




Swarm Intelligence-Based Algorithm for Workload Placement in Edge-Fog-Cloud Continuum

Kefan Wu¹^a, Abdorasoul Ghasemi²^b and Melanie Schranz¹^c

¹Lakeside Labs, Klagenfurt, Austria

²Research Centre for Computational Sciences and Mathematical Modelling, Coventry University, Coventry, U.K.
{wu, schranz}@lakeside-labs.com, ae4959@coventry.ac.uk

Keywords: Swarm Intelligence, Edge-Fog-Cloud Continuum, Ant Colony Optimization.

Abstract: This paper addresses the workload placement problem in the edge-fog-cloud continuum. We model the edge-fog-cloud computing continuum as a multi-agent framework consisting of networked resource supply and demand agents. Inspired by the swarm intelligence behavior of the ant colony optimization, we propose a workload scheduler for the arriving demand agents to increase local resource utilization and reduce communication costs without relying on a centralized scheduler. Like the ants, the demand agents will release pheromones on the resource agent to indicate the available resources. The next arriving demand agent will most probably choose a neighbor, following the pheromone value and communication cost. The framework's performance is evaluated in terms of local resource utilization, dependency on fog and cloud, and communication cost. We compare these metrics for the ant-inspired algorithm with random and greedy algorithms. The simulation results reveal that the proposed algorithm inspired by swarm intelligence can increase resource utilization at the edge and reduce the dependency on higher layers, while also decreasing the communication cost for the task of resource allocation.

1 INTRODUCTION


Data management and computing are key research areas in the Internet of Things (IoT). The initial solution is based on cloud computing due to its scalability and efficiency. However, transmitting a large amount of data to the cloud servers suffers from unpredictable delays (Jiao et al., 2013). Storing privacy-critical data in the cloud server may also trigger security issues (Parikh et al., 2019). Hence, data processing closer to the user, on the edge, should be considered.


Edge computing facilitates data processing at the network's edge (IoT devices or local servers) (Satyanarayanan, 2017). This approach brings computation closer to the data source, allowing for faster response times and improved data security (Simić et al., 2021). Nevertheless, edge devices have limited computing and storage capabilities compared to cloud servers. To bridge this gap, fog computing acts as an intermediary layer to extend cloud services closer to the edge devices (Bonomi et al., 2012), materializing the


edge-fog-cloud continuum. When the request, e.g., a pod, the smallest and most basic deployable unit in Kubernetes (Kim et al., 2021), arrives in the continuum step by step randomly, they will be placed based on their specific requirements such as resource demand, latency, data security, etc. Hence, the workload scheduler should be designed carefully to minimize latency and maximize resource utilization.

Inspired by the collective behaviors in nature such as schools of fish and flocks of birds, swarm intelligence operates through decentralized, self-organizing systems where the agents follow local rules without centralized controllers leading to scalable, robust and adaptive solutions (Schranz et al., 2021). These characteristics make agent-based, swarm intelligence methods particularly well-suited for optimizing resource allocation and task distribution in the heterogeneous and dynamic edge-fog-cloud environment.

Exactly this approach presents the focus of our paper: We model the edge-fog-cloud computing continuum as an agent-based system with resource agents (edge, fog, cloud agents) and demand agents (pods). Resource agents are characterized by different parameters such as available resources, communication cost, etc. Demand agents decide the optimal resource

^a <https://orcid.org/0000-0001-7612-581X>

^b <https://orcid.org/0000-0002-4432-4504>

^c <https://orcid.org/0000-0002-0714-6569>

agent for processing based on these parameters. We propose a swarm intelligence workload scheduler motivated by the ant colony optimization (ACO) algorithm (Dorigo and Stützle, 2003). Pods are modeled as ants. They can lay down the pheromones at each resource agent with a designed update rule based on the available resources when pods leave the agent. We also define the probability of the pods moving to the neighbors based on the pheromone values and communication costs if pods are not served in the current resource agent. Simulation results illustrate that the ACO-based scheduler outperforms baseline schedulers by employing random and greedy strategies. Summarized, the main contributions of the paper are the following:

- We use agent-based modelling to map the resource and demand agents, presenting the complex interplay in the edge-fog-cloud continuum.
- We propose a swarm intelligence-based workload scheduler inspired by ACO. Specifically, we introduce the pheromone update to indicate the available resources, joined with the communication cost for a reallocation on each resource agent.
- We develop a new simulation framework to evaluate the ACO algorithm against random and greedy methods through different case studies. The results show that ACO algorithm outperforms the baseline algorithms in terms of higher local resource utilization, lower communication cost, and a reduced dependency on fog and cloud nodes.

The remainder of the paper proceeds as follows: Section 2 introduces the current state-of-the-art in the edge-fog-cloud continuum, swarm intelligence, and ACO. Section 3 presents the model of the edge-fog-cloud continuum. The workload scheduler based on ACO is proposed in Section 4. Section 5 provides the simulation case studies compared with two baseline algorithms. Section 6 concludes this work.

