IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 23 August 2024, accepted 9 September 2024, date of publication 16 September 2024, date of current version 25 September 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3461149

RESEARCH ARTICLE

Efficient Microservice Deployment in the Edge-Cloud Networks With Policy-Gradient Reinforcement Learning

KEVIN AFACHAO[1], **ADNAN M. ABU-MAHFOUZ**[1,2], **(Senior Member, IEEE), AND GERHARD P. HANKE JR.**[3], **(Fellow, IEEE)**

[1]Department of Electrical, Electronic and Computer Engineering, University of Pretoria, Pretoria 0028, South Africa
[2]Council for Scientific and Industrial Research (CSIR), Pretoria 0083, South Africa
[3]Department of Computer Science, City University of Hong Kong, Hong Kong, SAR

Corresponding author: Kevin Afachao (u22851217@tuks.co.za)

This work was supported in part by Telkom, South Africa.

ABSTRACT The rise of user-centric design demands ubiquitous access to infrastructure and applications, facilitated by the Edge-Cloud network and microservices. However, efficiently managing resource allocation while orchestrating microservice placement in such dynamic environments presents a significant challenge. These challenges stem from the limited resources of edge devices, the need for low latency responses, and the potential for performance degradation due to service failures or inefficient deployments. This paper addresses the challenge of microservice placement in Edge-Cloud environments by proposing a novel Reinforcement Learning algorithm called Bi-Generic Advantage Actor-Critic for Microservice Placement Policy. This algorithm's ability to learn and adapt to the dynamic environment makes it well-suited for optimizing resource allocation and service placement decisions within the Edge-Cloud. We compare this algorithm against three baseline algorithms through simulations on a real-world dataset, evaluating performance metrics such as execution time, network usage, average migration delay, and energy consumption. The results demonstrate the superiority of the proposed method, with an 8% reduction in execution time, translating to faster response times for users. Additionally, it achieves a 4% decrease in network usage and a 2% decrease in energy consumption compared to the best-performing baseline. This research contributes by reproducing the Edge-Cloud environment, applying the novel Bi-Generic Advantage Actor-Critic technique, and demonstrating significant improvements over the state-of-the-art baseline algorithms in microservice placement and resource management within Edge-Cloud environments.

INDEX TERMS Edge computing, microservices, network optimization, online placement, scheduling algorithms, reinforcement learning.

I. INTRODUCTION

The ongoing shift toward a data-driven technological paradigm has led to the development of numerous devices and applications capable of harnessing extensive data [1]. This technological expansion, driven by the anticipation of future user needs, has resulted in resource-intensive applications such as online streaming, mobile gaming and mixed-reality experience services [2]. Initially, Cloud computing addressed resource scarcity because of the vast amounts of resources it offers. However, concerns over data privacy [3], the cost of Cloud resources [4], and transmission latency due to geographical distances [5] have prompted the proposal of a complementary infrastructure known as Edge-Cloud [6]. Unlike Cloud computing, which offers resources but may be latency-constrained, Edge computing typically has limited computational resources and storage. By effectively integrating both infrastructures, we can optimize performance by offloading demanding tasks to the cloud while leveraging the low latency capabilities of Edge computing.

The associate editor coordinating the review of this manuscript and approving it for publication was Loris Belcastro.

In response to user demands for ubiquitous access to applications across devices and scenarios, the Microservice Architecture (MSA) emerged [7], [8]. This architectural shift involves decoupling traditional applications into isolated and interdependent functionalities within container instances. However, implementing microservices faces challenges, mainly when one microservice failure can result in prolonged downtime compared to traditional applications. This failure is due to complex inter-dependencies [9].

The deployment of microservice applications in the Edge-Cloud presents a formidable challenge. The Edge-Cloud environment demands optimization of various competing performance factors because microservice applications are susceptible to failures. This optimization impacts the Quality of Service (QoS) in terms of latency and communication costs during scheduling and placement [10]. To address this joint optimization problem of resource allocation and service placement in the Edge-Cloud environment, Reinforcement Learning (RL) emerges as a promising solution [11]. RL, in contrast with other optimization techniques, learns from experience the optimal policy for the complex environment by accounting for the multiple interdependent factors which affect performance. RL is adaptive and suitable for a long-term use case, which is beneficial in the Edge-Cloud environment. This paper poses the central research question: "How can RL be implemented to enhance QoS by minimizing latency and resource cost for microservices placement in the Edge-Cloud?" The Bi-Generic Advantage Actor-Critic for Microservice Placement Policy (BAMPP), an RL algorithm, is chosen for its ability to model complex environments and make decisions that were previously unpredictable in attempts to solve this problem. The key contributions of this paper include:

1) The Edge-Cloud environment, including the complex interactions between edge devices and microservice applications, is modelled as an RL environment.
2) We propose a novel algorithm called Bi-Generic Advantage Actor-Critic for Microservice Placement Policy (BAMPP) to address the joint optimization problem of microservice placement and resource management. BAMPP extends the traditional Advantage Actor-Critic method by computing the critic network updates based on the state-value loss ratio, which improves the network's stability throughout the learning process. This innovation allows for more efficient and reliable resource allocation in Edge-Cloud environments.
3) We simulated real-world datasets and applications. The simulation models the Edge-Cloud behavior in the event of microservice application deployment.
4) Through simulations on EdgeSIMPy, BAMPP is demonstrated to be better than other baseline methods in achieving optimal microservice performance in the Edge Cloud.

The paper includes the following sections: Section II delves into preliminary studies, and Section III deals with system and application models. Section IV explains the problem formulation. Section V touches on the design of the RL model. Section VI describes the simulation setup. Section VII elaborates on the results and discusses them. Finally, Section VIII presents the conclusion.

II. RELATED WORK

The Edge-Cloud is a crucial paradigm to meet the demands of modern, latency-sensitive, and resource-intensive applications for the Internet of Things (IoT). This literature review examines recent research on optimizing various aspects of microservice deployment, resource management, and service orchestration to ensure QoS and efficient resource usage in Edge-Cloud environments. In light of the contribution of this paper, an overview of the literature is provided as shown in Table 1.

A. DYNAMIC APPROACHES

Several studies have explored dynamic algorithms for microservice placement and resource management. In their research, Pallewatta et al. [12] propose a decentralized microservices-based IoT application placement policy specifically designed for a heterogeneous and resource-constrained Edge-Cloud environment. Their work demonstrates a 40% improvement in application placement compared to the independent Edge or Cloud computing environment. In subsequent research, they affirm the benefits of decentralized over centralized placement and emphasize the need to address resource challenges in both queuing and placement decisions [13]. Similar to the adaptive policy formed by Faticanti et al. [14] introduce an algorithm called Fog Placement Algorithm (FPA). FPA is a cascade solution with two algorithms: one for throughput-oriented partitioning and another for microservice orchestration. The FPA's execution time and migration delay are comparable and scalable to baseline algorithms, but the results lack detailed numerical information.

Furthermore, the algorithm's performance is validated through experiments based on assumptions such as fixed throughput requirements and specific network configurations. These limitations hinder the findings' generalizability. He et al. [15] determined that to address the service placement problem for microservice orchestration; the issue needs to be formulated as a fractional polynomial problem and then resolved using a greedy-based algorithm. They observe that the problem of microservice placement arises due to the complex dependencies between microservices and the occasional presence of multiple instances of a single microservice within a container. Dependency chains between microservices within a container increase overall service execution time if an instance fails to instantiate. The authors evaluate their strategy on average response and execution times compared to a Genetic Algorithm (GA) and Random algorithm. They further analyze the performance of their algorithms with different system scales, including varying numbers of users, services, servers, and user requirements. Their approach notably reduces the computational complexity of microservice dependencies and demonstrates a superior performance in terms of speed.

Similar to Luo et al. [16] observe that microservice placement presents significant challenges due to the dynamic and complex dependencies, leading to inefficiencies in resource utilization and violations of Service Level Agreements (SLAs). SLAs are vital performance guarantees that service providers must meet to ensure the viability of their products. To address these issues, the authors present ERMS, an algorithm designed for proactive scheduling. ERMS profiles microservices based on latency and resource demand, enabling efficient resource allocation and improved performance. Their findings reveal that ERMS not only slashes resource utilization by over 40% compared to baselines but also successfully reduces the count of deployed containers by an impressive 58%. All this is achieved while preserving the end-to-end latency, ensuring a consistent user experience. While the studies by He et al. and Luo et al. share similarities with the current research due to their consideration of microservice dependencies, this paper expands upon their work by incorporating additional performance metrics, such as energy consumption. This broader perspective allows for a more comprehensive evaluation of the effectiveness of an RL-based algorithm when applied to this problem within the context of real-world data. Dynamic approaches show promising results in optimizing resource usage and application placement. These approaches allow adaptation to changing conditions, but may require substantial modifications to accommodate additional constraints.

