Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

IoT Service Placement in Dynamic Fog Environments Using Variable Action-Set Learning Automata

babak anari

babak.anari@iau.ac.ir

Islamic Azad University, Shabestar 53815-159, Iran

zohreh anari

Payame Noor University (PNU), Tehran, Iran

Research Article

Keywords: Fog Computing, IoT, Service Deployment, Learning Automata, Variable Action-Set Learning Automata

Posted Date: May 4th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-9251600/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

IoT Service Placement in Dynamic Fog Environments Using Variable Action-Set Learning Automata

Babak Anari[a, *], Zohreh Anari[b, *]

a) Department of Computer Engineering, Shab.C., Islamic Azad University, Shabestar 53815-159, Iran. babak.anari@iau.ac.ir
b) Department of Computer Engineering and Information Technology, Payame Noor University (PNU), Tehran, Iran. zanari323@pnu.ac.ir

[*] ***Corresponding Authors:***

Babak Anari, Zohreh Anari

Abstract

Nowadays, many time-sensitive applications, such as Internet of Things (IoT) devices, mission-critical applications, and autonomous driving, require low-latency, high-performance computing close to end users. Fog computing addresses this need by deploying decentralized fog nodes near IoT devices to deliver application services efficiently. However, selecting suitable fog nodes for deploying and executing IoT services is a challenging task and is known as the service placement problem (SPP). Most existing fog SPP algorithms rely on static assumptions, under which the number of available fog nodes, the links between them, and service requirements do not change. This issue leads to high computational costs and reduced effectiveness in real-world applications. To overcome these challenges, we propose VASLA-FSP (Variable Action-Set Learning Automata for Fog Service Placement), a method based on variable action-set learning automata. In VASLA-FSP, the fog topology is modeled as a graph, and by mapping it to variable-action set learning automata, the IoT service placement problem is effectively addressed. We evaluated VASLA-FSP through extensive simulations using iFogSim under various scenarios. Simulation results demonstrate that VASLA-FSP outperforms existing methods across various evaluation criteria, achieving an average energy reduction of 17% in energy consumption, up to an 18% reduction in application delay, and approximately 20% improvement in the overall cost function.

Keywords: Fog Computing, IoT, Service Deployment, Learning Automata, Variable Action-Set Learning Automata.

1- Introduction

One of the popular technology terms of recent times is the Internet of Things (IoT). It is a self-organizing network that enables the connection of real-world objects to access the internet. The Internet of Things (IoT) enables one device to send information to other devices or transfer information through a communication network without requiring human interaction or direct computer-to-computer interaction [1]. Due to the increasing development of IoT devices for all intelligent and automated applications in recent years, these devices generate

a large amount of data, called big data [2]. Cisco estimates that by 2030, the number of IoT devices will exceed 80 billion units, generating approximately 850 zettabytes of data by that time [3]. Since IoT devices have limited storage space and computing resources, a standard solution is to use cloud computing to process this data remotely. In cloud infrastructure, all services are centrally located in the data center and geographically distant from the data source. Therefore, they consume a significant amount of bandwidth to transfer all the data, resulting in considerable latency between IoT devices and between the IoT devices and users [4]. Hence, fog computing is introduced as a distributed computing technology to overcome some of the constraints of cloud data centers and serve the requirements of IoT applications [5-8].

Fog or edge computing has been introduced as a vital paradigm in IoT architectures, providing cloud services closer to IoT devices and end users, thereby reducing communication delays and enhancing system efficiency [9, 10]. Furthermore, they perform some additional tasks for IoT devices, such as: (i) location awareness, (ii) context-aware decision-making, (iii) decreasing cloud network traffic, (iv) user mobility support, (v) real-time interactions, (vi) reducing latency and response time, (c) scalability, and (d) interoperability, compared with cloud computing [11, 12]. A fog device, or fog node, is any computational and storage resource within a fog computing context. The fog nodes are geographically distributed and serve as a middle layer that connects end-user devices to cloud services [13, 14]. Because fog environments are heterogeneous and dynamic, and nodes in the system may have limited energy and other resources, distributing applications (i.e., services) optimally in fog or cloud resources is a challenging problem [15].

Recently, several studies have addressed SPP in fog computing environments. Researchers typically classify existing service placement algorithms into two categories. The first group primarily relies on meta-heuristic-based approaches. These algorithms require searching the entire fog space to properly place IoT applications on fog nodes, which increases the search space and computational time. The second group considers only partial search spaces. However, their problem is that they search all fog nodes regardless of whether they have sufficient capacity to host the service, or if some fog nodes have broken down, which increases the search space and response times.

Abofathi et al. [16] proposed a learning automata-based approach to allocate services on fog nodes. Their method helps reduce energy consumption and delays in IoT applications, providing a more efficient use of fog resources. Although the topology of their approach is a graph, it has the following limitations: (1) A fog node can only connect to nodes of the same or a higher level, and it is impossible to return to a previous level. When a service is placed on a high-level fog node, even if there is available space in a low-level node, the system cannot assign the service to the low-level node; (2) the topology is static and does not work for dynamic environments; and (3) their method searches all edges of fog nodes regardless of whether they have empty capacity to place fog nodes properly or are non-functional, which increases the search time and computational complexity.

Most existing approaches to the service placement problem use a hierarchical or graph-based structure to represent the fog computing environment. A limitation of these approaches is that they assume a predefined topology with a fixed number of nodes and edges. In contrast, the

proposed method is independent of the fog topology, allowing the number of nodes and edges (i.e., the topology structure) to change dynamically. This paper proposes a new service placement framework based on VASLA. The proposed framework includes four steps. In the first step, the fog topology is modeled as a graph consisting of fog nodes and their connecting edges. Based on this graph, in the second step, a Variable Action Set Learning Automaton (VASLA) is assigned to each fog node. Initially, each VASLA automaton contains a set of actions. The number of these actions for each VASLA automaton equals the number of fog nodes. Since the number of nodes and edges can change dynamically during execution, each automaton in VASLA selects only a subset of its actions (active actions) based on its neighbors. In the third step, we propose a mechanism to select the subset of active actions. The fourth step of the proposed method is service placement. In this step, each VASLA automaton selects one action from the active actions specified in the previous step. The selected action determines the new location for the service placement. The automaton then receives a reward or a penalty based on the delay and energy consumption calculated for placing the desired service in the new location. In this way, each automaton learns the optimal location for service placement. In the third step, the active actions list may be empty. For this purpose, a stack is used to store the visited locations, allowing the algorithm to return to earlier positions. The main contributions in this paper are summarized as follows:

1. We propose VASLA-FSP, an intelligent service placement framework based on Variable Action-Set Learning Automata, capable of making efficient placement decisions in dynamic fog environments.
2. The proposed approach dynamically adapts its action set according to changes in fog topology, enabling effective decision-making under varying numbers of nodes and communication links.
3. VASLA-FSP reduces the decision search space by selecting a subset of candidate fog nodes, leading to faster and more efficient service placement.
4. The proposed method is topology-independent and can operate effectively in both homogeneous and heterogeneous fog environments.
5. Extensive simulation results using iFogSim demonstrate that the proposed approach significantly improves system performance in terms of latency, energy consumption, network usage, and execution time compared to existing methods.

The remainder of this paper is structured as follows: Section 2 provides the related work on the service placement of IoT applications over a fog computing environment; Section 3 provides an overview of the preliminaries and main concepts; Section 4 describes the proposed service placement algorithm; Section 5 discusses the details of the experiments conducted to validate the performance of our approach; and Section 6 presents the conclusions and directions for future research.

2. Related work

Several earlier studies have addressed the problem of resource placement using optimization approaches (e.g., evolutionary algorithms, machine learning-based optimization, and mathematical modeling). The following section discusses several relevant papers on resource placement strategies in fog computing environments. At the end of this section, the algorithms compared and summarized will be listed in Table 1.

Dejima et al. [17] proposed an optimization strategy based on discrete particle swarm optimization (PSO) to minimize application latency and energy consumption in IoT service placement. Their method takes into account the logical structures of IoT applications and service latency needs to strike a balance between power consumption and delay.

Mehran et al. [18] proposed a genetic multi-objective optimization algorithm that uses the Pareto decision-making method to identify the optimal allocation of IoT applications in fog-based computing systems. The balancing process of their method is modeled as a finite-state machine (FSM) to optimize energy consumption, delay, and total cost. The findings show that, compared to other strategies, this strategy improves energy efficiency while minimizing energy consumption and economic costs.

Lera et al. [19] proposed a multi-criteria decision-making (MCDM) approach for fog service placement. Their method includes a decision-making model that ranks possible placements based on several criteria. Furthermore, they applied metrics including the number of hubs, latency, power, and cost to rank the alternatives. The researchers employed heterogeneous weights for each criterion to evaluate their method against a weighted-average approach. The results demonstrate its effectiveness in efficiently mapping IoT applications.

Natesha et al. [20] presented a two-layer approach to resource provisioning that optimizes IoT application deployment on fog nodes by applying an elitist genetic algorithm (EGA). They modeled the service placement problem in fog computing as an optimization task to minimize cost, energy, and delay.

Rajabzadeh et al. [21] introduced the multi-Fog Service Placement (MFP) algorithm for resource placement in IoT systems with a fog layer. Their research introduced a multi-zone fog-layer framework to exploit underutilized computing resources across various fog zones. This approach improved the efficiency of both time and energy consumption for solving the main problems in fog computing.

Maia et al. [22] developed a multi-objective genetic algorithm incorporating both random and heuristic techniques for service placement and load balancing in edge computing. To reduce SLA violations, operational costs, and service-availability deadlines, they formulated the service placement and load distribution problems as a mixed-integer nonlinear programming (MINLP) optimization problem.

Qian et al. [23] addressed the issue of service placement with respect to privacy in edge cloud systems by developing a hybrid algorithm, called PSP, for service placement. The method

leverages distributed federated learning to analyze and understand users' privacy preferences across different services. For optimal service placement, their method uses a greedy algorithm. According to simulation results, the algorithm meets service requests while protecting user privacy in mobile edge clouds.

