and Ph.D. degrees from the School of Electrical and Computer Engineering, National Technical University of Athens (NTUA). He is a Postdoctoral Researcher at NTUA focusing on the fields of 5G, cloud/edge computing, IoT and machine learning technologies.

Nikos Filinis received his diploma in Electrical and Computer Engineering from the National Technical University of Athens (NTUA). He is currently pursuing his PhD at NTUA. His research interests lie in the areas of intelligent orchestration mechanisms, reinforcement learning and 5G networks.

Eleni Fotopoulou received her diploma in Electrical and Computer Engineering from the National Technical University of Athens (NTUA). She is currently pursuing her PhD at NTUA. She is a Senior Software Engineer with expertise on the development of intelligent orchestration in the 5G, cloud computing and IoT domains.

Symeon Papavassiliou [SM] is currently a Professor with the School of Electrical and Computer Engineering, National Technical University of Athens (NTUA). From 1995 to 1999, he was a Senior Technical Staff Member at AT&T Laboratories, NJ, USA. In August 1999, he joined the ECE Department, New Jersey Institute of Technology, USA, where he was an Associate Professor, until 2004. He has an established record of publications, while he has received several scientific awards and distinctions.