B. NATURE-INSPIRED APPROACHES

Nature-inspired algorithms have been applied to address the complexities of microservice orchestration. In their study, Lin et al. [17] identify various challenges in optimizing scheduling schemes for microservices, including balancing resource load, minimizing network transmission overhead, and enhancing the reliability of microservice clusters. Their application scenario involves a single container with multiple instances of each microservice. To address these challenges, the authors present an Ant Colony Optimization (ACO) algorithm that considers device resources, microservice operational requirements, and failure rates. This algorithm aims to improve the overall performance of the system. While the algorithm outperformed GA, it is crucial to acknowledge that its performance may vary if applied to real-world applications or datasets. This discrepancy in performance is due to its limited generalizability, which could impact its effectiveness in different scenarios. Fan et al. [18] proposed a Particle Swarm Optimization (PSO) algorithm to address this challenge. Their methodology reduces network latency, improves service reliability, and accomplishes efficient load balancing. By iteratively eliminating infeasible particles from optimization, they achieved quicker convergence. Comparative results with other variants of PSO showed significant improvements in network latency, reliability and load balancing. It is worth noting that their algorithm's performance primarily depends on the number of user requests; with fewer than two requests, it performs similarly to baselines due to the smaller search space. This preference is because the number of infeasible particles retracted from the search space is consistent with the user requests; thus, smaller requests mean fewer particles. Given the limitations, Chen and Xiao [19] extend the research by pursuing a parallel PSO model. The parallel PSO extends the performance of a single PSO model. Their method leverages the fast convergence speed and fewer parameters of the PSO algorithm, combined with parallel computing and Pareto-optimal theory. Their approach aims to improve load balancing, reduce network transmission overhead, and enhance optimization speed. While their model shows performance gains, the added computational costs act as a bottleneck. The algorithm's partial load balancing performance is slightly worse than other algorithms.

Nevertheless, nature-inspired algorithms are reliable compared to simple deterministic algorithms such as the Random algorithm and produce reasonable solutions. For this reason, Wen et al. [20] employ GA to address the critical issue of reliability in microservice orchestration across geo-distributed remote data centers. Fundamentally, PSO draws inspiration from the collective behavior of large animal groups, like flocks of birds or schools of fish. Each solution, symbolized by a particle, dynamically adjusts its position based on its own experiences and those of its neighboring particles within the swarm. In contrast, GA mimics the process of natural selection by using operations like selection, crossover, and mutation to evolve solutions over generations. For the authors, a reliable microservice orchestration means a microservice orchestration that is progressively adaptive and secure from cyber-attacks. Their algorithm determines microservice allocation and deployment while monitoring the system's security through state-of-the-art methods. While they consider reliability, they focus on the setting of cybersecurity, while this paper considers reliability in the setting of microservice dependencies. Nature-inspired algorithms demonstrate effectiveness in handling multi-objective optimization problems in microservice management. They offer flexibility in accommodating multiple constraints but may require significant computational resources to find optimal solutions.

C. REINFORCEMENT LEARNING APPROACHES

Resource management is crucial in supporting mobile users running microservice applications in Edge computing. RL emerges as a prominent means to tackle this challenge. Several authors have explored various RL approaches to optimize Edge computing environments. Tang et al. [21] utilized a Markov Decision Process (MDP) for container migration to enhance mobile user experience. Their strategy outperformed existing approaches regarding delay but overlooked the impact of microservice dependencies crucial in Edge-Cloud computing scenarios. Conversely, Lv et al. [22] tackled the deployment of microservices in Edge computing environments. They aim to reduce overall service response time while maintaining load balance. Their novel Reward Sharing Deep Q Learning (RSDQL) algorithm aims to minimize communication overhead and achieve task load balance. Experiments validated RSDQL's superiority over existing methods regarding communication overhead and load balance, including GA, PSO, and Random Search (RS). However, other performance metrics, such as scalability and reliability, were not explicitly addressed. Yu et al. [23] focused on online microservice orchestration for IoT applications. They proposed a multi-objective Deep Reinforcement Learning (DRL) algorithm called MOTION, which simultaneously reduces energy consumption and service response time. MOTION demonstrated impressive service time, energy cost, and rewards performance but overlooked factors like reliability and scalability. Tong et al. [24] tackled the challenge of microservice autoscaling in Edge-Cloud environments using an RL method called Soft Actor-Critic (SAC). Their approach, Graph-based Joint Microservice Autoscaling (GJMA), utilizes spectral graph theory, graph convolutional networks, and multi-agent RL to enable servers to independently and collaboratively determine auto-scaling strategies. GJMA achieved superior average waiting time and 99 percentile latency compared to other algorithms, indicating quick response times.

Nevertheless, a comprehensive analysis of factors such as reliability and scalability remains crucial for a holistic understanding of Edge computing resource management. RL approaches show promise in adapting to dynamic environments and balancing multiple objectives. They offer a good trade-off between optimization and computational demand, settling for near-optimal solutions with less computational overhead compared to nature-inspired algorithms. Recent research in the Edge-Cloud framework and microservice placement has explored various approaches, from dynamic algorithms to nature-inspired optimization and RL. Each approach offers unique strengths: dynamic methods provide adaptable outcomes, nature-inspired algorithms excel at multi-objective optimization, and RL techniques offer adaptability to dynamic environments.

However, The microservices placement problem is an end-to-end challenge. The various studies presented target a sector of this pipeline. While these studies contribute to optimizing resource management in Edge computing by showcasing the performance gains in implementing these unique algorithms, there is a lack of a comprehensive solution that covers the entire system. Most comparisons are limited to their respective domains, highlighting improvements over traditional methods but not considering the broader scope of microservices orchestration [25]. This neglect limits the generalizability of their results, making it harder to decide on a uniform standard for microservice deployment under the constraint of resource management for the Edge-Cloud environment.

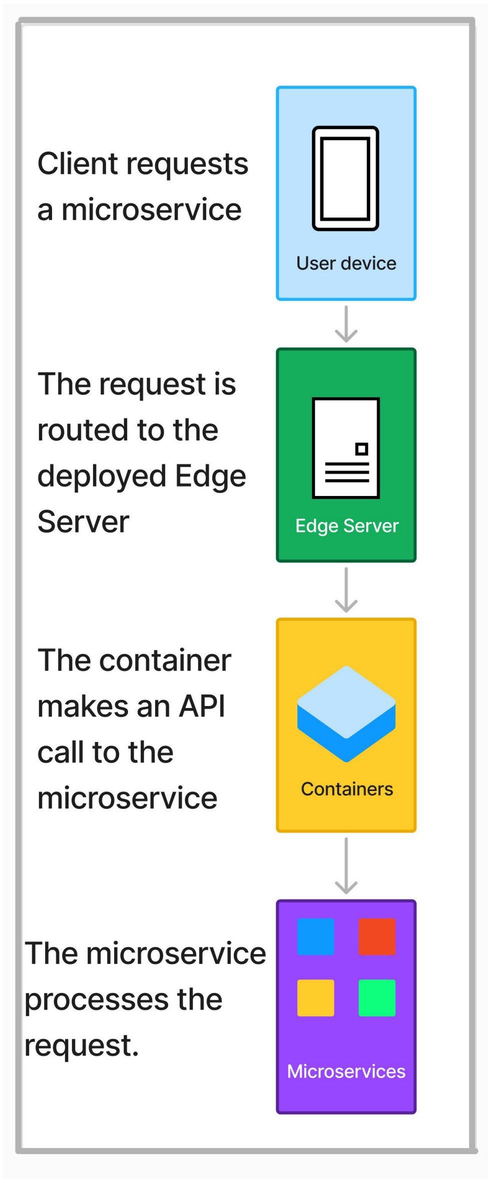
Client requests a microservice
User device
The request is routed to the deployed Edge Server
Edge Server
The container makes an API call to the microservice
Containers
The microservice processes the request.
Microservices

FIGURE 1. The diagram above illustrates the process involved in user requests of a microservice.

III. SYSTEM MODEL

This section, explores the microservice system architecture. The network architecture is the first component contributing to the system architecture, which describes the devices present and narrates their organization to achieve the microservices migration and placement. Next, the application model describes the applications considered in demonstrating our system's capabilities. The network usage model describes the use of the network to facilitate system communication during runtime. Additionally, there are two other

TABLE 1. Comparison of related works.

Reference	Proposed Method	Execution time	Network usage	Energy consumption	Reliability
[12]	Decentralized	✓	✓		
[14]	FPA	✓			
[15]	Greedy algorithm	✓			
[16]	ERMS	✓			
[17]	ACO		✓		✓
[18]	PSO	✓			✓
[19]	PSO		✓		✓
[20]	GA				✓
[21]	MDP	✓			
[22]	RSDQL		✓		
[23]	MOTION	✓		✓	
[24]	GJMA	✓			
This paper	BAMPP	✓	✓	✓	✓

TABLE 2. Summary of notations.