Oma et al. [24] introduced a tree-structured fog model to distribute computing processes among data storage and fog nodes. In this framework, the root fog node defines the cloud servers, and the leaf nodes refer to the edge nodes connected to the devices. This architecture reduces total electric power consumption and execution time in the Internet of Things (IoT). This technique provides an enhanced solution for computational resource allocation and energy-efficient scheduling, thereby improving the proposed fog computing paradigms for IoT sharing networks.

Yousepour et al. [25] proposed a general architecture for IoT fog-cloud systems and a model to measure service delay. They presented a fog-device-aware delay-minimizing policy, and their results showed that it significantly reduces service delay in fog environments.

Mahmoud et al. [26] proposed an energy-aware allocation strategy to place application modules (tasks) on fog devices. This algorithm enhances energy efficiency at fog nodes by allocating application tasks using an improved round-robin (IRR) and dynamic voltage and frequency scaling (DVFS). They evaluated the proposed strategy using the iFogSim simulator and found it more energy-efficient than the default allocation and cloud-only policies.

Huang et al. [27] introduced a multi-objective approach to placing service replicas within a fog computing infrastructure, aiming to optimize both deployment cost and service delay. They proposed a method, named Pareto-based Ant Colony Optimization for Multiple Replicas (MRPACO), based on ant colony optimization to tackle this challenge. The experimental results reveal that their method achieved higher accuracy and diversity than the other methods.

Ghobaei-Arani [28] proposed a low-cost methodology to address a key problem in fog computing: the allocation of Internet of Things (IoT) services. They proposed an IoT service placement technique using the whale optimization algorithm (WOA) to achieve efficient deployment. They designed an autonomous deployment approach by adopting the MAPE-K control model and modeling it in a multi-objective optimization framework. The scheme continually minimizes delay, energy consumption, and throughput while satisfying the quality-of-service (QoS) specifications.

Roy et al. [29] proposed a multi-objective integer linear programming (MILP) approach integrated with a binary particle swarm optimization (BPSO) technique within the framework of artificial intelligence (AI) to solve combinatorial optimization problems. The proposed methodology improves solution accuracy, increases convergence rates, and improves efficiency compared with conventional optimization methods.

Sarrafzade et al. [30] introduced a GA-based algorithm for service placement. This algorithm enhances the service placement by also considering network latency and using the network more effectively. Their method serves as a good example of the growing role of AI-based optimization

in fog computing, highlighting the promise of genetic algorithms in addressing complex service placement issues.

In the study by Liu et al. [31], the Cuckoo Search Algorithm was applied to optimize the placement of IoT services with multiple objectives. The researchers implemented the proposed approach across several coordinated colonies and controlled them using the Cuckoo Fog Computing Model (CFCM), an architecture for managing fog cells. The MAPE-K (Monitoring, Analysis, Planning, Execution, and Knowledge Transformation) model is tailored for fog systems to manage a common knowledge base, enabling decision-making in fog infrastructures.

Dadashi Gavaber et al. [21] presented a novel approach, Multi-Fog Placement (MFP), for resource placement in Internet of Things systems within a fog environment. The authors built the proposed approach on multiple coordinated colonies and used the Cuckoo Fog Computing Model (CFCM) to govern these colonies and organize the fog cell. This structure allows greater flexibility in service placement. If a fog node within a zone becomes unavailable, the system can use another fog node in the same zone to maintain service continuity. The main advantage of this approach is that it leverages the computing resources of fog nodes across regions by allocating as many services as possible to them.

Salimian et al. [32] proposed an evolutionary algorithm for deploying Internet of Things (IoT) applications in fog computing environments, combining the decision-making mechanism of the MADE-k approach with the particle swarm optimization (PSO) algorithm. They introduced an autonomous scheme based on the MADE-k model in the decision-making phase, along with the PSO algorithm to solve this optimization problem. Their method minimizes latency and costs while maximizing the use of fog resources. This model is implemented in the middleware for fog-cloud control and governs all fog colonies. The model is based on a structured process that monitors, analyzes, and prioritizes fog nodes and IoT services, then provides an optimal placement plan.

Taghizadeh et al. [33] proposed a metaheuristic technique, adapted from the non-dominated sorting genetic algorithm II (NSGA-II), to place IoT application components in fog infrastructures effectively. They also proposed a data transfer scheduling algorithm that could be applied to other similar data transfers to ensure the effective operation of the fog-cloud architecture. They used NSGA-II to enhance load balancing, latency, and resource use in fog environments.

Tekiyehband et al. [34] introduced an efficient, adaptive service delivery mechanism based on the learning automata method to improve decision-making in a dynamic fog environment for service deployment and publication. Their method improves fog node deployment efficiency by dynamically selecting fog nodes for IoT application placement, thereby reducing delays and maximizing resource utilization.

Overall, the service placement problem (SPP) is NP-hard and therefore often requires metaheuristic strategies to obtain near-optimal solutions. Skarlat et al. [35] introduced a GA approach to optimize service placement and resource allocation in a fog environment. Their approach considers the heterogeneous nature of applications and resources from a QoS perspective to place services effectively. The authors present a cloud-fog control middleware and fog orchestration control nodes. The cloud-fog control middleware manages all the fog cells, while the control nodes in the fog orchestration handle numerous fog cells or other controllers within their

respective coverage areas. The experimental results show that their system effectively utilizes the available fog landscape resources and achieves lower runtime than the other methods.

Guerrero et al. [36] investigated the application of three nature-inspired optimization methods: the weighted sum genetic algorithm (WSGA), NSGA-II, a genetic algorithm based on non-dominated sorting, and the multi-objective evolutionary algorithm using decomposition techniques (MOEA/D) to solve the service deployment problem in fog computing environments. They presented three objective functions: minimizing network latency, reducing cloud resource utilization, and optimizing service deployment. The results demonstrated that the evolutionary algorithms used by Guerrero et al. optimized the objectives more effectively and achieved a more diverse solution space, exhibiting superior performance compared to other approaches.

Canali et al. [37] proposed a GA-based heuristic for service placement in fog computing frameworks. They also developed an optimization framework for allocating data streams among fog nodes. Their approach considers both the current load on fog nodes and the latency between sensors on those nodes to compute the overall system delay. The experimental results of a realistic smart city demonstrate that their approach outperforms other algorithms in solving the mapping problem.

Jian et al. [38] presented a virtual machine placement learning model using an improved LSTM. In this article, the OEMBA algorithm uses data to train an LSTM model, enabling dynamic virtual machine placement. In addition, they combined the local search technique (OEM) with the Bat Algorithm (BA) to avoid local minima and improve global convergence. This algorithm can effectively reduce energy consumption at edge servers and optimize the cost efficiency of extensive server deployment.

Ayoubi et al. [39] proposed an autonomous IoT service placement methodology in fog computing to optimize application quality of service (QoS) requirements while maximizing the use of fog resources. Every fog node included a component named MADE-k, which managed the colonies throughout the fog network. Each colony contained some fog cells, an orchestration control node to manage, monitor, and control active IoT services, as well as all fog cells within the fog colony. The orchestration control node stored its related information within a knowledge base and used it to prioritize IoT service execution during the evaluation phase. During the decision phase, the system utilized the SPEAII algorithm to identify suitable fog cells for hosting the required IoT service.

Zare et al. [40] proposed an autonomous methodology for IoT service placement in fog computing to optimize application quality of service (QoS) requirements while maximizing the use of fog resources. Every fog node included a component named MADE-k, which managed the colonies throughout the fog network. Each colony contained some fog cells, an orchestration control node to manage, monitor, and control active IoT services, and all fog cells within the colony. The orchestration control node stored its related information within a knowledge base and used it to prioritize IoT service execution during the evaluation phase. During the decision phase, the system used the SPEAII algorithm to identify suitable fog cells to host the required IoT service.

Isaac and Guerrero [41] presented a DRL-based multi-objective optimization approach for fog computing application placement that employs a graph neural network (GNN). Their method achieves a trade-off among conflicting objectives while preserving flexibility and generalizability across different scenarios. The simulation results indicate that their method can significantly reduce execution time and increase flexibility compared to traditional approaches.

In another study, Wu et al. [42] presented a metaheuristic-based approach, IPGA-SPP, for multi-objective optimization in the deployment of IoT services. The authors of this paper employed a multi-objective optimization approach to maintain Pareto-optimal solution sets that minimize latency, reduce energy consumption, and maximize resource utilization in fog computing.

Table1: Comparison of fog service placement approaches.

Reference	Used technique	Performance metrics	Disadvantages	Case study	Simulation tools
Ghobaei-Arani et al. [28]	Whale optimization algorithm (WOA-FSP) Metaheuristic approach	Energy consumption, service delay, resource usage, service acceptance ratio	Metaheuristic may require high computation; performance depends on parameter tuning	IoT application	iFogSim
Canali et al. [37]	Heuristic algorithm based on genetic programming	Network latency, Execution time, Fast convergence	Single-objective; may not consider energy consumption	Smart city application	Python
Skarlat et al. [35]	Heuristic algorithm(Firstfit algorithm+Genetic algorithm)	Response time, service delay and resource utilization	Single-objective; may not scale well with large IoT networks	IoT application	Java and the Spring application framework.
Milad Ranjbari et al. [43]	Learning automata-based algorithm	Resource utilization, energy consumption, power consumption	Single-objective; may not handle multiple QoS metrics simultaneously	Cloud	Cloud Sim
Natesha et al. [20]	Elitism-based genetic algorithm (EGA)	Service time, cost and resource utilization, multi-objective	Requires careful parameter tuning; container-based implementation overhead	IoT application	Docker,containers and python
Yadav et al. [44]	Genetic and particle swarm optimization algorithms (GA-PSO)	Makespan, energy consumption, multi-objective	Complex hybrid algorithm; higher computational complexity	IoT application	C ++
Roy et al. [29]	Binary particle swarm optimization (BPSO)	QoS, user satisfaction ratio, cost, multi-objective	May converge prematurely; requires fine-tuning of parameters	Smart city applications	CloudSim
Sarrafzade et al. [30]	GA	Delay, network usage, energy consumption, cost, single-objective	Does not handle multiple objectives simultaneously	fog-cloud	iFogSim
Jian et al. [38]	Order exchange migration and bat algorithm (OEMBA)	Energy consumption and placement latency	Single-objective; limited to edge environments	Edge environment	CloudSim