2 RELATED WORK

This section reviews the current state-of-the-art in the edge-fog-cloud continuum workload placement mechanism, general and with a special focus on swarm intelligence.

Workload allocation in the edge-fog-cloud continuum presents significant challenges due to the heterogeneous nature of resources and varying computational capabilities across different layers (Shi et al., 2016). In recent years, several methods worked on strategies for workload offloading in edge computing: For instance, federated learning was implemented to

tackle the offloading problem in vehicular edge computing in (Hasan et al., 2024). In another work, game theory was considered for the task offloading in edge computing (Chen et al.,). In (Tong et al., 2020), an adaptive task offloading and resource allocation algorithm is proposed for mobile edge computing environments, utilizing deep reinforcement learning to make offloading decisions and allocate computing resources efficiently. However, in the works mentioned above, the paradigms are considered isolated. To integrate them as a continuum seamlessly over all layers on the edge, fog and cloud, analytical and hierarchical availability models were provided for the edge-fog-cloud computing continuum in (Pereira et al., 2022). A workload scheduling based on an adaptive container managed by Kubernetes was proposed for the continuum in (Robles-Enciso and Skarmeta, 2024). It is noticeable that how to manage the edge-fog-cloud continuum more effectively is still a challenge for workload placement design (Gkonis et al., 2023).

This brings us to swarm intelligence, a promising approach that provides adaptive methods for optimizing workload allocation in the edge-fog-cloud continuum, leveraging self-organization to manage distributed resources efficiently (Schranz et al., 2021). In (Rajesh et al., 2020), a swarm intelligence algorithm was applied in IoT routing to extend network life and cut energy costs. The authors in (Saba et al., 2023) proposed a load-balancing method using swarm intelligence for data management to reduce the response time of the cloud servers. In another work, an innovative hybrid swarm intelligence algorithm was designed in (Attiya et al., 2022) for task scheduling in cloud computing to increase the rate of convergence. The most innovative work comes from (Ghasemi and Schranz, 2024), where the authors tackle the dynamic nature of the available resources and workloads from the bottom-up. The delay-sensitive and insensitive pod agents are collocated in the modelled multi-agent system to minimize slack resources while improving resource utilization for edge micro data centers.

To tackle the high complexity behind workload allocation in the continuum, we implement the Ant Colony Optimization (ACO) based on the swarm behaviors of real ant colonies (Dorigo et al., 2006). The authors in (Corominas et al., 2023) proposed an ACO-based method to deal with the network alignment issue. In (Wu et al., 2023), an adaptive ACO algorithm was designed for path planning in robotics. In another work, the resource allocation problem was considered in (Du et al., 2023) for avionics systems. They provided an improved ACO strategy to minimize the energy cost during the allocation task. Moreover, the

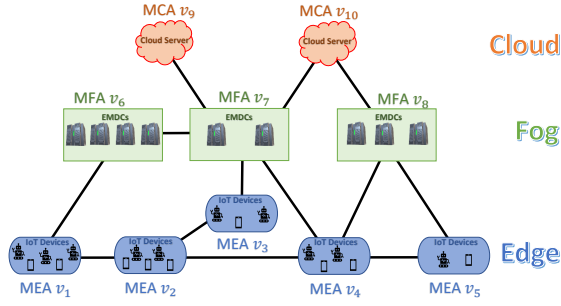


Figure 1: Exemplary architecture of the considered Edge-Fog-Cloud Continuum.

authors in (Chandrashekar et al., 2023) proposed a hybrid weight ACO method for task scheduling in cloud computing. This novel scheduler was able to enhance the performance compared with some traditional algorithms. Summarized, ACO is a powerful tool for optimization problems in network routing and scheduling (Bai et al., 2024). Thus, the ACO seems to be a candidate algorithm for the flexible workload placement we require in the highly dynamic edge-fog-cloud continuum.

3 SYSTEM MODEL AND MAIN OBJECTIVES

In this Section, we model the edge-fog-cloud continuum as a multi-agent swarm system and describe the features of the swarm agents. Three metrics are provided to evaluate the performance of the scheduler.

3.1 System Model of the Continuum

We introduce two types of swarm agents in the continuum: resource agents and demand agents. The resource agents are located in each computing node with available CPU and Memory (MEM) resources. They are embodied by a Multi-layer 360° dynamIc RunTime Orchestration (MIRTO) engine (related to the EU-Horizon project MYRTUS, see ACKNOWLEDGEMENT), so we call them MIRTO agents (Palumbo et al., 2024). Thus, the MIRTO agents in the edge/fog/cloud computing paradigm are the MIRTO edge/fog/cloud agents (MEA/MFA/MCA) in this paper. The type and capacity of the resources depend on their layer. Each MEA includes some IoT devices such as robots, smartphones, etc. MFA is comprised of a set of edge micro data centers (EMDC), and MCA is run by big servers with unlimited resource capacities. The main principle of the considered edge-fog-cloud continuum is

sketched in Figure 1. The MEA/MFA/MCA are denoted by blue/green/orange blocks. The solid black lines represent the connections in the continuum. The demand agents are represented by pods in the Kubernetes context. Each pod has a specific demand for resources and execution steps. So they are divided into small, medium, and large pods. Pods arrive in the continuum with specific steps generated by an exponential random variable with a parameter $\mu \in (0, 1]$, which reveals the frequency of arriving pods.