Notation	Description
T	The time taken for the application to execute
E_t	The current total energy utilized
E_{t-1}	The previous total energy value
μs_set	A set of unique microservices
μs_i	The ith microservice in μs_set
$\mu s\_relation$	A set of pairs of related microservices
$(\mu s_i, \mu s_j)$	A pair of related microservices i and j
r_i	The RAM resource required by microservice μs_i
c_i^t	The CPU resource required by microservice μs_i
$\langle r_i, c_i^t \rangle$	The microservice μs_i resource requirements
o_{ij}	The cost of communication between microservice i and j
U	A set of unique edge servers
u_p	The pth edge server
r_p	The RAM resource of the edge server u_p
c_p	The CPU resource of the edge server u_p
$\langle r_p, c_p \rangle$	The edge server u_p resource constraints
$res_{i,k}$	The resource k required by μs_i
$x_{i,p}$	The binary variable indicating if μs_i is placed
$res_{j,k}$	The resource k required by μs_j
$C_{p,k}$	The total resource capacity on u_p
$U_{available}$	The set of available edge servers.
$avg\_fail\_request$	The average failure rate of requests
$f_{i,t}$	The number of failed requests for μs_i
$s_{i,t}$	The number of successful requests for μs_i
sys_utils	The idle energy used by the edge device
$network\_usage$	The rate of utilization of the network
$payload_i$	The size of data transmitted
$trans\_latency$	The transmission latency incurred by the network link
b	The bandwidth of the network link

components: an energy consumption model that characterizes the energy expended by the system while executing services and communication costs incurred running the system. Fig.1. shows the main activities involved during the system response to a microservice request during runtime. The notations corresponding to these components are listed in the Table 2.

A. NETWORK ARCHITECTURE

This component describes the devices present and narrates their organization to achieve the microservice migration and placement. For this paper, we modeled the network architecture in three layers. The end layer comprises User Equipment (UE) such as smartphones and wearables, the edge layer comprises gateways, access points and micro data centers, and the cloud layer comprises remote data centers. In this paper, the devices vary in resource capacity; however, the overall view emphasizes the availability of resources the higher you go up the layer. In our network architecture, allocating resources from the edge layer to facilitate user service allocation is the main priority. Fig.2 provides an illustration of the Edge-Cloud architecture used in this paper.

B. APPLICATION MODEL

This component describes the applications considered in the demonstration of our system. The application model is based on the concept of microservices, which are small, independent, and loosely coupled services that communicate with each other through well-defined interfaces. The application model defines how the applications interact with each other.

We build the microservices in the context of the system's functionality. This construction is known as the bounded context. The microservices are then deployed in container instances, forming part of the application's directed acyclic graph (DAG). The application can be described as a tuple $\langle \mu s\_set, \mu s\_relation \rangle$ where the set of unique microservices and the relationship between microservices is $\mu s\_set = \{\mu s_1, \mu s_2, \ldots, \mu s_n\}$ and $\mu s\_relation$ respectively. Vertices and edges characterize the DAG. Fig. 3 displays an example of the DAG of an application. In this case, the vertices are the set of unique microservices, while the edges are the relationships between the microservices. When a microservice μs_i depends on another μs_j, they are considered related or interdependent, denoted as $(\mu s_i, \mu s_j) \in \mu s\_relation$. A microservice μs_i's resource properties within the application are represented by the tuple $\langle r_i, c_i^t \rangle$; the relationship between μs_i and μs_j, denoted $(\mu s_i, \mu s_j)$ includes resource cost such as communication overhead o_{ij}, RAM requirement r_i, and computational cost c_i^t required by the microservices respectively.

In this paper, we implement three microservice application instances with differing sizes to illustrate the performance of the proposed algorithm concerning how the number

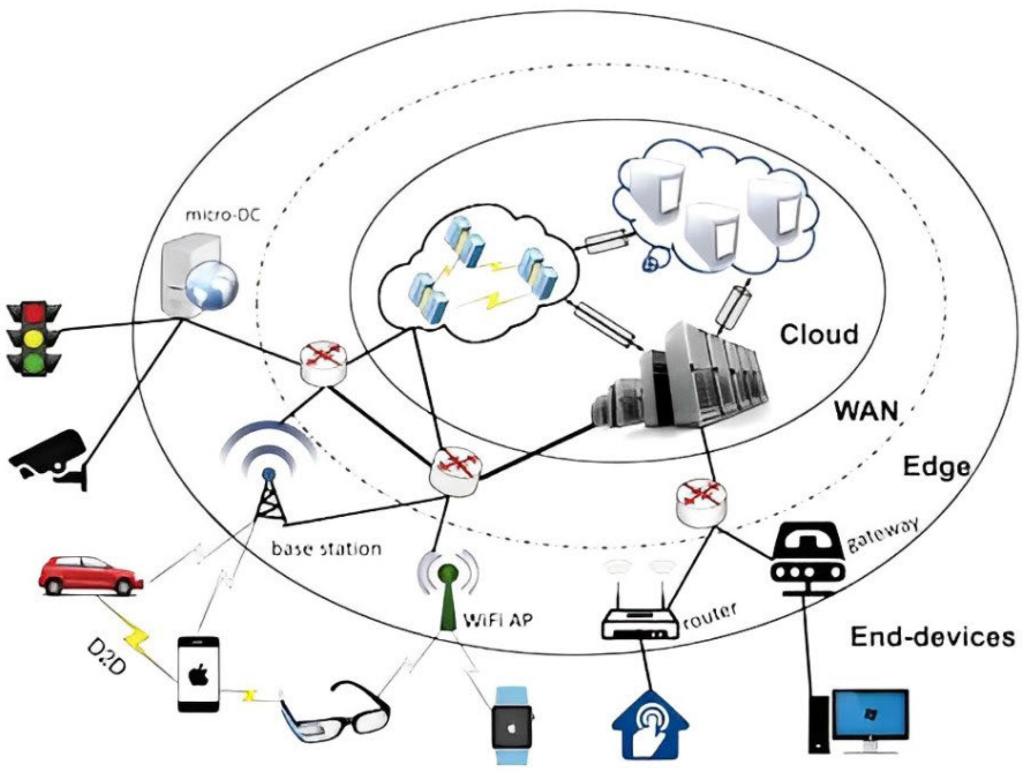

FIGURE 2. An illustration of the Edge-Cloud network based on [2].

of microservices in an application affects performance. Fig.1 displays how the microservices relate to the Network Infrastructure.

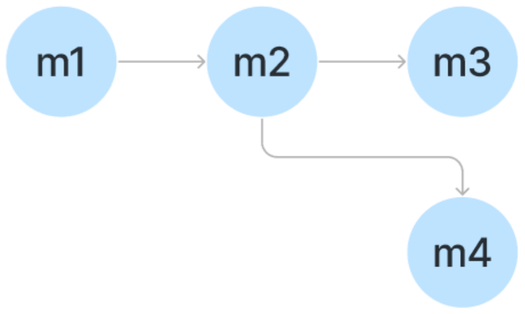

FIGURE 3. A simple microservice DAG.

C. RESOURCE UTILIZATION MODEL

In this paper, the system's operation is assumed to be according to a fixed length of time slots $T = \{0, 1, 2, \ldots, t\}$ where each t denotes a finite time unit in milliseconds. For this reason, each application's computational runtime is determined by the microservices' runtime. In this paper, the set of servers is represented as $U = \{u_1, u_2, u_3, \ldots, u_p\}$, each server u_p in the network has properties $\langle r_p, c_p \rangle$ where r_p is the fixed RAM of the server u_p and c_p is the computational limit of each server.

During runtime, each requested microservice is placed onto one of the available servers in the set $U_{available}$. This process includes checking if inserting the specified microservice into any server would not exceed its resource limitations, considering the existing resource usage on that server. The system makes the allocation decision by taking into account the device resource availability and latency between the server and the user, as well as respecting the dependencies between the microservices. The dependencies demand that should one service rely on another before it can be called on the server, the supporting microservice must be deployed to avoid additional latency and communication overheads. The server u_p is said to be available when its current resources allow for processing additional microservice requests. This relationship is mathematically expressed as;

$$\sum_{i=1}^{n} res_{i,k} \cdot x_{i,p} + res_{j,k} \leq Capacity_{p,k}, \quad \forall k \in R, \forall p \in U_{available} \quad (1)$$

Here, n is the number of microservices already allocated to the server. i indexes the already allocated microservices, j represents the new microservice being considered for allocation, and k references the different resources being considered. $res_{i,k}$ is the amount of resource k required by microservice μs_i. Also, $x_{i,p}$ is a binary variable indicating whether microservice μs_i is allocated to server p, $res_{j,k}$ is the amount of resource k required by the next microservice μs_j. $C_{p,k}$ is the total capacity of resource k on the server u_p. R is the set of RAM and computation resource types. $U_{available}$ is the set of all available servers.

In the event that a request is made to a device, the microservice controller checks the operation of the edge device to determine if it can run the microservice task with the resources available; if not, the task is forwarded to the next edge device. However, when the device is unavailable to be allocated a microservice, the microservice is terminated, and the request is considered a failure. The average failed request is expressed as;

$$avg\_fail\_request = \frac{1}{T}\sum_{t=1}^{T}\frac{\sum_{i=1}^{\mu s\_set} f_{i,t}}{\sum_{i=1}^{\mu s\_set}(s_{i,t}+f_{i,t})} \tag{2}$$

where $avg\_fail\_request$ is the average failure rate, T is the total runtime of the application, and $f_{i,t}$ is the number of failed requests for the microservice i during time interval t. $s_{i,t}$ is the number of successful requests considered for any instance of microservice μs_i during time interval t.