Liu et al. [31]	Cuckoo search algorithm	Resource utilization, cost, delay, response time and energy consumption, multi-objective	May require many iterations for convergence	Smart vehicle environment	MATLAB
Huang et al.[27]	Multi replicas pareto ant colony optimization (MRPACO)	Cost, service latency, multi-objective	Computationally intensive; complex parameter setting	Cloud-fog	iFogSim
Ayoubi et al. [39]	Strength pareto evolutionary algorithm II (SPEAII)	Energy consumption, network usage, multi-objective	Algorithm complexity; may require large population size	Fog environment	iFogSim
Djemai et al. [17]	PSO	Energy consumption, delay	May get stuck in local minima	IoT application	iFogSim
Mehran et al. [18]	Pareto-based optimization	Energy consumption, execution time, cost, multiple objectives	High computational complexity	Healthcare applications	JMetal solver
Gavaber et al. [45]	Multi fog placement algorithm (MFPA)	Delay, energy consumption, Bandwidth, Multi-objective	Complexity increases with number of fog nodes	Fog environment	iFogSim
Maia et al. [22]	GA	Response deadline, resource usage, scalability, availability, multi-objective	Requires computational resources; GA may converge slowly	IoT and Edge Computing environment	Python with the CPLEX solver
Khosroabadi et al. [46]	SCATTER algorithm	Response time, network usage, delay, energy consumption.	Limited to single-objective; may ignore other QoS factors	IoT and Fog Computing environment	iFogSim
Guerrero et al. [36]	WSGA, NSGA-II and MOEA/D	Network latency, resource usage, service spread, execution time	High computational overhead; complex setup	Barabasi-Albert network topology	Simulation matlab
Yousefpour et al. [25]	Markovian queueing	Service delay	Single-objective; limited real-world application	IoT-fog-cloud	analytical model
Mahmoud et al. [26]	Improved round robin (IRR) and dynamic voltage and frequency scaling algorithms (DVFS)	Latency, resource utilization and energy consumption	Single-objective; may require specific hardware support	Remote patient monitoring (RPM)	iFogSim
Salimian et al. [32]	PSO	Resource utilization, Qos, latency, cost	May require tuning of swarm size and iterations	Fog-cloud	MATLAB R2019a
Taghizadeh et al. [33]	NSGA-II	Latency, cost, data availability	Computationally intensive	Fog-cloud	iFogSim
Tekiyeband et al. [34]	A learning automata-based approach	Service delay, cost	Single-objective; limited scalability	Fog	iFogSim
Lera et al. [19]	Ranking method (a multi-criteria decision method)	Latency, hop count, energy consumption, cost	Single-objective; may not be optimal under dynamic conditions	Fog	YAFS
Rajabzadeh et al. [45]	Multi fog service placement	Latency, energy consumption	Single-objective; may ignore other QoS metrics	Fog-cloud	iFogSim
Adyson et al. [22]	Multi objective genetic algorithm based on BRKGA and NSGA-II	Response time, cost, availability	High computational cost	Fog-cloud	Simulations

Oma et al. [24]	Tree-based fog computing model	Delay time, energy consumption	Single-objective; limited adaptability	Fog-cloud	Simulations
Azimzadeh et al. [47]	Complex Networks features (FSPCN)	Optimize resource usage, response time, delay time, bandwidth, reliability	High complexity; may require simulation tools	Fog	YAFS
Abofathi et al. [16]	A learning automata-based approach	Delay time, energy consumption, network usage, response time	Single-objective; may not handle multiple metrics simultaneously	Fog	iFogSim

3. Preliminaries

This section presents an overview of the learning automata, variable action-set learning automata, and the service placement problem.

3.1 Learning automata

A learning automaton (LA) is a self-operating machine that can be used in a random environment [48, 49]. This model involves selecting an optimal action from a set of actions through repetitive interactions with a random environment. Each automaton chooses an action from its action set and applies it to the random environment. The random environment evaluates the desired action and generates a response (reinforcement signal) to the automaton. LA uses this reinforcement signal and a learning algorithm to decide whether the selected action is rewarded or penalized. This cycle of the learning process is repeated for each LA until it finds its optimal action.

LA is a valuable tool used in numerous applications, such as computer networks [50], image processing [51], deep reinforcement learning [52], fuzzy applications [53], social networks [54], optimization problem [55], graph theory[56], and load balancing [57]. Fig. 1 shows the interaction of the LA with its random environment.

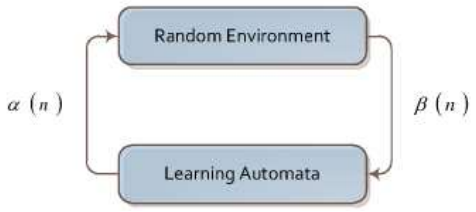


Fig. 1 The relationship of the LA with the random environment.

LA can be grouped into fixed-structure learning automata (FSLA) and variable-structure learning automata (VSLA). In FSLA, both the action probability vector and the state transition probabilities remain constant, while in VSLA these parameters vary over time. The random environment in LA is described using $< \alpha, \beta, c >$ where $\alpha = \{\alpha_1, \alpha_2, \dots, \alpha_r\}$ represents the finite set of actions, $\beta = \{\beta_1, \beta_2, \dots, \beta_r\}$ the environment's response (reinforcement signal), and $c = \{c_1, c_2, \dots, c_r\}$ is the set of the penalty probabilities, where each c_i corresponds to penalty probability of each action α_i. In the following, we explain the VSLA employed in this study.VSLA represented as $< \alpha, \beta, p, T >$. Here, $\alpha = \{\alpha_1, \alpha_2, \dots, \alpha_r\}$ denotes the action set, $\beta = \{\beta_1, \beta_2, \dots, \beta_r\}$ corresponds to the input set, $p \equiv \{p_1, p_2, \dots, p_r\}$ shows the probability distribution over actions such that each p_i represents the probability of choosing action α_i. T refers to the learning algorithm that updates the probability

vector of the action set of each automaton according to the response acquired from the environment. Additionally, r denotes the total number of actions. At time t, $\alpha_i(t) \in \alpha$ represents the chosen action, while $p(t)$ represents the corresponding action probability vector. The action obtains the reinforcement signal by deploying it into the environment. If the signal represents a favorable outcome, the action probability vector $p(t)$ is updated when the selected action is rewarded according to the linear learning algorithm as given in Eq. (8). Otherwise, the action chosen $\alpha_i(t)$is penalized using Eq. (9). In Equations (8) and (9), the parameters *a* (reward) and *b* (penalty) are used to determine how the action probabilities increase and decrease. When a = b, Eqs. 8 and 9 represent the linear reward-penalty L_{R-P} algorithm. When a $\gg$ b, they are known as the reward-epsilon-penalty $L_{R\varepsilon P}$ algorithm. When $b = 0$, the algorithm is called the linear reward-inaction L_{R-I}.

$$p_j(t+1) = \begin{cases} p_j(t) + a\left[1 - p_j(t)\right], & j = i \\ (1-a)p_j(t), & \forall j \neq i \end{cases} \quad (1)$$

$$p_j(t+1) = \begin{cases} (1-b)p_j(t), & j = i \\ \left(\frac{b}{r-1}\right) + (1-b)p_j(t), & \forall j \neq i \end{cases} \quad (2)$$

3.2 variable action-set learning automata(VASLA)

A variable action-set learning automaton is a class of learning automaton used when the actions available for the system to perform are not constant but may change from one situation to another [58]. While conventional learning automata have a fixed set of possible actions from which they can learn, VASLA allows the action set to change dynamically and adapts its behavior accordingly. By allowing the action set to vary dynamically, VASLA enables the automaton to better interpret and respond to environmental changes. A learning automaton with a constantly changing set of actions achieves expediency and ε-optimality when the reinforcement scheme is based on L_{R-I} [58]. A finite set of *n* actions, represented as $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$, defines this type of automaton. $A = \{A_1, A_2, \ldots, A_n\}$ representing the set of action subsets, where A(k) $\subseteq$ α denotes a subset of all possible actions that the automaton could select from at a specific time *k*. An external agency uses the probability distribution $q(k) = \{q_1(k), q_2(k), \ldots, q_m(k)\}$ to select a specific subset of actions randomly.

The model describes this distribution over a subset of possible actions, where $q_i(k) = prob[A(k) = A_i | A_i \epsilon A, 1 \leq i \leq 2^2 - 1]$. $\hat{p}_i(k) = prob[\alpha(k) = \alpha_i | A(k), \alpha_i \epsilon A(k)]$; indicates the likelihood of selecting action α_i, assuming that $\alpha_i \in A(k)$ and that the action subset A(*k*) has already been selected. Equation 3 defines the scaled probability $\hat{p}_i(k)$ as:

$$\hat{p}_i(k) = \frac{p_i(k)}{K(k)} \quad (3)$$

$K(k) = \sum_{\alpha_i \epsilon A(k)} = P_i(k)$ represents the total probability of the actions in the subset A(*k*), and $p_i(k) = prob[\alpha(k) = \alpha_i]$ denotes the probability of choosing action α_i when A(*k*) is selected. The following describes how a variable action-set learning automaton chooses an action to update the probability. Before choosing an action, Equation (3) adjusts the probabilities of every action

in the selected subset. After that, the automaton randomly chooses one of its possible actions based on the normalized action probability vector $\hat{p}(k)$. It adapts this probability vector in response to input from the environment, and then rescales the probability vector of the actions in the selected subset, which is defined as follows :

$$p_i(k+1) = \hat{p}_i(k+1).K(k) \tag{4}$$

For each α_i belonging to $A(k)$, the approach described above has been shown to achieve both absolute expedience and and ε-optimality, as detailed in the findings of [58].