The different resource agents are also connected to ensure that pods can move freely through the links. This enables all resource agents to jointly form a seamless edge-fog-cloud continuum. According to the concepts in graph theory (Godsil and Royle, 2013), we can define the interaction topology of the resource agents in the continuum as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. $\mathcal{V} = \mathcal{V}_e \cup \mathcal{V}_f \cup \mathcal{V}_c = \{v_1, \dots, v_N\}$ is the vertex set of each resource swarm agent, where $\mathcal{V}_e = \{v_1, \dots, v_{N_e}\}$, $\mathcal{V}_f = \{v_{N_e+1}, \dots, v_{N_e+N_f}\}$, and $\mathcal{V}_c = \{v_{N_e+N_f+1}, \dots, v_N\}$ represent the vertex set of the MEA, MFA, and MCA, respectively. N_e , N_f , and N_c denote the number of the MEA, MFA, and MCA. $N = N_e + N_f + N_c$ stands for the number of all the resource swarm agents. The set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ reveals the connection of the continuum. $\forall v_i, v_j \in \mathcal{V}$, $(v_i, v_j) \in \mathcal{E}$ means that there are connection between the resource agent i and j , and pods can move between the i^{th} and j^{th} resource agent, otherwise not. Since the connections in the continuum can be inter-layer or intra-layer, the edge set \mathcal{E} can be rewritten as

$$\mathcal{E} = \mathcal{E}_{ee} \cup \mathcal{E}_{ef} \cup \mathcal{E}_{ff} \cup \mathcal{E}_{fc} \quad (1)$$

where \mathcal{E}_{ee} and \mathcal{E}_{ff} stand for the set of the connections in edge and fog computing paradigm. \mathcal{E}_{ef} (\mathcal{E}_{fc}) is the set of connections between the edge layer and fog layer (fog layer and cloud layer). It can be observed from (1) that there are no connections between the cloud servers because they manage their resources in centralized manner. There is also no direct connections between the edge to the cloud layer. Then we can define the neighbor set of the resource agent v_i as

$$\mathcal{N}_i = \{v_j \in \mathcal{V} : (v_i, v_j) \in \mathcal{E}\}. \quad (2)$$

The available resources in v_i are denoted as $R_{v_i} = (R_{v_i}^{CPU}, R_{v_i}^{MEM})$, where $R_{v_i}^{CPU}$ and $R_{v_i}^{MEM}$ represent the available CPU and MEM resources. The demand resource for the k^{th} pod is denoted by $D_k = (D_k^{CPU}, D_k^{MEM})$. When the k^{th} pod arrives at the resource agent v_i , this MIRTO agent will provide information about the resource available to the pod. If

$$D_k^{CPU} \leq R_{v_i}^{CPU} \text{ and } D_k^{MEM} \leq R_{v_i}^{MEM} \quad (3)$$

there are enough resources that can be assigned and pods can finish execution here. Otherwise, the pod

can receive the available resource information from the neighbor set \mathcal{N}_i , chooses one, and moves to it. Each pod constantly rechecks the available resources until it finds the resource agent with adequate resources. Once pod k processes in the MIRTO agent v_i , the available resources in v_i will be reduced to $R_{v_i} - D_k$ until pod finishes execution. After processing, the resources are released again.

3.2 Main Objectives

This section focuses on developing the workload scheduler for the edge-fog-cloud continuum. The main objective for each pod is to find the resource agent to meet its demand with the least latency. To evaluate the performance of the workload scheduler, we introduce three metrics as key performance indicators (KPI): resource utilization (RU), dependency on fog/cloud (DF/DC) and communication cost (CC). The RU of CPU/MEM for v_i is defined as

$$RU_{v_i}^{CPU} = \frac{\hat{R}_{v_i}^{CPU} - R_{v_i}^{CPU}}{\hat{R}_{v_i}^{CPU}}, \quad RU_{v_i}^{MEM} = \frac{\hat{R}_{v_i}^{MEM} - R_{v_i}^{MEM}}{\hat{R}_{v_i}^{MEM}} \quad (4)$$

where $\hat{R}_{v_i}^{CPU}$ and $\hat{R}_{v_i}^{MEM}$ are the capability of CPU and MEM resources of v_i . $\hat{R}_{v_i}^{CPU} - R_{v_i}^{CPU}$ and $\hat{R}_{v_i}^{MEM} - R_{v_i}^{MEM}$ are the CPU and MEM resource used by v_i . These resources are assigned to pods for execution and will be released when pods leave the continuum.