D. ENERGY CONSUMPTION MODEL

This component measures the system's energy expenditure during service execution. The energy consumption model accounts for energy used by the devices and network communication. The energy consumption model aims to minimize the total energy cost of the system while satisfying the user's performance expectations. The energy consumption can be expressed as;

$$E_t = E_{t-1} + sys\_utils \tag{3}$$

where E_t is the energy consumption of the edge device u_p at time t, E_{t-1} is the previous energy consumption of the current edge device u_p, and sys_utils is the idle energy used by the device.

E. COMMUNICATION MODEL

This component highlights the communication cost incurred in running the system. The communication model considers both the latency and bandwidth of the network links. The communication model aims to optimize the system's network performance while ensuring reliable performance and secure data transmission. The mathematical model for the network usage is given as follows;

$$network\_usage = \sum_{i=1}^{n}\frac{payload_i}{trans\_latency * b} \tag{4}$$

where n is the number of microservices on a deployed server, $payload_i$ is the size of the actual data transmitted from microservice μs_i, $trans\_latency$ is the transmission latency incurred by the network link, and b is the network link bandwidth.

IV. PROBLEM FORMULATION

The problem at hand is designing a task scheduling policy for an application's microservices before execution by edge devices. The objective is to select the optimal microservice-to-edge device pairing based on the properties of the edge devices and the tasks to maximize performance and cost efficiency. The optimization problem can be expressed as a set of objectives and a set of constraints defined as follows.

$$min\{average\ \ failed\ \ requests\}, \tag{5}$$

$$min\{network\ \ usage\} \tag{6}$$

$$subject\ to: \quad C1: \sum_{i=1}^{n} res_{i,r}\cdot x_{i,p} + res_{j,r} \leq Capacity_{p,r} \tag{7}$$

$$C2: \sum_{i=1}^{n} res_{i,c}\cdot x_{i,p} + res_{j,c} \leq Capacity_{p,c} \tag{8}$$

where n is the number of microservices allocated to the server, i indexes the already allocated microservices, j represents the new microservice being considered for allocation, r references the RAM resource, and c refers to the computational resource. $res_{i,r}$ is the amount of RAM r required by microservice μs_i and $res_{i,c}$ is the amount of CPU or computation c required by microservice μs_i. Also, $x_{i,p}$ is a binary variable indicating whether microservice μs_i is allocated to server p, $res_{j,r}$ is the amount of RAM r required by the next microservice μs_j and $res_{j,c}$ is the amount of CPU c needed for the next microservice μs_j $C_{p,c}$ is the total capacity of CPU c on the server u_p.

The Equations represent the optimization objectives of this paper, respectively: minimizing the average failed requests and minimizing the network usage. The Equations also represent the constraints of this paper, where C1 stands for the RAM constraint, and C2 stands for the CPU availability constraint.

V. MICROSERVICE PLACEMENT USING REINFORCEMENT LEARNING

In this section, we model an approximation of the joint optimization problem after the Markov property. The Markov property implies that future decisions depend only on the current state and action. Under RL, we formulate an agent that, at each time step t, occupies a specific state s_t in the environment and takes an action a_t. The decision policy π maps each action to a state. The agent receives a reward upon transitioning to the next state, s_{t+1}. This process happens repeatedly until the agent terminates. In the following subsections, first, a description of the agent's state, action and reward functions is given. Next, we will discuss the model-free agent utilized to generate the decision policy for the environment.

A. STATE SPACE, ACTION SPACE, REWARD

1) STATE SPACE

In the collaborative Edge-Cloud environment, the RL agent observes the state of the environment. The state space S consists of observations s_t where $t \in N$. We must observe system resources at each time step to determine if the constraints are met while pursuing the objective. For instance, when placing a microservice on a device, the state value or observation as seen by the agent is shown in Fig.4. The training process concludes when all microservices are placed on edge devices. Specifically, termination occurs as soon as there are no more microservices to be placed, indicating the completion of the learning phase. In a failed placement, the agent initiates failure recovery by either forwarding the microservice to the cloud device or placing it on another edge device. As shown in Fig.4, the device resources are considered a grid. We consider the microservice resources only when the edge devices can allocate utilization. The cloud data center is only viewed as a final resort. These observations are crucial for evaluating the system resources and ensuring constraints are met while pursuing objectives.

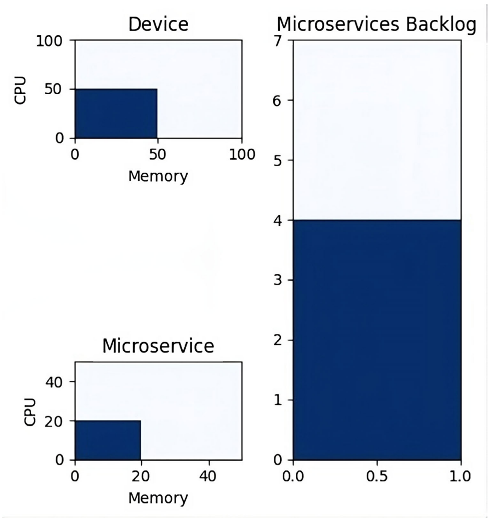

FIGURE 4. An example of a state observation by the agent during training.

2) ACTION SPACE

The action space $A = \{a_0, a_1, \ldots, a_n\}$ is constituted by the action values a_i. Agents select actions from this space in each step, influencing state transitions. Once an observation is made within the state space, the agent chooses an action from the action space based on the policy π in each time step. Over time, the environment evolves by altering the utilization of memory or CPU resources on edge devices, increasing device usage with each placement.

3) REWARD

RL aims to maximize the cumulative reward for all time slots over a discount factor of $0 < \gamma < 1$. The reward function, designed to minimize computational cost, failure requests, and network usage, is described in [26] as follows:

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^{T-1} r_T \tag{9}$$

The reward value is;

$$r = \begin{cases} 1, & \text{when constraints C1, C2 are met} \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

When the agent successfully places a microservice, it receives a reward of 1; otherwise, it receives a reward of 0. The goal is to maximize the expected return.

B. MARKOV DECISION PROCESS

In RL, the recursive actions of the agent within the environment according to a policy π to increase the possible accumulated reward is termed training. The policy π maps agent actions to environment states, determining which action is taken in each state. According to the problem formulation, the objective is to achieve minimal network usage, energy and execution latency despite the constraints during microservice deployment and placement. RL is proposed to achieve these joint objectives because it can effectively handle multiple goals simultaneously. Based on this premise, there are different approaches we can take to develop a policy that guarantees success. One approach suggests that we consider which actions under particular states would grant the most rewards and then apply this intuition by following the policy for the entire trajectory. Another approach focuses on identifying the optimal starting state to maximize rewards and then following the policy throughout the trajectory. A third approach proposes directly optimizing the policy itself to find the best decisions for the agent, maximizing rewards across the entire trajectory. Therefore, policy-based methods are particularly well-suited to address these challenges and will be the focus of this paper.

C. POLICY-BASED REINFORCEMENT LEARNING

A neural network with specific parameters generates a policy. We use the neural network to find the parameters that maximize the expected rewards in the environment through iterative parameter adjustment. The policy π_θ maps the state-action space $Q^{\pi_\theta}(s, a)$ to the state value $V^{\pi_\theta}(s)$, where θ represents the parameters learned by the policy gradient algorithm. How this works is that the policy gradient algorithms search for a local maximum in the state space by ascending the gradient of the policy with respect to the parameters.

Given a stochastic policy, the aim is to find the best parameters to satisfy the expectation (11). By calculating the gradient of the expectation, we adjust the policy parameters to favor sequences that lead to higher rewards.

$$J(\theta) = E_{\tau \sim p_\theta(\tau)}[\sum_t r(s_t, a_t)] \tag{11}$$

Actor-Critic methods refine rewards by adjusting the probability of actions based on their relative value compared to the average reward.

D. BI-GENERIC ADVANTAGE ACTOR-CRITIC FOR MICROSERVICE PLACEMENT POLICY

The policy gradient algorithm is an RL technique that optimizes the parameterized policy with respect to the expected long-term gains. We take the gradient of the policy with respect to the parameters to increase the probability of generating the best trajectory, which results in the most rewards. It is formulated from [27] as:

$$\nabla_\theta J(\theta) = \mathbb{E}[\nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s,a)] \tag{12}$$

where $Q^{*\theta}(s,a)$ is the long-term reward value, however, during training, the policy gradient attains high variance, which makes learning challenging to reach convergence. For this reason, a baseline is introduced: $B(s) = V^{\pi_\theta}(s)$, and the baseline is chosen to be the state-value function because of its feasibility, considering the already existing relationship between the state-action function and the state-value function (13).

$$Q^{\pi_\theta}(s_t, a_t) = \mathbb{E}[r_{t+1} + \gamma V^{\pi_\theta}(s_t + 1)] \tag{13}$$

An advantage function is introduced (14).

$$A^{\pi_\theta}(s,a) = Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s) \tag{14}$$

The advantage function represents the value at any given state where there is an advantage of choosing one action over the generally expected action. From (14), the policy gradient can now be written as

$$\nabla_\theta J(\theta) = \mathbb{E}[\nabla_\theta \log \pi_\theta(a|s) A^{\pi_\theta}(s,a)] \tag{15}$$

where $Q^{\pi_\theta}(s,a)$ serves as the actor by updating the policy distribution while $V^{\pi_\theta}(s)$ serves as the critic estimating the value function [27]. A neural network is utilized as a function approximator to derive the actor function [28]. The advantage estimation can struggle because it handles data and makes updates with step sizes, leading to slow and sometimes unreliable learning. For this reason, incorporating a penalty term limits the training to an acceptable range. The penalty term balances policy improvement and stability by guiding step sizes. The penalty term guides the policy gradient step sizes, helping to balance policy improvement and stability. In contrast to PPO, the critic is updated based on the loss between the predicted state value and the current state value; the BAMPP algorithm updates the critic based on the difference between the previous and current state value. Fig. 5 shows how these concepts fit together in addressing the problem.