3.3 The service placement problem

In the proposed approach, we assume that a set of fog nodes (fog devices) is denoted by $Fog = \{Fog_0, Fog_1, Fog_2, \ldots, Fog_d\}$. Fog_0 represents the cloud, and other fogs are also indicated by Fog_1 to Fog_d , respectively. d shows the total number of fogs. Let $App = \{App_1, \ldots, App_k, \ldots, App_n\}$ be the set of n IoT applications. Each App_k is a set of services: $App_k = \{s_{k,1}, \ldots, s_{k,t}, \ldots, s_{k,n_k}\}$, where $k = 1,2, \ldots, n$, $t = 1,2, \ldots, n_k$, and n_k represents the number of services within App_k, which may vary for each k. The cloud or fog infrastructure runs each App_k at the user's request. Each service $s_{k,t}$ represents the service *t* in App_k and $|App_k|$ indicates the number of services in App_k. The placement algorithm must be deploy each service in a proper fog node or cloud. The model defines each service based on resource requirements, such as $s_{k,t} \epsilon\ App_k$. The model denotes the resources associated with the QoS needs of each service as follows:

$$s_{k,t} = \{s_{k,t}^{CPU}, s_{k,t}^{RAM}, s_{k,t}^{Disk}, s_{k,t}^{BW}, s_{k,t}^{DL}, s_{k,t}^{size}\} \tag{5}$$

Where $s_{k,t}^{CPU}$, $s_{k,t}^{RAM}$, $s_{k,t}^{Disk}$, $s_{k,t}^{BW}$, $s_{k,t}^{DL}$ and $s_{k,t}^{size}$ represent the amount of processor usage, memory usage, storage capacity usage, bandwidth, deadline, and number of instructions to run the IoT service in milliseconds, respectively. The service placement problem is defined as follows. If $S_{i,j}$ represents the j^{th}service of the i^{th} IoT application, then the function of f maps each service of each application to the suitable fog node, and we have:

$$f: \bigcup_{k=1}^{n} App_k \rightarrow Fog \tag{6}$$

$$f(S_{i,j}) = fog_m \quad , m \in \{0,1,2, \ldots., d\} \tag{7}$$

In that case, the service placement problem becomes an optimization problem and is defined as follows:

$$\min_{f} \sum_{i=1}^{n} \sum_{j=1}^{n_k} cost(s_{i,j}, f(s_{i,j})) \tag{8}$$

Furthermore, the fog nodes determine how to deploy services according to their available resources. The selected fog node must have the necessary resources, including CPU, RAM, and storage, to support each service. The following three constraints must be satisfied. Equation (9) indicates that the CPU, primary memory, and storage needs of IoT services should remain within the accessible resources of the chosen fog node.

4. The proposed VASLA-FSP algorithm

This section explains the proposed framework for solving the service placement problem.

4.1 The proposed framework

The proposed service placement framework includes four steps, as illustrated in Figure 2. In the first step, we propose a fog computing model represented as a graph consisting of nodes and edges. In the second step, we map the fog nodes to the learning automata. In the third step, we determine the variable action set of the learning automata. Finally, we explain the service placement algorithm using VASLA.

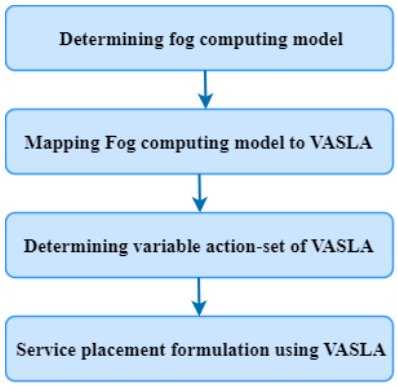


Fig. 2: The proposed framework for service placement in the fog computing environment.

4.1.1 Determining fog computing model

This section explains the proposed model for deploying IoT applications in a fog-based infrastructure. Figure 3 illustrates the architecture of fog computing, which divides into three layers: cloud layer, fog computing layer, and IoT device. The cloud layer appears at level zero and includes the nodes of cloud data centers. This layer provides services and storage resources for requests the fog layer cannot execute. The middle layer contains various fog devices or nodes with different processing capacities. Several gateways, called Fog and Cloud Gateways, connect the fog and cloud layers. Fog nodes at each level interact with IoT devices, and the network deploys other fog nodes. At the top layer, the user end devices in IoT environment, such as smartphones, sensors and actuators are serving as a user communication tools with the fog network by sending requests and receiving response. In this paper, we model the fog environment as a graph structure, where nodes represent fog nodes or clouds, and edges represent the links between them. Let $G =$

(N, L) be a directed graph, where $N = \{Fog_0, Fog_1, Fog_2, \ldots, Fog_d\}$ denote a set of fog nodes. We assume that, Fog_0 represented a cloud. Let $L \subset N \times N$ denote a set of links connecting these fog nodes or cloud; and $l_{(i,j)} \in L$ denotes a communication link from Fog_i to Fog_j .

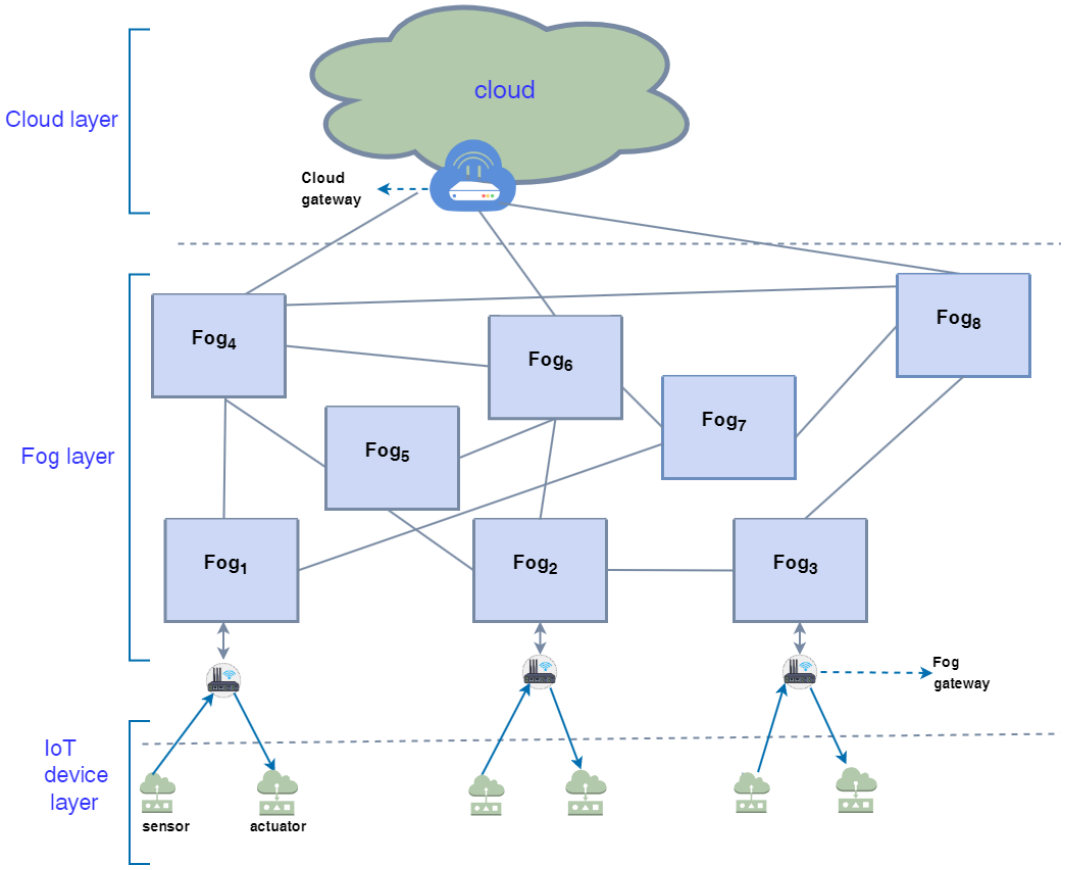


Fig. 3: Fog computing model.

4.1.2 Mapping fog computing model to VASLA

According to the fog computing model, the system maps each fog node to a variable action set learning automaton (VASLA), as illustrated in Figure 4.

$$\forall Fog_i \in Fog, \exists\, LA_i \in VASLA \; such \; that \; Fog_i \rightarrow LA_i \tag{9}$$

The action set of each automaton of LA_i is defined as follows:

$$\alpha(LA_i) = \left(\alpha_{i,0}, \ldots, \alpha_{i,j}, \ldots, \alpha_{i,d}\right) \tag{10}$$

In the above equation, $\alpha_{i,j}$ represents the j^{th} action executed by the automaton LA_i. Additionally, the value of $\alpha_{i,j}$ represents the probability of selecting Fog_j using LA_i. In other words, the number of actions of each automaton is equal to the total number of fog nodes and the cloud.

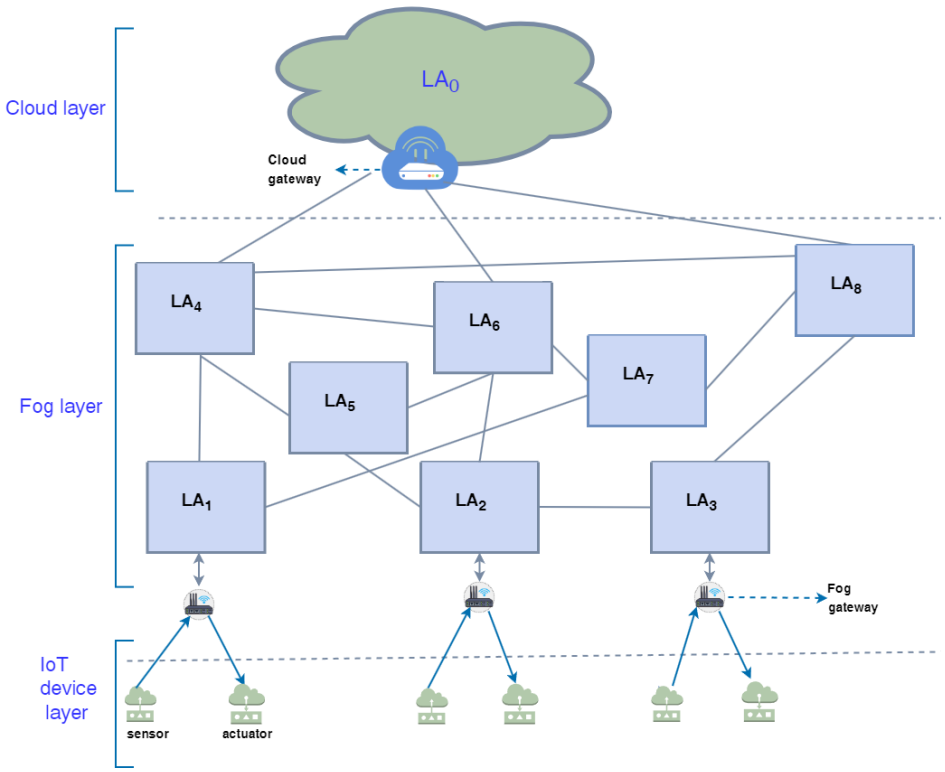


Fig. 4: Mapping the fog topology to VASLA.