Assume that there are Q pods arriving in the continuum. For the k^{th} arriving pod, denote the visited resource agent set as $M_k = \{v_{k_1}, \dots, v_{k_m}\} \subseteq \mathcal{V}$. Let I_k^f and I_k^c be the indicators to reflect whether the k^{th} pod visited MFA and MCA. They are shown as follows

$$I_k^f = \begin{cases} 1, & \text{if } M_k \cap \mathcal{V}_f \neq \emptyset \\ 0, & \text{if } M_k \cap \mathcal{V}_f = \emptyset \end{cases} \quad (5)$$

and

$$I_k^c = \begin{cases} 1, & \text{if } M_k \cap \mathcal{V}_c \neq \emptyset \\ 0, & \text{if } M_k \cap \mathcal{V}_c = \emptyset. \end{cases} \quad (6)$$

Hence, DF and DC are defined as

$$DF = \frac{\sum_{k=1}^Q I_k^f}{Q}, \quad DC = \frac{\sum_{k=1}^Q I_k^c}{Q}. \quad (7)$$

The path of the k^{th} pod is defined as $P_k = \{(v_{k_1}, v_{k_2}), \dots, (v_{k_{m-1}}, v_{k_m})\} \subseteq \mathcal{E}$. Let c_{ij} be the CC to transmit pod from resource agent v_i to resource agent v_j , thus, the average CC for Q pods that arrive in the continuum is defined as

$$CC_{avg} = \frac{\sum_{k=1}^Q \sum_{(v_i, v_j) \in P_k} c_{ij}}{Q} \quad (8)$$

where $\sum_{(v_i, v_j) \in P_k} c_{ij}$ denotes the CC of the k^{th} pod (CC_k). In summary, the ultimate goal of this work is to design a workload scheduler with high local RU, low DF/DC, and low CC.

4 METHODOLOGY

In this Section, we will introduce the swarm intelligence workload placement inspired by the ACO algorithm, where we map the pods to the role of ants that can release pheromones on each resource agent. Therefore, we define the pheromone value on the resource agent v_i as τ_i . The initial pheromone value on each resource agent depends on the layer they are located at. We design the update scheme τ_i on the MIRTO agent v_i as

$$\tau_i := \tau_i + \gamma \Delta \tau_i^k \quad (9)$$

if the k^{th} pod finished execution on v_i . $\Delta \tau_i^k$ is the released pheromone value by pod k on v_i . This value is related to the demand of the k^{th} pod. γ is a parameter which represents the pheromone deposit factor.

In contrast, if the k^{th} pod arrives at v_i , but cannot be served here, we introduce the evaporation for the pheromone value on v_i . This can be regarded as the reduction of the pheromone value if the pods are rejected to be served. The updated rule is designed as

$$\tau_i := (1 - \rho^k) \tau_i \quad (10)$$

where ρ^k denotes the evaporation rate which is also related to the demand of the k^{th} pod. Then, the pod should decide where to go next from the neighbor set \mathcal{N}_i . It can be observed from (9) and (10) that the pheromone values reveal the available resources on each MIRTO agent. When it comes to how to select the neighbors, we first implement two baseline algorithms (random and greedy). After that, we design a swarm intelligence algorithm for the selection based on the ACO method to make a decision for the next MIRTO agent from the set \mathcal{N}_i . The algorithms are presented as follows.

- **Random.** The pods randomly choose a resource agent from the neighbor set \mathcal{N}_i and check the available resource again until they find the resource agent meeting their demand. Although the random algorithm exhibits excellent scalability, often requires less computational resources and can be easier to implement, it is hard to find the appropriate resource agent at an appropriate time in many cases.
- **Greedy.** The pods move to a resource agent with the maximum pheromone value from the neighbor set \mathcal{N}_i (if there are multiple MIRTO agents

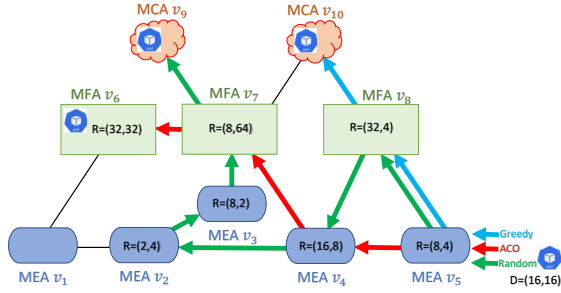


Figure 2: Three possible paths of a pod to find a suitable resource agent in the continuum under random (green), greedy (blue), and ACO (red) algorithms.

with the same maximum pheromone value, one is randomly chosen). However, privacy issues may not be guaranteed since the pods are inclined to go to the higher layer (edge to fog, or fog to cloud) under the greedy algorithm. Additionally, the inter-layer transmission cost is much higher than the intra-layer, which is not suitable for latency-sensitive tasks.