VI. DESIGN AND IMPLEMENTATION

With the approach to microservice placement outlined, the next step involves simulating the Edge-Cloud environment. This simulation, described in detail below, models the optimization problem, and evaluates the proposed algorithm's performance. First, we describe the simulation setup in the context of the problem. Next, we detail the microservice applications considered.

Algorithm 1 BAMPP Algorithm

```
Input: Edge Servers information, Application information
Output: Decision Policy
1:  Initialize learning rate $a$, discount factor $\gamma$, clip range $\epsilon$
2:  Initialize actor network, $\pi(a|s,\theta)$ and critic network $V^\pi(s,w)$
3:  for episode $= 1, \ldots, T$ do
4:      for time slot$t = 1, \ldots, t_{\max}$ do
5:          Select action $a_t \sim \pi(\cdot|s_t, \theta)$
6:          Execute $a_t$, and observe the reward $r_t$ and the next state $s_{t+1}$
7:          Calculate the advantage function:
8:          $A^\pi(s_t, a_t) = r_t + \gamma V(s_{t+1}, w) - V(s_t, w)$
9:          Calculate the ratio of probabilities:
10:         $\text{ratio} = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$
11:         Update the actor network:
12:         $\theta \leftarrow \theta_{old} + a\nabla_\theta \log\left(\text{ratio}_{clipped} \cdot A^\pi(s_t, a_t)\right)$
13:         Update the critic network:
14:         $w \leftarrow w_{old} - a\nabla_w \frac{V}{V_{old}}$
15:     end for
16: end for
```

A. SIMULATION SETUP

The simulation setup specifies the configurations for the environment setup. The simulation is conducted using the EdgeSimPy platform [29], a Python programming environment. We chose EdgeSimPy because of its libraries, which enable the simulation of various resource management policies and deployment schemes for microservice applications. Since Python's RL toolkit is readily available, there's no need for external extensions.

We simulated the RL algorithms with the Stable Baselines package. We conducted the simulations on a Windows 11 PC with AMD Athlon Gold 3150U, Radeon Graphics 2.40 GHz, and 8 GB of RAM. EdgeSimPy can simulate complex environments with multiple network infrastructures ordered in hierarchies. This hierarchical structure is crucial for the simulation as it closely mimics real-world scenarios, representing both vertical and horizontal scaling of services during application deployment. In this hierarchy, we categorize the devices into levels. Level 0 represents a cloud data center, level 1 is a proxy server, level 2 is a gateway server, and level 3 corresponds to User Equipment (UE). The devices at levels 1 and 2 constitute the Edge environment, level 0 devices represent the Cloud infrastructure, and level 3 is for the user devices. This hierarchical arrangement reflects resource availability and geographical distance from the devices. The lower the level, the more resources the device has available. Although devices further from the user have lower levels, the number of devices closer is many and offers their resources

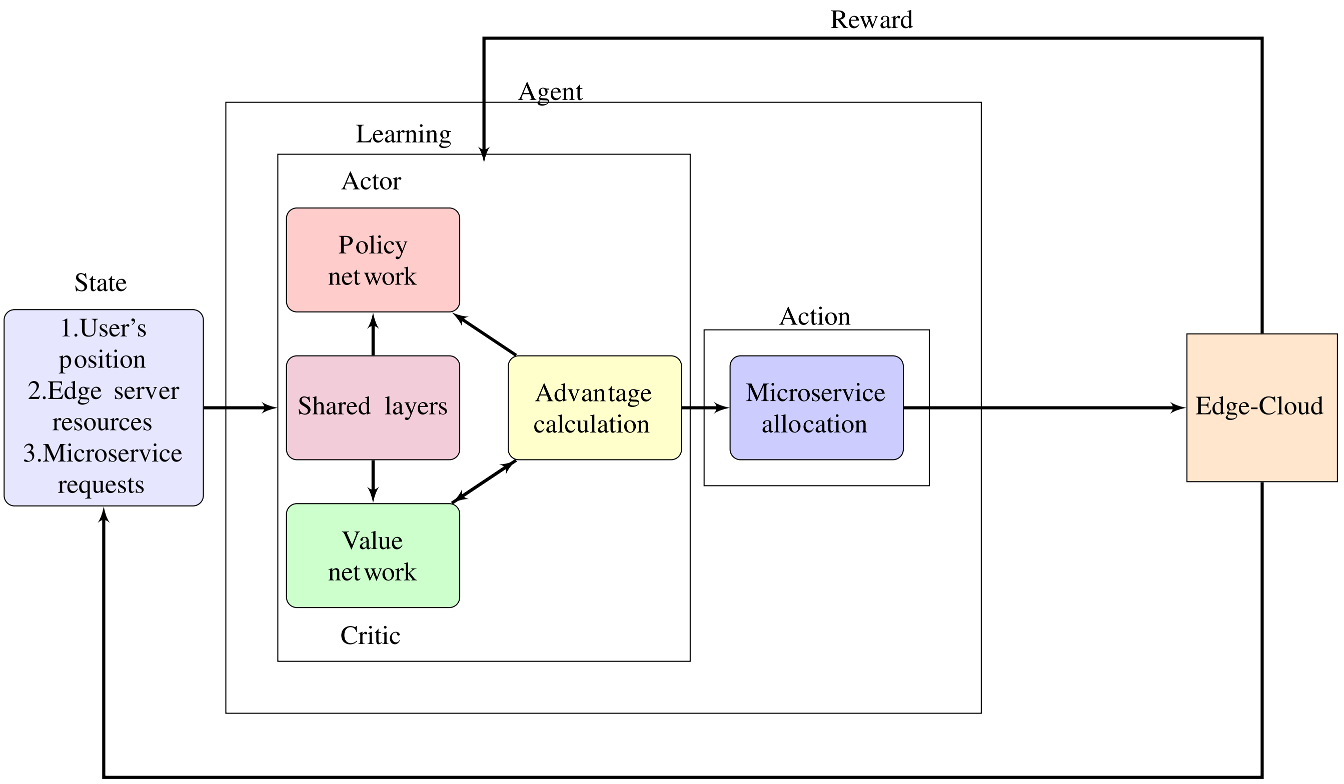

FIGURE 5. An illustration of the components of the BAMPP system.

to the UE. We spell out the details of the configurations for the network infrastructure in Table 3. In this simulation, we regard the entire scenario as occurring in the Melbourne Central Business District, with applications dependent on the Edge infrastructure and 132 devices unless the edge devices exceed their capacity. The next section details the specific configurations for the microservice applications.

B. APPLICATION CONFIGURATION

In the simulator, we created applications as microservices, with each microservice denoted as an application module. The microservices are then deployed in a single container instance using system-level virtualization. We consider three smart applications: the Cardiovascular Health Monitoring (CHM) application [30], the Drone Swarm Coordination (DRN) application [31] and the E-commerce (ECOM) application [31]. These applications generate synthetic workloads that model real-world applications. The CHM application has four microservices, DRN has 21 microservices, and ECOM has 34 microservices. The selection of these applications aimed to assess the proposed algorithm's scalability in terms of increasing microservices, workloads, and complex dependencies. Managing a microservice infrastructure becomes more challenging as the number of dependencies increases. After we built the microservice applications, we focused on the heart of the investigation: the experimental results and performance evaluation. The following section presents experimental results using configuration settings detailed in Table 3, Table4 and Table 5. These experiments compare our proposed RL algorithm with established baselines to evaluate its effectiveness in optimizing resources within a heterogeneous environment.

VII. EXPERIMENTAL RESULTS AND PERFORMANCE EVALUATION

In conducting these experiments, variations in the placement of the microservice occurs between the centralized placement on the cloud and the distributed placement at the network edge, with a preference for network edge placement. The experiments were run ten times, with the average taken and presented in the results. Moreover, we compare the proposed RL algorithm to the following baselines;

1) Integer Linear Programming Scheme(IPS) [13]: Integer Linear Programming models the optimization problem as a mathematical programming equation. By utilizing mathematical programming methods, we formulated the best placement order.
2) Proximal Policy Optimization (PPO) [27]: The PPO algorithm is an online policy-gradient RL method that improves the stability and convergence of the learning rate by maintaining the policy updates within a reasonable range. PPO is a variant of the Actor-Critic method, with the actor and critic sharing one neural network instead of separated networks.