4.1.3 Determining variable action-set of VASLA

At a given time *t*, only a subset of variable actions is selected to form the active action set for each automaton LA_i. The active action set of LA_i is determined as follows:

1) Action $\alpha_{i,i}$ is always considered as an active action by LA_i , allowing the automaton to evaluate its own service placement.

2) If there exists an edge of length one from Fog_i to Fog_j and the conditions in Eq. 8 are satisfied, then action $\alpha_{i,j}$ is added to the active action set of LA_i.

Example 1:

This section explains an example of how VASLA selects active actions. As shown in Figure 5, the fog topology includes the following nodes: $Fog = \{Cloud = Fog_0, Fog_1, Fog_2, \ldots, Fog_8\}$. Each node in this fog computing topology corresponds to a VASLA, and the complete set of learning automata comprises $LA = \{LA_1, LA_2, \ldots, LA_8\}$. We determine the action set of each automaton as:

$LA_1 = (\alpha_{1,1}, \alpha_{1,4}, \alpha_{1,7})$, $LA_2 = (\alpha_{2,2}, \alpha_{2,5}, \alpha_{2,6}, \alpha_{2,3})$, $LA_3 = (\alpha_{3,3}, \alpha_{3,2}, \alpha_{3,8})$, $LA_4 = (\alpha_{4,4}, \alpha_{4,1}, \alpha_{4,5}, \alpha_{4,6}, \alpha_{4,8}, \alpha_{4,0})$, $LA_5 = (\alpha_{5,5}, \alpha_{5,6}, \alpha_{5,4}, \alpha_{5,2})$, and $LA_6 = (\alpha_{6,6}, \alpha_{6,5}, \alpha_{6,7}, \alpha_{6,8}, \alpha_{6,4}, \alpha_{6,2}, \alpha_{6,0})$, $LA_7 = (\alpha_{7,7}, \alpha_{7,8}, \alpha_{7,6}, \alpha_{7,1})$, $LA_8 = (\alpha_{8,8}, \alpha_{8,3}, \alpha_{8,4}, \alpha_{8,6}, \alpha_{8,7}, \alpha_{8,0})$.

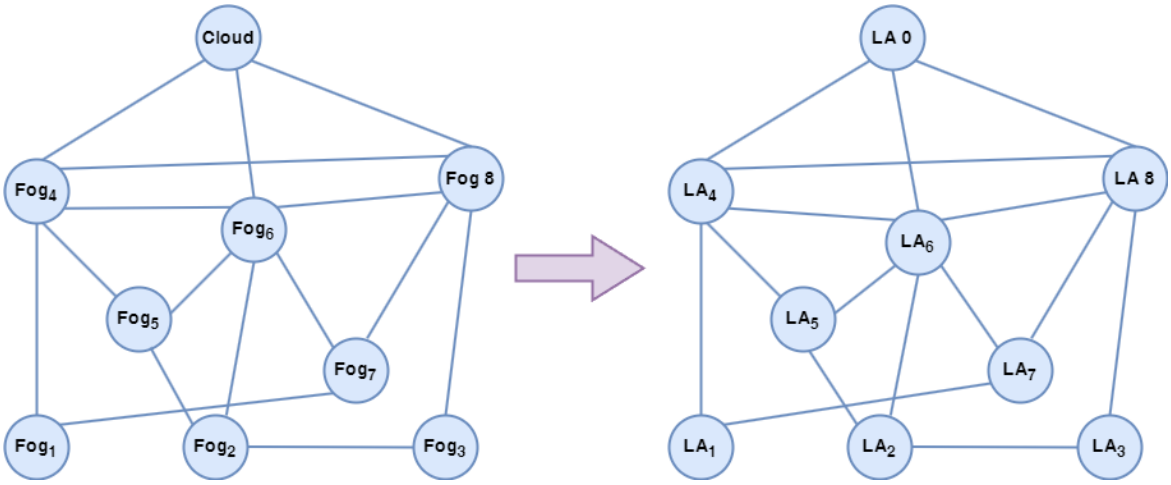


Fig. 5: Mapping the fog topplogy to VASLA.

Table 2 presents the specifications of resources corresponding to services and cloud and fog nodes in terms of CPU, RAM, and Disk capacity.

Table 2 : Specifications of resources associated with services, cloud and fog nodes.

Item	CPU (GHz)	RAM (GB)	Disk (GB or TB)
Fog_1	2	4	35 GB
Fog_2	2	4	60 GB
Fog_3	4	8	100 GB
Fog_4	6	12	150 GB
Fog_5	3	6	80 GB
Fog_6	8	16	200 GB
Fog_7	4	8	100 GB
Fog_8	5	10	120 GB
Fog_9	6	12	150 GB
Service S_1	4	8	40 GB
Service S_2	4	8	50 GB
Service S_3	3	11	80 GB
Service S_4	5	9	100 GB
Cloud	32	128	2 TB

Scenario 1 : Suppose LA_4 wants to perform service placement for the service of S_1 at time t_1 , as shown in Figure 5. Using the rules explained in section 3.2.3 and Table 2, the active action set of LA_4 is determined as $(\alpha_{4,4}, \alpha_{4,6}, \alpha_{4,8}, \alpha_{4,0})$. $\alpha_{4,5}$ cannot be selected because its available resources (RAM, CPU) cannot meet the service requirements. Additionally, $\alpha_{4,1}$ cannot be selected because its available resources (RAM, CPU, and disk space) cannot meet the service requirements.

Scenario 2: Supposed to perform service placement for the service of S_1 at time t_2. Additionally, we joined a new Fog_9 as shown in Figure 6. The active action set of LA_2 is updated as $(\alpha_{2,6}, \alpha_{2,9})$ $\alpha_{2,2}$ and $\alpha_{2,5}$ cannot be selected because their available resources (RAM, CPU) cannot meet the service requirements.

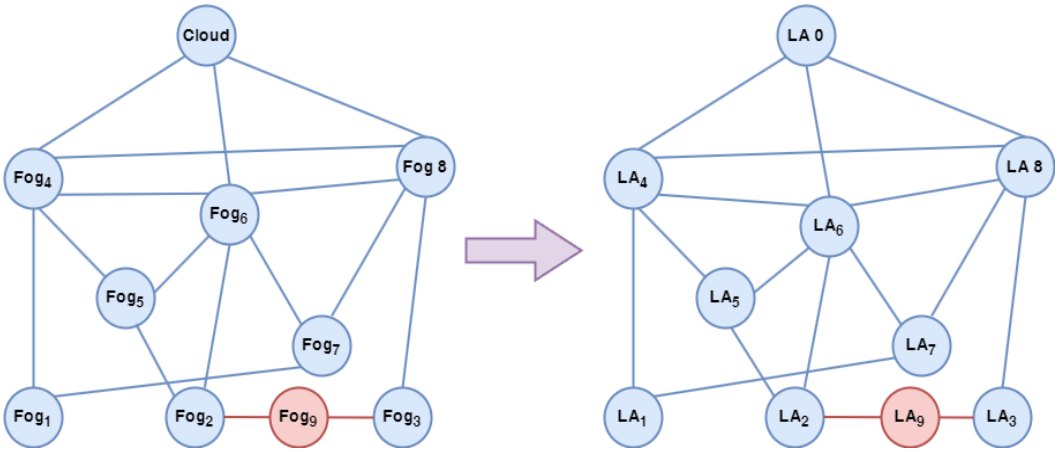


Fig. 6: Joining a new fog to the fog topology.

Scenario 3: Suppose LA_4 intends to place the service S_1 at time t_3. Additionally, we removed Fog_6 as shown in Figure 7. The active action set of LA_4 is determined as $(\alpha_{4,4}, \alpha_{4,8}, \alpha_{4,0}, \alpha_{4,1})$. $\alpha_{4,5}$ cannot be selected because its available resources (RAM, CPU) cannot meet the service requirements.

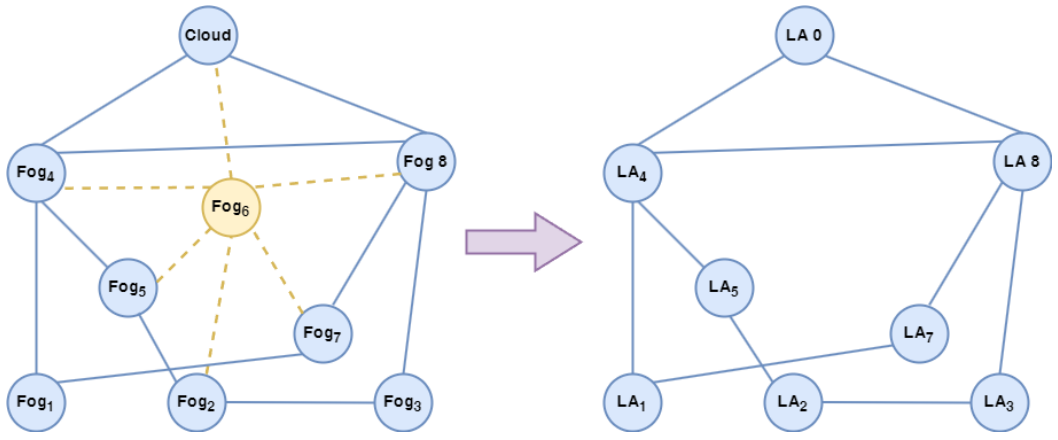


Fig. 7: Leaving a fog node from the fog topology.