- **ACO.** As the transmission cost is also an important factor for workload management, it should be taken into consideration when pods make their decision. Motivated by the ACO method, the pods move to a resource agent $v_j \in \mathcal{N}_i$ with the probability p_j defined as follows

$$p_j = \frac{\tau_j^\alpha \eta_{ij}^\beta}{\sum_{v_j \in \mathcal{N}_i} \tau_j^\alpha \eta_{ij}^\beta} \quad (11)$$

where $\eta_{ij} = 1/c_{ij}$ denotes the heuristic desirability which is the inverse of the transmission cost between v_i and v_j . α and β are two weighting parameters, one for the pheromone value, the other for the transmission cost. It can be observed from (11) that pods are favorable to move to the neighbor with a large number of available resources and low communication costs. Hence, more pods can be served locally on the edge layer instead of moving to the fog or cloud layer. Algorithm 1 summarizes the behavior of pods by implementing the ACO-based workload scheduling.

Figure 2 illustrates three possible paths of a pod n with an exemplary demand of $D_n = (D_n^{CPU}, D_n^{MEM}) = (16, 16)$ to find the resource agent. The green path is generated under the random algorithm. Since the neighbor is selected randomly, the pod visits six MIRTO agents without enough available resources. Finally, it is transmitted to the cloud server. The blue path stands for the greedy algorithm, the pod moves directly to the MFA v_8 and then goes to the cloud server. The red one represents the most likely path

Algorithm 1: ACO Based Workload Placement.

Input: Pods enter the continuum

Output: The path to the best resource agent

- 1 Initialization phrase;
- 2 **for** $k \leftarrow 1$ **to** Q **do**
- 3 Start randomly from a MEA v_i ;
- 4 Check the available resources ;
- 5 **while** Resource condition (3) is not satisfied **do**
- 6 Update the pheromone value by (10) ;
- 7 Choose a neighbor based on (11);
- 8 Move to the selected resource agent;
- 9 Recheck the available resources.
- 10 Finishing execution;
- 11 Update the pheromone value by (9);
- 12 Leave the continuum.

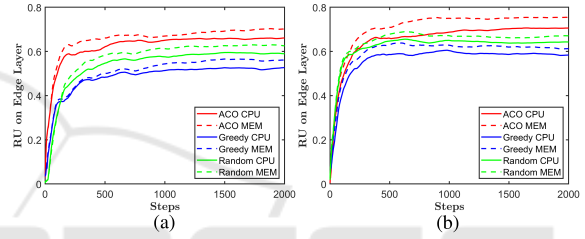


Figure 3: Resource utilization (RU) on edge layer of the three algorithms with $\mu = 0.5$ (a) and $\mu = 0.8$ (b).

of the pod under ACO algorithm. It can be served on fog layer without visiting too many MIRTO agents.

5 SIMULATION CASE STUDIES

In this Section, the edge-fog-cloud continuum is simulated using the MESA library (Masad et al., 2015) in a Python environment. We first set up the parameters in the continuum. After that, the analysis and discussion of the simulation results are presented to compare the ACO algorithm with two baseline algorithms in three aspects (RU, DF/DC, and CC), as mentioned in Section 3.2.

5.1 Setup of the Continuum

In this simulation framework, we exemplarily consider ten MEA, five MFA, and two MCA in the continuum. The capacity is defined as 16 or 32 CPU/MEM resource units for each MEA. MFA is comprised of a set of EMDC with a capacity equal to 64 or 128 CPU/MEM resource units, and the capacity MCA is assumed unlimited. Their connections are generated randomly by NetworkX (Platt, 2019) to ensure that