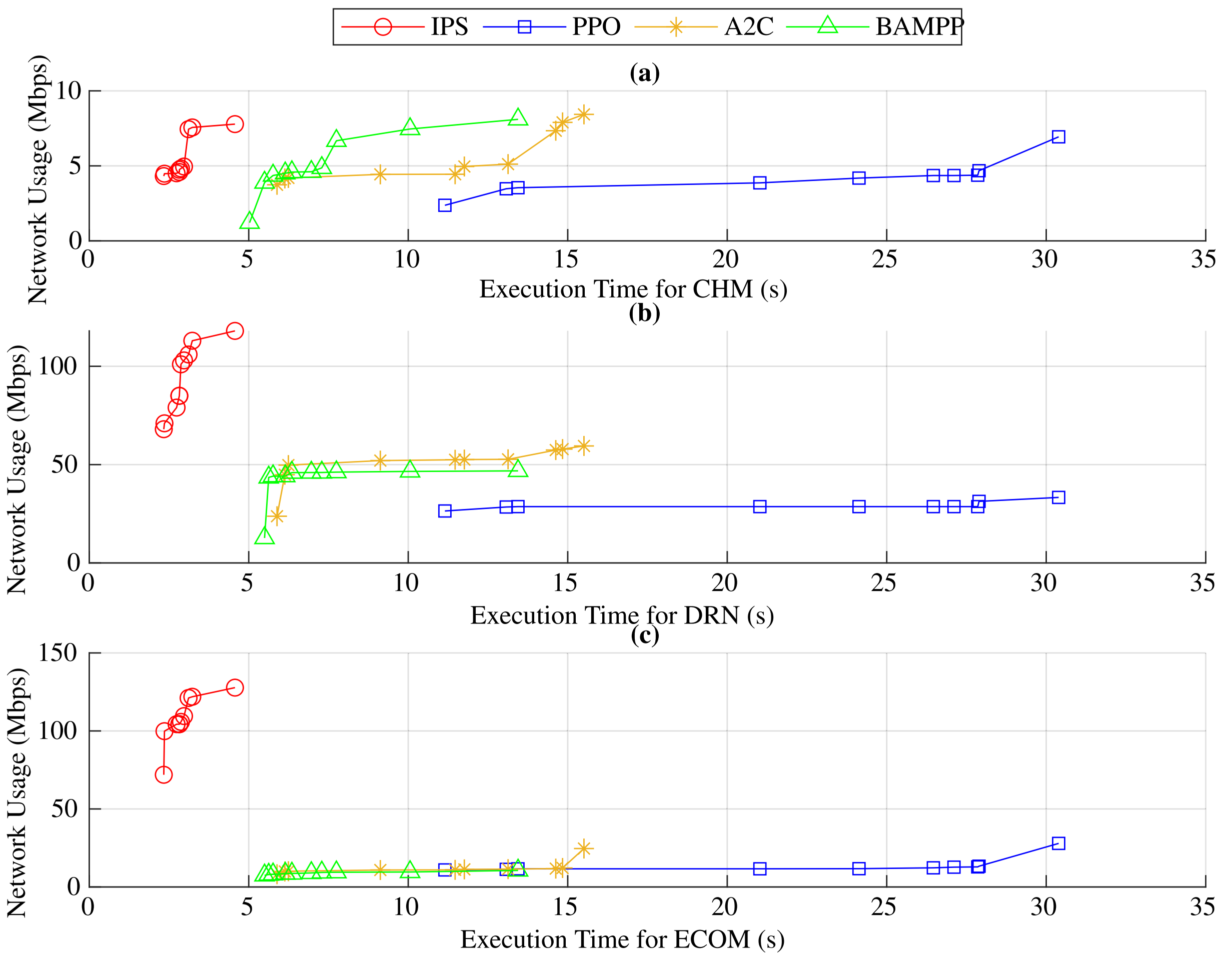

FIGURE 6. Network Usage against Execution time.

3) Advantage Actor-Critic (A2C) [32]: The A2C algorithm is an online policy-gradient algorithm. The actor is the policy function trained by the neural network, while the critic is a value function likewise trained by a separate neural network. The actor (policy) and critic (value function) estimate advantages, which help focus neural network updates during training on actions that lead to better-than-expected outcomes.

The algorithms represent the standard architectures present in microservice placement studies because of the prevalence of literature regarding their use. The simulations aim to determine which algorithm best optimizes resources for enhanced QoS within the heterogeneous environment. For this reason, we conducted the experiments to analyze the following: how well the network usage varies with the execution time, as shown in Fig.6; the average migration delay of microservices within deployment time, as shown in Fig.7, the average energy consumed within deployment time as shown in Fig.8 and the performance of the RL algorithms during evaluation show their reliability in Fig. 9. These metrics are important because they show how well the algorithms meet the objectives of the optimization problem while observing the constraints.

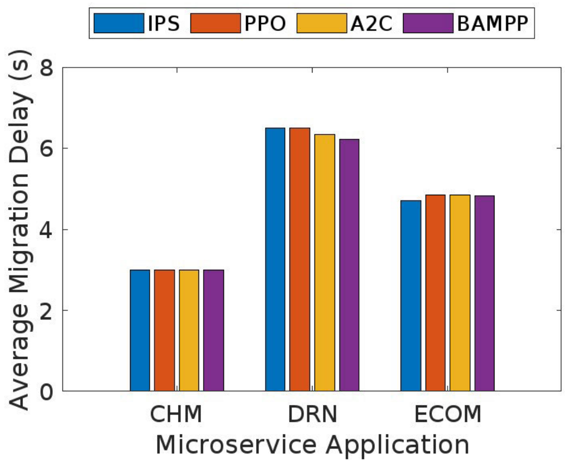

FIGURE 7. The figure above is the average migration delay recorded for microservice dependencies during application deployment.

Microservices are communication-intensive applications, meaning the individual microservices make many API calls to co-dependent microservices during execution. In the event that a particular microservice call fails to be created, the chain of dependencies risks failure. Other microservices that could be created in the container cannot complete their

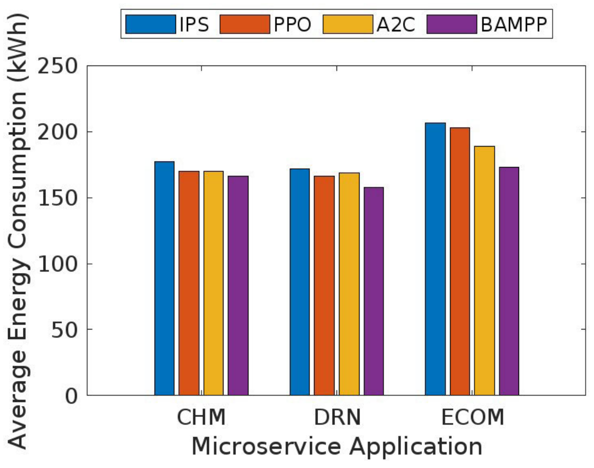

FIGURE 8. **The energy consumed by the edge devices during application deployment.**

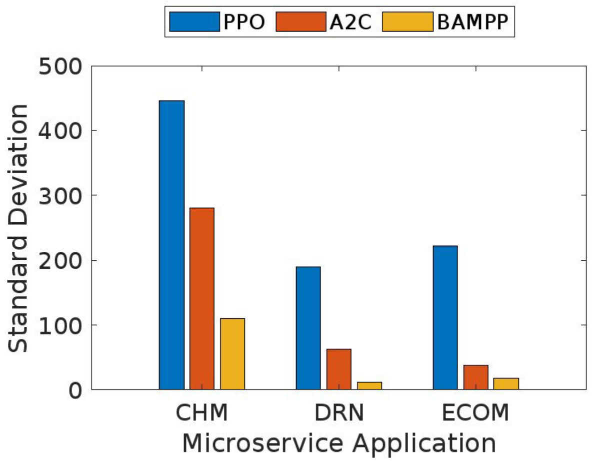

FIGURE 9. **The variation of reward scores during the evaluation of the algorithms.**

execution cycle without calling the microservice in their dependency chain; thus, the network bursts in congestion of API calls, flooding the network space and leading to an increase in communication overhead. The execution time for a microservice varies proportionally to the network usage, as shown in Fig.6. However, the execution of DRN and ECOM differ from the execution of CHM due to the impact of microservice failures. In DRN and ECOM, microservice failures increase network usage, leading to a spike in network congestion. The increased number of failures experienced by the IPS algorithm leads to a spike in network usage. This spike then results in a shorter execution time. The other algorithms, IPS, PPO, and A2C, follow a similar trend. However, the BAMPP algorithm consistently achieves a shorter execution time for all its microservices during the CHM scenario. As the number of microservices increases in DRN and ECOM scenarios, the performance advantage of the BAMPP algorithm becomes increasingly apparent. This consistent performance solidifies its position as the superior method among those compared.