4. 1. 4 Service placement formulation using VASLA

After performing steps 1, 2, and 3 of the proposed frameworks, we consider a learning automaton (VASLA) for the desired fog topology. This section describes how the service placement problem uses the obtained VASLA to allocate services across fog nodes optimally. The proposed algorithm defines the fog nodes closest to the end users as the edge fog nodes. For example, as Figure 3 illustrates, we consider Fog_1, Fog_2, and Fog_3 as edge fog nodes. A key feature of edge fog nodes is that their corresponding automata initiate the service placement algorithm. Suppose we have L edge fog nodes, denoted as $EdgeFogs = \{EdgeFog_1, \dots, EdgeFog_L\}$, and n IoT applications, represented as $App = \{App_1, \dots, App_k, \dots, App_n\}$, where each application App_k consists of a set of services: $App_k = \{s_{k,1}, \dots, s_{k,t}, \dots, s_{k,n_k}\}$. In each iteration of the proposed algorithm, all automata corresponding to each edge fog node perform placement sequentially for App_1, then App_2, and finally App_n. The placement of an application, such as App_k, in an edge fog node, such as $Fog_{currentFog}$, consists of the following steps:

Step 1: Obtain variable actions: Let us denote the automaton corresponding to $Fog_{currentFog}$ as $LA_{currentfog}$. This automaton first considers the first service of the application and determines its set of variable actions, as described in Section 4. Variable actions may be empty during execution. To solve this problem, the system uses a stack. This stack stores the list of fog nodes where the system has already performed service placement. In this way, when the system cannot find variable actions for an automaton, it retrieves other fogs for service placement by popping a fog node from the stack. When the stack is empty, the system performs service placement in the cloud.

Step 2: Select an action and deploy the service: Suppose the automaton corresponding to the $Fog_{currentFog}$, namely $LA_{currentfog}$, selects an action called $\alpha_{currentFog,nextFog}$ from its set of variable actions. This selection indicates that the automaton will place the desired service in $Fog_{nextFog}$.

Step 3: Cost function evaluation and learning process: In this step, the automaton $LA_{currentfog}$ decides whether to apply a reward or a penalty to the chosen action $\alpha_{currentFog,nextFog}$. The automaton makes this decision such that if placing the service in the current iteration results in lower energy consumption and delay compared to the previous iteration, then $LA_{currentfog}$rewards the action $\alpha_{currentFog,nextFog}$; otherwise, it penalizes it. Subsequently, the automaton $LA_{currentfog}$ continues by placing the second service of App_1, starting from the previously selected fog node, i.e., $Fog_{nextFog}$ or cloud. The system follows the above steps to place the remaining application services.
As a result, the automaton associated with $Fog_{currentFog}$ places all services of App_1. After all automata place App_1, they repeat the same process for App_2 and the rest of the applications, proceeding sequentially from the first to the last edge fog node. The system calculates the cost function using Equation (11).

$$Cost = Energy\ consumption + Delay \tag{11}$$

The algorithm considers the iteration with the lowest cost as the optimal service placement. The energy consumption of fog node fog_j , denoted as E_{fog_j}, is the sum of the energy used during data transmission, $E_{fog_j}^{trans}$, and the energy spent on processing the services, $E_{fog_j}^{proc}$. Equation (12) calculates this value.

$$E_{fog_j} = E_{fog_j}^{proc} + E_{fog_j}^{trans} \tag{12}$$

The processing energy for a single service s_i on fog_j is the power consumed by IoT devices and edge/fog nodes while processing data and executing services, as follows:

$$E_{fog_j}^{proc} = (P_{fog_j}^{idle} + \alpha_{i,j} \times \frac{size_i}{\gamma_{fog_j}} \times P_{fog_j}^{proc}) \times t \tag{13}$$

Where, $P_{fog_j}^{idle}$ represents the energy consumed when the fog node j is idle. $\alpha_{i,j}$ indicates service allocation coefficient. The system assigns the value of $\alpha_{i,j}$ as 1 when fog node j processes service i; otherwise, it is 0. $Size_i$ represents the size of service s_i that fog_j hosts. γ_{fog_j} indicates the computational capacity of fog node j. $P_{fog_j}^{proc}$ shows the maximum power that fog node j consumes during service processing, and t refers to the time duration over which we measure the energy consumption. The transition energy is the amount of energy consumed by a system or device for transmitting, receiving, and processing data over a communication network. As a result, the total energy consumption for transferring tuples/data between two fog nodes, considering service delay and geographic locations, is determined using Equation (14), where $UC_E_{fog_j}$ represents the unit computing energy consumption of fog node j which defines how much energy is consumed per unit of computation, $\lambda_i^{(fog_j, fog_k)}$ represent a parameter that describes the processing or transmission rate between fog nodes j and fog node k, $X_{s_i}^{fog_j}$ is a binary variable representing whether service s_i has been assigned to fog node j (1 if assigned, zero otherwise). t is the time duration of data transmission.

$$E_{fog_j}^{trans} = X_{s_i}^{fog_j} \times \lambda_i^{(fog_j, fog_k)} \times UC_E_{fog_j} \times t \tag{14}$$

Equation (15) determines the overall delay for a single service s_i on a specific fog node fog_j by summing four components: the transmission delay required for transmitting the service from the IoT device into the fog infrastructure ($D_{s_i, fog_j}^{IoT \rightarrow fog}$), the processing delay of the service within the fog infrastructure ($D_{s_i, fog_j}^{process}$), and the delay incurred when offloading the service to the cloud due to insufficient resources ($D_{s_i, fog_j}^{fog \rightarrow cloud}$). Fog nodes process tasks (tuples) in a FIFO (First-In-First-Out) queue in a fog computing environment .The processing time in the queue is commonly denoted as

(D_{s_i,fog_j}^{Queue}), where D_{s_i,fog_j}^{Queue} represents the delay experienced by tasks waiting in the queue until the fog node processes them.

$$D_{s_i,fog_j}^{Total} = (D_{s_i,fog_j}^{IoT\rightarrow fog} + D_{s_i,fog_j}^{process} + D_{s_i,fog_j}^{fog\rightarrow cloud} + D_{s_i,fog_j}^{Queue}) \quad (15)$$

$$D_{s_i,fog_j}^{IoT\rightarrow fog} = \left(SZ_{s_i}^{rq} + SZ_{s_i}^{rs}\right).l_{IOT}^{fog_i} \quad (16)$$

$$D_{s_i,fog_j}^{process} = \frac{SZ_{s_i}^{rq}.t(s_i,fog_j)}{S(s_i,fog_j)} \quad (17)$$

$$D_{s_i,fog_j}^{fog\rightarrow cloud} = \left(SZ_{s_i}^{rq}.d_{fog_j}^{cloud} + SZ_{s_i}^{rq}.d_{cloud}^{process}\right) + SZ_{s_i}^{res}.d_{cloud}^{IoT} \quad (18)$$

$$D_{s_i,fog_j}^{Queue} = d_{s_i}^{wait} \quad (19)$$

Where $SZ_{s_i}^{rq}$ is the request data size, $SZ_{s_i}^{rs}$ is the response data size, $l_{IOT}^{fog_i}$, is the transmission delay between the IoT device and fog node i, $t(s_i,fog_j)$ is the number of CPU cycles required to process s_i, $S(s_i,fog_j)$ processing capacity of fog node j , $d_{fog_j}^{cloud}$ is the communication delay from fog node j to the cloud, $d_{cloud}^{process}$ is the cloud processing time per unit of data, $SZ_{s_i}^{res}$ is the result data size returned to the IoT, d_{cloud}^{IoT} represents the delay between the cloud and the IoT, and $d_{s_i}^{wait}$ is the time that the service s_i waits in the fog node's queue.

Step 4: Finding the optimal service placement:

Each fog node calculates the energy usage and application delay at the end of each iteration. Averaging these values provides the mean energy consumption and delay for the iteration.

Example 2:

Suppose an application named App_1, which includes services S_2, S_3, and S_4 (as shown in Table 2), wants to perform service placement starting from $EdgeFogNode_2$. According to step 1 of the proposed algorithm, the automaton corresponding to the $EdgeFogNode_2$, i.e., LA_2, considers the first application service, S_2 , and begins searching for the variable action-set of that automaton. This step considers Fog_2 as the $currentFog$. According to Table 2 and Section 4.1.3, LA_2 selects its set of variable actions as active actions. In step 2, LA_2 considers only the active actions $(\alpha_{2,3}, \alpha_{2,6})$ from its complete action set ($\alpha_{2,2}$, $\alpha_{2,3}$, $\alpha_{2,5}$, $\alpha_{2,6}$) and randomly selects one of them. Suppose it selects the action $\alpha_{2,3}$; in that case, it places service S_2 in Fog_3. Then, it pushes Fog_2 onto the stack. Step 3 calculates the energy and delay for service S_2 of App_1 during the current iteration, taking into account Fog_2. The map stores the energy consumption, delay, and iteration number. The process determines whether the chosen action should be rewarded or penalized by comparing the energy consumption and delay values of the current iteration with those of the previous iterations stored in the map. Assuming that the values of energy consumption and delay in the current iteration and the previous iteration are (energy consumption previous = 25 megajoules, delay previous = 450 milliseconds) and (energy energy consumption current = 10 megajoules , delay current = 260 milliseconds), respectively. LA_2, i.e. the automaton $LA_{currentfog}$,

rewards the action $\alpha_{2,3}$ according to equation (2) and the Fog_3 is pushed into the stack. Figure 8 illustrates the placement of the first service, S_2.