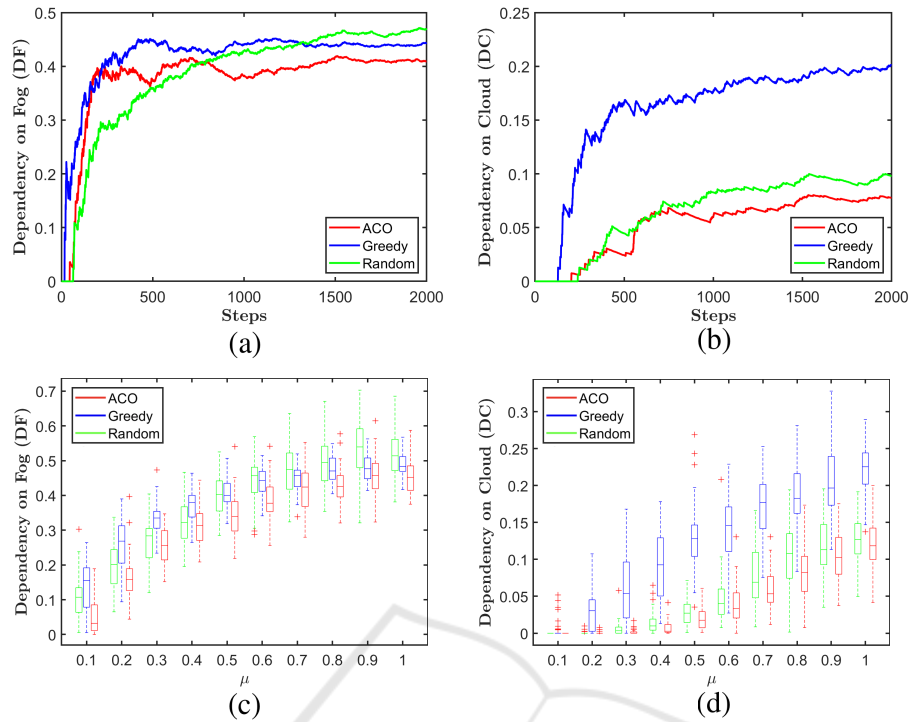


Figure 4: Dependency on fog (a) and cloud (b) (DF/DC) of the three algorithms with $\mu = 0.8$, and box-plot of DF (c) and DC (d) with different μ under three algorithms in 50 simulations.

the continuum is connected according to our model described in Section 3. The communication cost c_{ij} is defined as follows

$$c_{ij} = \begin{cases} 1, & \text{if } i, j \in \mathcal{V}_e \\ 2, & \text{if } i, j \in \mathcal{V}_f \\ 5, & \text{if } i \in \mathcal{V}_e, j \in \mathcal{V}_f \\ 30, & \text{if } i \in \mathcal{V}_f, j \in \mathcal{V}_c \end{cases} \quad (12)$$

The initial pheromone values τ_i for the MEA are defined as $\tau_i = 4$, if the capacity is (16,16), $\tau_i = 6$ if the capacity is (32,32), and $\tau_i = 5$ otherwise. Similarly, for the MFA, $\tau_i = 12$, if the capacity is (64,64), $\tau_i = 18$ if the capacity is (128,128), and $\tau_i = 15$ otherwise. The initial pheromone values on MCA equal to 50 and will not be updated during the process.

The demand agents are represented by pods in the Kubernetes context. Small (Medium) pods require 1 or 2 (4 or 6) CPU/MEM resource units with the execution steps distributed from 20 to 30 (50 to 130) uniformly and randomly. Their frequency of arrival is 0.4. For large pods, they need 8 or 16 CPU/MEM resource units with the execution steps distributed from 200 to 400 and the frequency of arrival is 0.2. Table 1 lists all the features of the arriving pods.

The parameters in the algorithms are defined based on following criteria: As γ indicates the probability of considering pheromone values for a resource

Table 1: Pod profiles.

	CPU/MEM Demand	Execution Steps
Small	1 or 2 / 1 or 2	[20,30]
Medium	4 or 6 / 4 or 6	[50,130]
Large	8 or 16 / 8 or 16	[200,400]

allocation decision, we set $\gamma = 1$ on edge layer, $\gamma = 0.6$ on fog layer, and $\gamma = 0$ on cloud servers. As we want to achieve a low dependency on the cloud, the values are chosen in descending order. The released pheromone $\Delta\tau^k$ and ρ^k are designed related to the demand of the pods. We set $\Delta\tau^k$ as 1, 2, and 3 for small, medium, and large pods. ρ^k is set as 0.15, 0.1, and 0.05 for small, medium, and large pods. The weight parameters are designed as $\alpha = 1.4$ and $\beta = 1$. We set the iteration steps to 2,000 for each simulation.

5.2 Results and Discussion

Figure 3 demonstrates the resource utilization (RU) on the edge layer first, as lightly loaded edge-fog-cloud continuum with $\mu = 0.5$ and second, a heavily loaded one using $\mu = 0.8$. It can be obtained that RU on the edge layer under the ACO algorithm is increased compared with the other two baseline algorithms. More pods can be served locally instead of moving to higher layers under ACO algorithm for

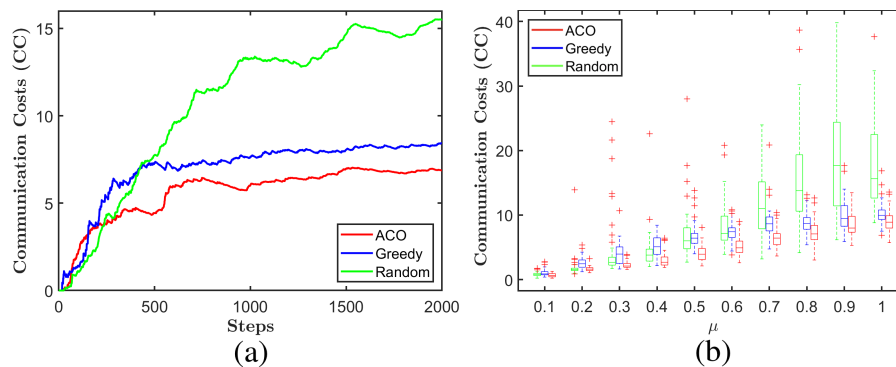


Figure 5: Communication costs (CC) of the three algorithms with $\mu = 0.8$ (a), and box-plot of CC with different μ in 50 simulations (b).

both lightly and heavily loaded continuums.