Fig.7 compares the average migration delay in seconds of the four methods, IPS, PPO, A2C, and BAMPP, across the microservice applications. Across the CHM application, all four methods exhibit similar average migration delays, with the BAMPP algorithm demonstrating a slightly lower delay than the others. In the case of the ECOM application, the differences in performance become less pronounced as the Cloud Server consolidates the performance. The BAMPP algorithm demonstrates a lower average migration delay than the other three methods. The performance gap is even more significant for the DRN application, where the BAMPP algorithm showcases a 4% lower average migration delay compared to IPS and PPO while achieving a 2% reduction compared to A2C.

Fig.8 depicts the energy consumption of the edge devices for the execution of the three applications. The performance of CHM varies only slightly from DRN: in the execution of both applications, the cloud passively supplements the edge devices; hence, the edge devices do not consume much energy. In the execution of ECOM, the overwhelming number of microservices coupled with the nature of the algorithms reveals the disparity between IPS, PPO, A2C, and BAMPP. IPS, as a mathematical programming alternative, attempts to reconcile the vast number of microservices with the allocation between edge devices. Thus, it drives up the CPU utilization to ensure the microservices are distributed cost-effectively between the edge devices. As a result, energy consumption increases for the edge devices. However, PPO, A2C, and BAMPP cause edge devices to consume less energy by learning a placement strategy and adapting it to the scenario. In ECOM, the BAMPP algorithm achieves the lowest energy consumption of 172 KJ, representing an improvement of 16.5% over IPS, 14.9% over PPO, and 8.5% over A2C. The energy consumption is a metric that reflects the ability of the placement algorithm to balance between the edge devices under the prospect of long-term gains. Over the long term, edge devices are preferred to have energy management policies to prolong their lifespan. Edge devices lack the level of cooling given to cloud data servers, leading to a longer life span for the latter.

A problem with RL methods is that the empirical returns tend to have a large scale of values and lead to a high variance of the rewards. For example, given a discount value of 0.99 and a time horizon of 100 where a sample of the rewards are (2195, 2070, 1000, 2060) for PPO and (24995, 24983, 24987, 24992) for BAMPP, the high variation between the results of PPO shows a tendency to prioritize exploration over exploitation in an unstable manner compared to that of BAMPP. During deployment, PPO might exhibit both periods of significant success and abrupt failures. The results are explainable with little to no sudden failures, given the stable range of variation with the BAMPP algorithm. Specifically, BAMPP's architecture utilizes several key components. It incorporates clipped surrogate losses from PPO, advantage

TABLE 3. Configuration settings for simulation of CHM.

Microservice	CPU (GHz)	RAM (MB)	Usage rate (MB/s)	Dependencies
1	(0.1 0.3)	20	(2 16)	
2	(0.1 0.3)	5	(2 16)	[1]
3	(0.1 0.3)	10	(2 16)	[2]
4	(0.1 0.3)	15	(2 16)	[2]

TABLE 4. Configuration settings for simulation of DRN.

Microservice	CPU (GHz)	RAM (MB)	Usage rate (MB/s)	Dependencies
1	(0.1 0.3)	50	(2 32)	
2	(0.1 0.3)	100	(2 32)	[1]
3	(0.1 0.3)	80	(2 32)	[1,4]
4	(0.1 0.3)	60	(2 32)	[3]
5	(0.1 0.3)	120	(2 32)	[4,6]
6	(0.1 0.5)	90	(2 32)	[5]
7	(0.1 0.3)	150	(2 32)	[5,8]
8	(0.1 0.3)	70	(2 32)	[7]
9	(0.1 0.5)	180	(2 32)	[7,10]
10	(0.1 0.3)	110	(2 32)	[9]
11	(0.1 0.3)	40	(2 32)	[9,12]
12	(0.1 0.5)	80	(2 32)	[10,11]
13	(0.1 0.3)	60	(2 32)	[11]
14	(0.1 0.3)	90	(2 32)	[13]
15	(0.1 0.5)	120	(2 32)	[14]
16	(0.1 0.5)	200	(2 32)	[15]
17	(0.1 0.3)	300	(2 32)	[16, 18]
18	(0.1 0.5)	180	(2 32)	[17]
19	(0.1 0.5)	100	(2 32)	[17]
20	(0.1 0.3)	60	(2 32)	[19]
21	(0.1 0.3)	80	(2 32)	[20]

TABLE 5. Configuration settings for simulation of ECOM.

Microservice	CPU (GHz)	RAM (MB)	Usage rate (MB/s)	Dependencies
1	(0.1 0.3)	150	(2 32)	
2	(0.1 0.3)	80	(2 32)	[1]
3	(0.1 0.3)	100	(2 32)	[1]
4	(0.1 0.3)	100	(2 32)	[2]
5	(0.1 0.3)	60	(2 32)	[3]
6	(0.1 0.3)	40	(2 32)	[4]
7	(0.1 0.5)	60	(2 32)	[4]
8	(0.1 0.3)	90	(2 32)	[5]
9	(0.1 0.5)	60	(2 32)	[6]
10	(0.1 0.5)	100	(2 32)	[6]
11	(0.1 0.3)	70	(2 32)	[7,8]
12	(0.1 0.3)	75	(2 32)	[10,11]
13	(0.1 0.5)	100	(2 32)	[9]
14	(0.1 0.3)	100	(2 32)	[12]
15	(0.1 0.5)	90	(2 32)	[13]
16	(0.1 0.5)	20	(2 32)	[14]
17	(0.1 0.5)	80	(2 32)	[15, 16]
18	(0.1 0.3)	20	(2 32)	[17]
19	(0.1 0.5)	100	(2 32)	[17]
20	(0.1 0.3)	90	(2 32)	[17]
21	(0.1 0.3)	120	(2 32)	[18,19,20]
22	(0.1 0.3)	100	(2 32)	[21]
23	(0.1 0.3)	150	(2 32)	[21]
24	(0.1 0.3)	100	(2 32)	[21]
25	(0.1 0.3)	150	(2 32)	[22,23,24]
26	(0.1 0.5)	100	(2 32)	[25]
27	(0.1 0.3)	60	(2 32)	[25]
28	(0.1 0.3)	100	(2 32)	[25]
29	(0.1 0.3)	80	(2 32)	[26,27,28]
30	(0.1 0.3)	80	(2 32)	[29]
31	(0.1 0.3)	80	(2 32)	[29]
32	(0.1 0.3)	100	(2 32)	[29]
33	(0.1 0.3)	80	(2 32)	[32]
34	(0.1 0.3)	100	(2 32)	[33]

function approximation from A2C, and state value loss calculations for the critic network, enabling efficient and stable learning. Fig 9 shows the reward variation of the various RL methods when evaluated for an equal time horizon of

1000 episodes. The BAMPP algorithm performs better in all three scenarios than the others due to its more stable updates.

The performance of the BAMPP algorithm shows the performance gains RL brings to the placement of microservices. The neural network used in the policy-gradient methods is a function approximator that allows the agent to converge on a function faster than the deterministic approach, which requires many iterations to approximate the function. The RL agents efficiently learn the environment through exploratory and exploitative interaction with the environment, which allows policies to be developed that consider the context of each microservice and the placement decision that would favor it. The preference for BAMPP over the others is evident from the results. Also, the RL algorithms performed better than the IPS algorithm in terms of execution time and network usage costs, migration delay, energy consumption and reliability. In CHM execution, migration delays were equal across all four algorithms. However, BAMPP experienced a 6% improvement over IPS and a 2% improvement over both PPO and A2C in energy consumption.

VIII. CONCLUSION AND FUTURE WORK

This paper presented the need for resource management in microservice scheduling and placement in the Edge-Cloud. For this reason, we considered an investigation into the optimization of microservices during execution with RL. The two proposed algorithms under RL were Q-learning and A2C. With the environment specified and the problem well described, a simulation was carried out with the iFogSim tool to evaluate the performance of these proposed algorithms with respect to the Scalable placement algorithm and the Mixed Integer Linear Programming algorithm. The simulation considered the following metrics: network usage, execution time, failure of inter-microservice dependencies and the average energy consumption. The simulation modeled three smart application microservices: the Cardiovascular Health Monitoring application, the Drone Swarm Coordination application, and the E-commerce application. During the execution of the E-commerce application, the BAMPP algorithm experienced a gain of 91% over IPS, 35% over PPO, and 27% over A2C in network usage. Overall, the RL methods performed better than the other baseline methods. In a future paper, the scope of applications examined would be more comprehensive, considering nature-inspired algorithms like Particle Swarm Algorithm, different kinds of microservice dependencies as well as numbers of microservices to establish the point where the performance of the algorithms begins to wane under fixed Edge-Cloud resources.

APPENDIX: SIMULATION PARAMETERS

See Tables 3–5.

REFERENCES

[1] M. De Donno, K. Tange, and N. Dragoni, "Foundations and evolution of modern computing paradigms: Cloud, IoT, edge, and fog," *IEEE Access*, vol. 7, pp. 150936–150948, 2019.

[2] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proc. IEEE*, vol. 107, no. 8, pp. 1738–1762, Aug. 2019.

[3] M. Mukherjee, R. Matam, C. X. Mavromoustakis, H. Jiang, G. Mastorakis, and M. Guo, "Intelligent edge computing: Security and privacy challenges," *IEEE Commun. Mag.*, vol. 58, no. 9, pp. 26–31, Sep. 2020.