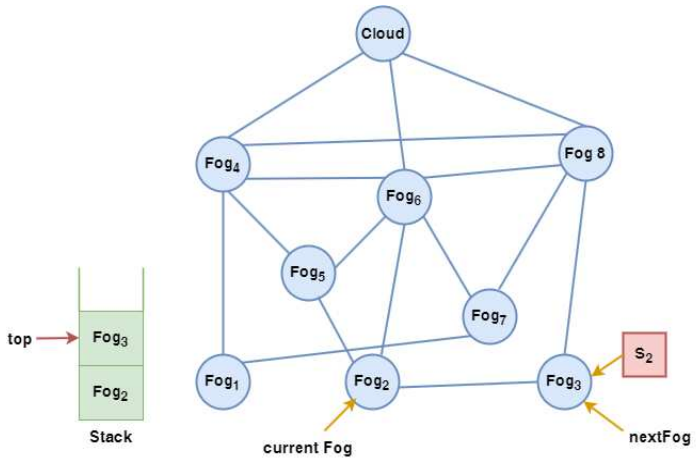


Fig. 8: Service placement for service S_2 .

After placing service S_2, it is time to put the second service, S_3. The automaton corresponding to Fog_3 (i.e., LA_3) will perform the placement of this service. Before deploying this service, the $currentFog$ will point to the $nextFog$, and the automaton will deploy service S_3 in the same way it deployed service S_2. For this purpose, LA_3 calculates its set of variable actions according to Table 4. Because LA_3 finds the variable action set empty, it leaves Fog_3 from the top of the stack, and it uses Fog_2 as the current fog to place the S_3 service. According to Table 4, LA_2 calculates the variable action set as ($\alpha_{2,6}$). As a result, it places the S_3 service in Fog_6 and it pushes Fog_6 onto the stack. After the placement of service S_3, similar to the placement of service S_2, the energy and delay values are calculated and compared with the previous energy and delay values in the last iteration. Assuming that the energy or delay calculated for service S_3 in Fog_6 is greater, the action $\alpha_{2,6}$ is penalized according to Equation 3. Figure 9 shows the placement of the S_3 service.

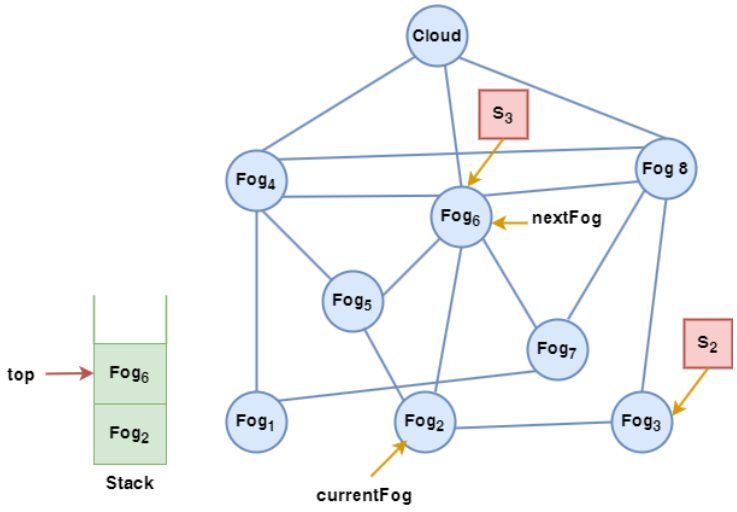


Fig. 9: Service placement for service S_3 .

After the placement of service S_3, it is time to place the third service, S_4. The automaton corresponding to Fog_6 (i.e., LA6) will perform the placement of this service. Before deploying this service, the $currentFog$ will point to the $nextFog$, and the service placement operation for the S_4 service will be similar to the deployment of the S_3 service. For this purpose, LA_6 calculates its variable actions according to Table 4 and section 4.1.3. The set of action variables of the LA_6 automaton are selected as variable actions ($\alpha_{6,6}$, $\alpha_{6,8}$, $\alpha_{6,4}$, $\alpha_{6,0}$) according to Table 2 and Section 4.3.1. Subsequently, the automaton randomly selects one action from its action set. We assume that this action is equal to $\alpha_{6,6}$. As a result, the automaton places service S_4 in Fog_6. Since Fog_6 is already at the top of the stack, the automaton skips pushing it again. Similar to the placement of service S_2, the energy consumption and delay for service S_3, after placing it in Fog_6, were calculated. Based on the comparison result, the automaton assigned a reward or penalty to action $\alpha_{6,6}$. Figure 10 shows the placement of the S_4 service.

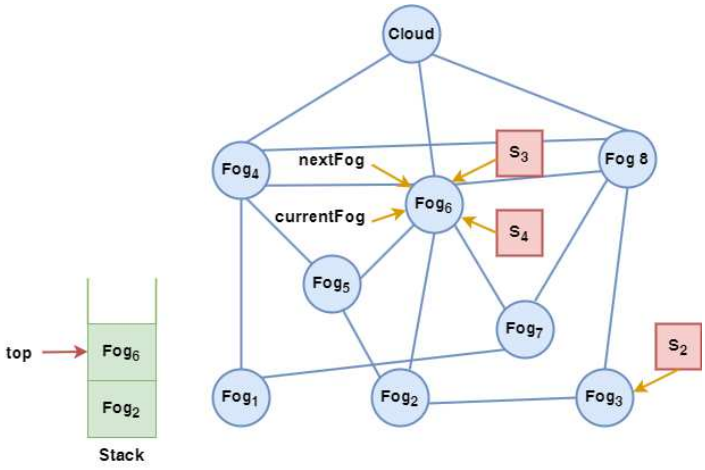


Fig. 10: Service placement for service S_4.

After deploying all the services of App_1 from the first edge fog node, the stack is empty, and $currentFog$ points to the second fog node, Fog_1. Similar to the deployment mentioned above, the process begins by deploying all App_1 services from Fog_1. The same method places all App_1 services for Fog_3. The placement of other applications follows the same procedure used for App_1. As a result, at the end of the algorithm iteration, all applications, such as App_2, App_3, ..., will be deployed from all edge fog nodes. At the end of the algorithm iteration, the algorithm executes step 4 and compares the placement made in this iteration with the best solution found so far. If the answer is less than the best answer found, this answer is considered the best.

4.2 Pseudo-code of the proposed algorithm

This section provides the pseudocode of the proposed algorithm. Algorithm 1 presents the main body of the proposed VASLA-FSP algorithm.

Algorithm 1: VASLA-FSP

```
   // Determining fog computing model
1  fogTopology ← createFogTopology()
   // Mapping fog computing model to VASLA
2  LA ← build-VASLA(fogTopology)
   // All the necessary information for service placement is stored in the Map
3  Map ← createMap()
4  itr ← 1
5  repeat
       // Clear the Map
6      empty(Map)
       // Remove all the placed services from the fog nodes
7      empty(fogNodes)
       // Perform service placement
8      servicePlacement(itr)
       // Check the found solution for service placement
9      if Obtained solution is the best one found so far then
10         Store it as a BestSolution
11     end if
12     itr ← itr+1
13 until VASLA converges to the best actions;
14 return BestSolution
```

In line 1 of the proposed algorithm, the fog topology is constructed as a graph using the ifogsim simulator, as employed in step 1 of the proposed framework. The fog topology is created by calling the createFogTopology method in line 1. Algorithm 2 provides the code for calling the createFogTopology method. In line 2, a map stores the service placement results. This map is composed of two elements: the key and the value. The key part contains the iteration number, the application name, the name of the service deployed on the fog node, and the name of the fog node found for service placement.

Additionally, the value stores the amount of energy used and the delay caused by the service assigned to the fog node. In line 3, after constructing the fog topology by step 2 of the proposed framework, a learning automaton with variable structure is allocated to each fog node. Section 3 determines the automaton's action set. Algorithm 3 provides the code for mapping the learning automaton to each fog node. Lines 5 to 10 form the key section of the proposed VASLA-FSP algorithm, which performs these steps for all services of all applications in each iteration. Line 7 of Algorithm 1 addresses the service placement analysis and searches for the optimal solution. Algorithm 4 provides the code for the service placement algorithm, and Section 4.1.3 provides the details of this algorithm. In line 8, the process empties all deployed services in all fog nodes for redeployment.

Algorithm 2: createFogTopology

```
1 fogTopology ← create a graph of fog nodes using ifogsim simulator
2 return fogTopology
```

Algorithm 3: Build-VASLA($fogTopology$)

```
LA ← {}
foreach fog_i where i ∈ {1, ..., n} do
    assign LA_i to fog_i
    define actionSet of LA_i
    LA ← LA ⋃ LA_i
end foreach
return LA
```

Algorithm 4: servicePlacement($iterationNumber$)

```
foreach EdgeFogNode_l where l ∈ {1, ..., L} do
  foreach APP_k where k ∈ {1, ..., n} do
    stack=[]
    currentFog ← L
    foreach service s_{k,t} where t ∈ {1, ..., n_k} do
      isPlaced ← False
      if stack.top! = fog_{currentFog} then
        stack.push(fog_{currentFog})
      end if
      while ( !isplaced ) do
        // The Automaton of the Current Fog Obtains its Variable Actions.
        variableActions ← getVariableActions(fog_{currentFog}, app_k, s_{k,t})
        if |variableActions| == 0 then
          isFind ← False
          while ( !isFind ) do
            if Stack is empty then
              currentFog ← Cloud
              else
                currentFog ← stack.pop()
              end if
            end if
            variableActions ← getVariableActions(fog_{currentFog}, app_k, s_{k,t})
            if |variableActions|! = 0 then
              isFind ← True
            end if
          end while
        end if
        // The Automaton of the Current Fog Selects Action α_{currentFog,nextFog}
        α_{currentFog,nextFog} ← selectAction(fog_{currentFog},variableActions)
        // The Service is deployed in the Next Fog.
        isPlaced ← True
        place s_{k,t} in fog_{nextFog}
        // The Energy Consumption and Delay for the Service are obtained at the Next Fog.
        e_c ← getEnergyConsumption(fog_{nextFog})
        d_c ← getDelayConsumption(fog_{nextFog})
        key ← (itr, app_k, s_{k,t}, fog_{nextFog})
        value ← (e_c, d_c)
        Map ← Map ⋃ (key,value)
        if stack.top! = fog_{nextFog} then
          stack.push(fog_{nextFog})
        end if
        // The Automaton of Current Fog decides if the selected Action gets a Reward or Penalty.
        learningAutomataResponse(fog_{currentFog}, fog_{nextFog}, α_{currentFog,nextFog}, itr, app_k, s_{k,t})
        currentFog ← nextFog
      end while
    end foreach
  end foreach
end foreach
```

Algorithm 5: getVariableActions($fog_i, app_k, s_{k,t}$)

```
1 variableActions ← {}
2 foreach fog_j where j ∈ {1, ..., i, ..., n} do
3 |   if s_{k,t}^{CPU} ≤ fog_j^{CPU} and s_{k,t}^{RAM} ≤ fog_j^{RAM} and s_{k,t}^{DISK} ≤ fog_j^{DISK} then
4 |   |   variableActions ← variableActions ⋃ α_{i,j}
5 |   end if
6 end foreach
7 return variableActions
```

Algorithm 4 performs the service placement. Section 4.1.4 provides details and examples of Algorithm 4. Algorithm 5 obtains the list of variable actions of the learning automaton, as described in Section 4.1.3.