Figure 4 (a) and (b) illustrate the dependency on fog and cloud layer (DF/DC) with $\mu = 0.8$. We can observe that DF/DC under the ACO algorithm is lower than another algorithm after 2,000 steps if the load is heavy. Moreover, we explore the DF/DC with different μ . The results are shown by box-plot in Figure 4 (c) and (d) by running 50 simulations. It is noticeable that the greedy algorithm relies much more on the cloud (as it has much more resources available and the algorithm goes always for the highest availability). The DF/DC under the ACO algorithm keeps the lowest for the different loaded continuums.

Figure 5 (a) reflects the communication costs (CC) with $\mu = 0.8$. The baseline algorithms are still higher than the ACO algorithm as the iteration steps increase. Moreover, the CC with different μ is presented in Figure 5 (b) in 50 simulations. ACO algorithms can also save the CC compared with the baseline algorithms.

To sum up, compared with the two baseline algorithms, the ACO algorithm increases the local RU and decreases the DF/DC. This is because the scheduler based on the ACO algorithm considers both available resources and transmission costs. The pods are inclined to go to the matched MIRTO agent without too many costs. Implementing the ACO algorithm also reduces the CC. In conclusion, the ACO algorithm demonstrates improved performance in optimizing local resource utilization and minimizing data transfer and computational costs across the edge-fog-cloud continuum. This approach not only enhances the efficiency of workload distribution but also addresses critical concerns related to latency and privacy, making it particularly suitable for applications with stringent requirements in these areas, where computing on the edge is favored over cloud computing.

6 CONCLUSION

This paper investigates the workload placement problem for the edge-fog-cloud continuum. A multi-agent framework is established to model the continuum with arriving pods as demand agents and continuum nodes as resource agents. We design a swarm-based workload scheduler to place each demand agent at a proper resource agent considering the available resources and communication cost. The proposed algorithm is inspired by ACO, where pods mimic the behavior of ants by laying down pheromones on each resource agent to reveal the available resources. The probability of the pods moving to neighbors is also defined based on the pheromone value and communication costs. Compared with random and greedy algorithms, numerical studies illustrate that the workload scheduler based on swarm intelligence shows higher local resource utilization with lower dependency on fog and cloud. Future works will try to modify the algorithm to consider more factors, such as privacy issues. Moreover, other approaches, such as evolutionary algorithms and approaches from reinforcement learning, will be explored to design adaptable rules for distributed resource allocation in dynamic edge-fog-cloud continuum.

ACKNOWLEDGEMENT

Funded by the European Union, project MYRTUS, by grant No. 101135183. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