[4] M. Fazio, A. Celesti, R. Ranjan, C. Liu, L. Chen, and M. Villari, "Open issues in scheduling microservices in the cloud," *IEEE Cloud Comput.*, vol. 3, no. 5, pp. 81–88, Sep. 2016.

[5] H. Tabatabaee Malazi, S. R. Chaudhry, A. Kazmi, A. Palade, C. Cabrera, G. White, and S. Clarke, "Dynamic service placement in multi-access edge computing: A systematic literature review," *IEEE Access*, vol. 10, pp. 32639–32688, 2022.

[6] T. L. Duc, R. G. Leiva, P. Casari, and P.-O. Östberg, "Machine learning methods for reliable resource provisioning in edge-cloud computing: A survey," *ACM Comput. Surv.*, vol. 52, no. 5, pp. 1–39, Sep. 2020.

[7] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: Yesterday, today, and tomorrow," in *Present and Ulterior Software Engineering*. Cham, Switzerland: Springer, 2017.

[8] S. Hassan and R. Bahsoon, "Microservices and their design trade-offs: A self-adaptive roadmap," in *Proc. IEEE Int. Conf. Services Comput. (SCC)*, Jun. 2016, pp. 813–818.

[9] R. Heinrich, A. van Hoorn, H. Knoche, F. Li, L. E. Lwakatare, C. Pahl, S. Schulte, and J. Wettinger, "Performance engineering for microservices: Research challenges and directions," in *Proc. 8th ACM/SPEC Int. Conf. Perform. Eng. Companion*, Apr. 2017, pp. 223–226.

[10] I.-D. Filip, F. Pop, C. Serbanescu, and C. Choi, "Microservices scheduling model over heterogeneous cloud-edge environments as support for IoT applications," *IEEE Internet Things J.*, vol. 5, no. 4, pp. 2672–2681, Aug. 2018.

[11] P. Zhou, G. Wu, B. Alzahrani, A. Barnawi, A. Alhindi, and M. Chen, "Reinforcement learning for task placement in collaborative cloud-edge computing," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2021, pp. 1–6.

[12] S. Pallewatta, V. Kostakos, and R. Buyya, "Microservices-based IoT application placement within heterogeneous and resource constrained fog computing environments," in *Proc. 12th IEEE/ACM Int. Conf. Utility Cloud Comput.*, Dec. 2019, pp. 71–81.

[13] S. Pallewatta, V. Kostakos, and R. Buyya, "MicroFog: A framework for scalable placement of microservices-based IoT applications in federated fog environments," *J. Syst. Softw.*, vol. 209, Mar. 2024, Art. no. 111910. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121223003059

[14] F. Faticanti, F. De Pellegrini, D. Siracusa, D. Santoro, and S. Cretti, "Throughput-aware partitioning and placement of applications in fog computing," *IEEE Trans. Netw. Service Manage.*, vol. 17, no. 4, pp. 2436–2450, Dec. 2020.

[15] X. He, Z. Tu, M. Wagner, X. Xu, and Z. Wang, "Online deployment algorithms for microservice systems with complex dependencies," *IEEE Trans. Cloud Comput.*, vol. 11, no. 2, pp. 1746–1763, Jul. 2023.

[16] S. Luo, H. Xu, K. Ye, G. Xu, L. Zhang, J. He, G. Yang, and C. Xu, "Erms: Efficient resource management for shared microservices with SLA guarantees," in *Proc. 28th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.* New York, NY, USA: Association for Computing Machinery, Dec. 2022, pp. 62–77.

[17] M. Lin, J. Xi, W. Bai, and J. Wu, "Ant colony algorithm for multi-objective optimization of container-based microservice scheduling in cloud," *IEEE Access*, vol. 7, pp. 83088–83100, 2019.

[18] G. Fan, L. Chen, H. Yu, and W. Qi, "Multi-objective optimization of container-based microservice scheduling in edge computing," *Comput. Sci. Inf. Syst.*, vol. 18, no. 1, pp. 23–42, 2021.

[19] X. Chen and S. Xiao, "Multi-objective and parallel particle swarm optimization algorithm for container-based microservice scheduling," *Sensors*, vol. 21, no. 18, p. 6212, Sep. 2021.

[20] Z. Wen, T. Lin, R. Yang, S. Ji, R. Ranjan, A. Romanovsky, C. Lin, and J. Xu, "GA-Par: Dependable microservice orchestration framework for geo-distributed clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 1, pp. 129–143, Jan. 2020.

[21] Z. Tang, X. Zhou, F. Zhang, W. Jia, and W. Zhao, "Migration modeling and learning algorithms for containers in fog computing," *IEEE Trans. Services Comput.*, vol. 12, no. 5, pp. 712–725, Sep. 2019.

[22] W. Lv, Q. Wang, P. Yang, Y. Ding, B. Yi, Z. Wang, and C. Lin, "Microservice deployment in edge computing based on deep Q learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 11, pp. 2968–2978, Nov. 2022.

[23] S. Yu, X. Wang, and R. Langar, "Computation offloading for mobile edge computing: A deep learning approach," in *Proc. IEEE 28th Annu. Int. Symp. Pers., Indoor, Mobile Radio Commun. (PIMRC)*, Oct. 2017, pp. 1–6.

[24] G. Tong, C. Meng, S. Song, M. Pan, and Y. Yu, "GMA: Graph multi-agent microservice autoscaling algorithm in edge-cloud environment," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, vol. 14, Jul. 2023, pp. 393–404.

[25] L. Chen, Y. Xu, Z. Lu, J. Wu, K. Gai, P. C. K. Hung, and M. Qiu, "IoT microservice deployment in edge-cloud hybrid environment using reinforcement learning," *IEEE Internet Things J.*, vol. 8, no. 16, pp. 12610–12622, Aug. 2021.

[26] S. S. Mousavi, M. Schukat, and E. Howley, "Deep reinforcement learning: An overview," in *Proc. SAI Intell. Syst. Conf. (IntelliSys)*, vol. 2. Cham, Switzerland: Springer, Sep. 2018, pp. 426–440.

[27] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.

[28] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.

[29] P. S. Souza, T. Ferreto, and R. N. Calheiros, "EdgeSimPy: Python-based modeling and simulation of edge computing resource management policies," *Future Gener. Comput. Syst.*, vol. 148, pp. 446–459, Nov. 2023.

[30] R. Mahmud, S. Pallewatta, M. Goudarzi, and R. Buyya, "IFogSim2: An extended iFogSim simulator for mobility, clustering, and microservice management in edge and fog computing environments," *J. Syst. Softw.*, vol. 190, Aug. 2022, Art. no. 111351.

[31] L. Sun, Y. Li, and R. A. Memon, "An open IoT framework based on microservices architecture," *China Commun.*, vol. 14, no. 2, pp. 154–162, Feb. 2017.

[32] X. Tian, H. Meng, Y. Shen, J. Zhang, Y. Chen, and Y. Li, "Dynamic microservice deployment and offloading for things–edge–cloud computing," *IEEE Internet Things J.*, vol. 11, no. 11, pp. 19537–19548, Jun. 2024.

KEVIN AFACHAO received the B.Sc. degree in telecommunications engineering from the Kwame Nkrumah University of Science and Technology, in 2019, and the B.Eng. degree (Hons.) in computer engineering from the University of Pretoria, Pretoria, South Africa, in 2022, where he is currently pursuing the M.Eng. degree in computer engineering, with research interests include the Internet of Things, edge computing, and artificial intelligence.

ADNAN M. ABU-MAHFOUZ (Senior Member, IEEE) received the M.Eng. and Ph.D. degrees in computer engineering from the University of Pretoria, Pretoria, South Africa, in 2005 and 2011, respectively.

He is currently a Chief Researcher and the Centre Manager of the Emerging Digital Technologies for 4IR (EDT4IR) Research Centre, Council for Scientific and Industrial Research; an Extraordinary Professor with the University of Pretoria; and a Professor Extraordinaire with the Tshwane University of Technology, Pretoria. His research interests include wireless sensor and actuator networks, low power wide area networks, software-defined wireless sensor networks, cognitive radio, network security, network management, and sensor/actuator node development. He is a member of many IEEE technical communities. He is the Section Editor-in-Chief of *Journal of Sensor and Actuator Networks* and an Associate Editor of IEEE INTERNET OF THINGS JOURNAL, IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, IEEE TRANSACTIONS ON CYBERNETICS, and IEEE ACCESS.

GERHARD P. HANKE JR. (Fellow, IEEE) received the B.Eng. and M.Eng. degrees in computer engineering from the University of Pretoria, Pretoria, South Africa, in 2002 and 2003, respectively, and the Ph.D. degree in computer science from the Security Group, Computer Laboratory, University of Cambridge, Cambridge, U.K., in 2009. He is currently a Professor with the Department of Computer Science, City University of Hong Kong, Hong Kong, China, and an Extraordinary Professor with the University of Pretoria. His current research interests include system security, distributed sensing applications, and the Industrial Internet of Things.