REFERENCES

- Attiya, I., Abd Elaziz, M., Abualigah, L., Nguyen, T. N., and Abd El-Latif, A. A. (2022). An improved hybrid swarm intelligence for scheduling iot application tasks in the cloud. *IEEE Transactions on Industrial Informatics*, 18(9):6264–6272.
- Bai, X., Liu, D., and Xu, X. (2024). A review of improved methods for ant colony optimization in path planning. *Journal of Ship Research*, pages 1–16.
- Bonomi, F., Milito, R., Zhu, J., and Addepalli, S. (2012). Fog computing and its role in the internet of things. In *Proceedings of the 1st Edition of the MCC workshop on Mobile Cloud Computing*, pages 13–16.
- Chandrashekar, C., Krishnadoss, P., Kedalu Poornachary, V., Ananthkrishnan, B., and Rangasamy, K. (2023). Hwacoa scheduler: Hybrid weighted ant colony optimization algorithm for task scheduling in cloud computing. *Applied Sciences*, 13(6):3433.
- Chen, Y., Zhao, J., Hu, J., Wan, S., and Huang, J. Distributed task offloading and resource purchasing in noma-enabled mobile edge computing: Hierarchical game theoretical approaches. *ACM Transactions on Embedded Computing Systems*, 23(1):1–28.
- Corominas, G. R., Blesa, M. J., and Blum, C. (2023). Antnetalign: Ant colony optimization for network alignment. *Applied Soft Computing*, 132:109832.
- Dorigo, M., Birattari, M., and Stutzle, T. (2006). Ant colony optimization. *IEEE Computational Intelligence Magazine*, 1(4):28–39.
- Dorigo, M. and Stützle, T. (2003). The ant colony optimization metaheuristic: Algorithms, applications, and advances. *Handbook of Metaheuristics*, pages 250–285.
- Du, X., Du, C., Chen, J., and Liu, Y. (2023). An energy-aware resource allocation method for avionics systems based on improved ant colony optimization algorithm. *Computers and Electrical Engineering*, 105:108515.
- Ghasemi, A. and Schranz, M. (2024). Bottom-up resource orchestration in edge computing: An agent-based modeling approach. In *12th IEEE International Conference on Intelligent Systems*, pages 1–7.
- Gkonis, P., Giannopoulos, A., Trakadas, P., Masip-Bruin, X., and D’Andria, F. (2023). A survey on iot-edge-cloud continuum systems: status, challenges, use cases, and open issues. *Future Internet*, 15(12):383.
- Godsil, C. and Royle, G. F. (2013). *Algebraic graph theory*, volume 207. Springer Science & Business Media.
- Hasan, M. K., Jahan, N., Nazri, M. Z. A., Islam, S., Khan, M. A., Alzahrani, A. I., Alalwan, N., and Nam, Y. (2024). Federated learning for computational offloading and resource management of vehicular edge computing in 6g-v2x network. *IEEE Transactions on Consumer Electronics*, 70(1):3827–3847.
- Jiao, L., Friedman, R., Fu, X., Secci, S., Smoreda, Z., and Tschofenig, H. (2013). Cloud-based computation offloading for mobile devices: State of the art, challenges and opportunities. *2013 Future Network & Mobile Summit*, pages 1–11.
- Kim, E., Lee, K., and Yoo, C. (2021). On the resource management of kubernetes. In *2021 International Conference on Information Networking*, pages 154–158.
- Masad, D., Kazil, J. L., et al. (2015). Mesa: An agent-based modeling framework. In *SciPy*, pages 51–58. Citeseer.
- Palumbo, F., Zedda, M. K., Fanni, T., Bagnato, A., Castello, L., Castrillon, J., Ponte, R. D., Deng, Y., Driessen, B., Fadda, M., et al. (2024). Myrtus: Multi-layer 360 dynamic orchestration and interoperable design environment for compute-continuum systems. In *Proceedings of the 21st ACM International Conference on Computing Frontiers: Workshops and Special Sessions*, pages 101–106.
- Parikh, S., Dave, D., Patel, R., and Doshi, N. (2019). Security and privacy issues in cloud, fog and edge computing. *Procedia Computer Science*, 160:734–739.
- Pereira, P., Melo, C., Araujo, J., Dantas, J., Santos, V., and Maciel, P. (2022). Availability model for edge-fog-cloud continuum: an evaluation of an end-to-end infrastructure of intelligent traffic management service. *The Journal of Supercomputing*, pages 1–28.
- Platt, E. L. (2019). *Network science with Python and NetworkX quick start guide: explore and visualize network data effectively*. Packt Publishing Ltd.
- Rajesh, G., Mercilin Raajini, X., Ashoka Rajan, R., Gokuldhev, M., and Swetha, C. (2020). A multi-objective routing optimization using swarm intelligence in iot networks. In *In Proceedings of the Intelligent Computing and Innovation on Data Science Conference*, pages 603–613. Springer.
- Robles-Enciso, A. and Skarmeta, A. F. (2024). Adapting containerized workloads for the continuum computing. *IEEE Access*, 12:104102–104114.
- Saba, T., Rehman, A., Haseeb, K., Alam, T., and Jeon, G. (2023). Cloud-edge load balancing distributed protocol for ioe services using swarm intelligence. *Cluster Computing*, 26(5):2921–2931.
- Satyanarayanan, M. (2017). The emergence of edge computing. *Computer*, 50(1):30–39.
- Schranz, M., Di Caro, G. A., Schmickl, T., Elmenreich, W., Arvin, F., Şekercioğlu, A., and Sende, M. (2021). Swarm intelligence and cyber-physical systems: concepts, challenges and future trends. *Swarm and Evolutionary Computation*, 60:100762.
- Shi, W., Cao, J., Zhang, Q., Li, Y., and Xu, L. (2016). Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646.
- Simić, M., Prokić, I., Dedeić, J., Sladić, G., and Milosavljević, B. (2021). Towards edge computing as a service: Dynamic formation of the micro data-centers. *IEEE Access*, 9:114468–114484.
- Tong, Z., Deng, X., Ye, F., Basodi, S., Xiao, X., and Pan, Y. (2020). Adaptive computation offloading and resource allocation strategy in a mobile edge computing environment. *Information Sciences*, 537:116–131.
- Wu, L., Huang, X., Cui, J., Liu, C., and Xiao, W. (2023). Modified adaptive ant colony optimization algorithm and its application for solving path planning of mobile robot. *Expert Systems with Applications*, 215:119410.