Algorithm 6: learningAutomataResponse($fog_{currentFog}, fog_{nextFog}, \alpha_{currentFog,nextFog}, itr, app_k, s_{k,t}$)

```
1 key_(itr) ← (itr, app_k, s_{k,t}, fog_{nextFog})
2 key_(itr−1) ← (itr − 1, app_k, s_{k,t}, fog_{nextFog})
3 [e_c, d_c] ← Map(key_(itr−1))
4 [e_c', d_c'] ← Map(key_itr)
5 if e_c' ≤ e_c and d_c' ≤ d_c then
6 |   Reward action of α_{currentFog,nextFog} from fog_{currentFog}
7 else
8 |   Penalize action of α_{currentFog,nextFog} from fog_{currentFog}
9 end if
```

Algorithm 6 determines the reward or penalty for each automaton based on the chosen action. To do this, in lines 3 and 4, the automaton corresponding to the placed service is retrieved from the map and calculates the delay and energy usage. If the energy consumption and delay are better than in the previous iteration, the chosen action receives a reward; otherwise, it gets a penalty.

5. Experimental results and analysis

5.1 Simulation setup

The proposed method is implemented in the iFogSim simulator using Java programming on a Dell laptop equipped with an Intel Core i7-10700U processor at 3.5 GHz, 16 GB of RAM, and running Windows 11. We use three different scenarios for simulations. In all the experiments discussed in this paper, the linear reward-penalty L_(R-P) scheme updates the action probability vectors, with the reward (a) and penalty (b) set to 0.1. We consider the parameters of the other algorithms according to the specifications in the relevant articles.

5.2 Application module feature

In the experiments, we use the iFogSim simulator [40] to evaluate the efficiency of the proposed approach. This simulator comprises five main components, or modules: Camera (captures video),

Motion Detector (detects movement), Object Detector (recognizes objects), Object Tracker (tracks objects), and PTZ Controller (manages the camera's pan, tilt, and and zoom movements). Figure 11 illustrates the connections between the modules and their data flow. Table 3 details each module's requirements for resources (such as CPU or memory).

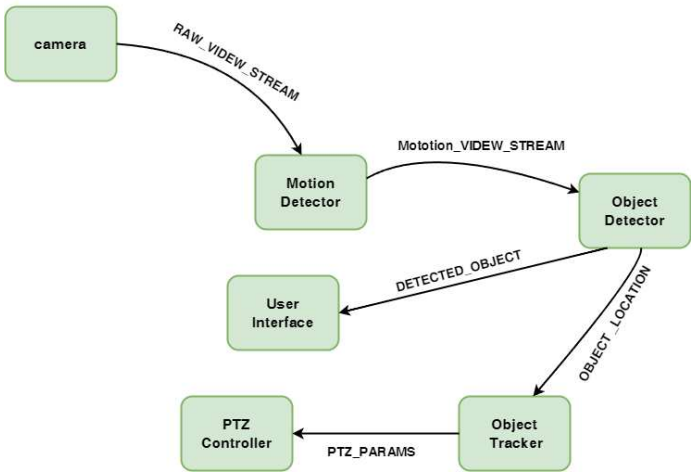


Fig. 11: Modules and edges of intelligent monitoring application [59].

Table 3: Description of resource details CPU (MIPS), RAM(GB) for different applications.

Application Name	Number of services	App_k(" $S^{CPU}_{(K,1)}, S^{RAM}_{(K,1),}$ ", " $S^{CPU}_{(K,2)}, S^{RAM}_{(K,2),}$ ", … , " $S^{CPU}_{(K,m)}, S^{RAM}_{(K,m),}$ ")
App_1	3	App_1("50,10","400,10","200,10"),
App_2	6	App_2("50,10","400,10","200,10","600,10","200,10","400,10")
App_3	9	App_3("50,10","400,10","200,10","600,10","200,10","400,10","200,10","600,10","200,10")
App_4	12	App_4("50,10","400,10","200,10","600,10","200,10","400,10","200,10","600,10","200,10","300,10","400,10","300,10")

5.3 Fog device feature

The simulation utilized three topologies, each containing 13, 28, and 57 fog nodes. The last layer of the fog topology contains the edge fog nodes, with 6, 13, and 26 nodes, respectively. These topologies are labeled FG (13,6), FG (28,13), and FG (57,26). Tables 4, 5, and 6 present the resource demand details for fog nodes in each topology. Table 7 shows the processing capabilities of various hardware components in the topology, including downstream (download) speed (Mbps), upstream (upload) speed (Mbps), and the energy consumption of each processing node in Idle states.

Table 4. Description of the fog topology containing 13 fog nodes and 6 edge fog nodes $FG(13, 6)$.

$\mathbf{Fog_{(i)}(CPU, RAM)}$
$Fog_{(1)}$(44000,20000), $Fog_{(2)}$(22000,12000), $Fog_{(3)}$(11000,8000), $Fog_{(4)}$(12600,8000), $Fog_{(5)}$(8000,6000), $Fog_{(6)}$(5600,4000), $Fog_{(7)}$(4200,3000), $Fog_{(8)}$(4000,2000), $Fog_{(9)}$(4000,2000), $Fog_{(10)}$(4000,2000), $Fog_{(11)}$(4000,2000), $Fog_{(12)}$(4000,2000), $Fog_{(13)}$(4000,2000).

Table 5. Description of the fog topology containing 28 fog nodes and 13 edge fog nodes $FG(28, 13)$.

$\mathbf{Fog_{(i)}(CPU, RAM)}$
$Fog_{(1)}$(44000,38000), $Fog_{(2)}$(22000,18000), $Fog_{(3)}$(30000,15000), $Fog_{(4)}$(20000,16000), $Fog_{(5)}$(18000,14000), $Fog_{(6)}$(16000,12000), $Fog_{(7)}$(8000,6000), $Fog_{(8)}$(12600,8000), $Fog_{(9)}$(14000,10000), $Fog_{(10)}$(18000,14000), $Fog_{(11)}$(10000,8000) $Fog_{(12)}$(16000,12000), $Fog_{(13)}$(6000,4000), $Fog_{(14)}$(5600,4000), $Fog_{(15)}$(4200,3000), $Fog_{(16)}$(4000,2000), $Fog_{(17)}$(4000,2000), ... , $Fog_{(28)}$(4000,2000)

Table 6. Description of the fog topology containing 57 fog nodes and 26 edge fog nodes $FG(57, 26)$.

$\mathbf{Fog_{(i)}(CPU, RAM)}$
$Fog_{(1)}$(44800,40000), $Fog_{(2)}$(22400,20000), $Fog_{(3)}$(33600,30000), $Fog_{(4)}$(18000,16000), $Fog_{(5)}$(17000,15000), $Fog_{(6)}$(19000,17000), $Fog_{(7)}$(20000,18000), $Fog_{(8)}$(8400,8000), $Fog_{(9)}$(8400,8000), $Fog_{(10)}$(5600,6000), $Fog_{(11)}$(5600,6000), $Fog_{(12)}$(16000,14000), $Fog_{(13)}$(15000,13000), $Fog_{(14)}$(14000,12000), $Fog_{(15)}$(12000,10000), $Fog_{(16)}$(8400,8000), $Fog_{(17)}$(8400,8000), $Fog_{(18)}$(18000,16000), $Fog_{(19)}$(8600,6000), $Fog_{(20)}$(12000,10000), $Fog_{(21)}$(13000,8000), $Fog_{(22)}$(16000,14000), $Fog_{(23)}$(9400,6000), $Fog_{(24)}$(16000,10000), $Fog_{(25)}$(5600,6000), $Fog_{(26)}$(4200,2000), $Fog_{(27)}$(5600,4000), $Fog_{(28)}$(14000,8000), $Fog_{(29)}$(12000,10000), $Fog_{(30)}$(8000,6000), $Fog_{(31)}$(6000,4000), $Fog_{(32)}$(4000,2000), $Fog_{(33)}$(4000,2000), ... , $Fog_{(57)}$(4000,2000).

Table 7. Specifications of IoT devices.

Device type	Downstream (Mbps)	Upstream (Mbps)	Power consumption(W)
Cloud	10,000	100	1521
Fog	10,000	10,000	5378

5.4 Evaluation Scenarios

To implement the proposed algorithm, we consider three different scenarios. Tables 8, 9, and 10 define the specifications of each scenario. Each scenario consists of four applications, each with a different number of services and a distinct set of edge fog nodes. Multiplying the service size by the number of edge fog nodes determines the total service.

Table 8: Description of fog nodes and services within the first scenario.

Fog Graph	Application Name	ServiceSize	Number of Edge Fog Node	Total Service
FG(13, 6)	$Application_1$	4	6	24
	$Application_2$	8	6	48
	$Application_3$	12	6	72
	$Application_4$	16	6	96

Table 9: Description of fog nodes and services within the second scenario.

Fog Graph	Application Name	ServiceSize	Number of Edge Fog Node	Total Service
FG(28 , 13)	$Application_1$	4	13	52
	$Application_2$	8	13	104
	$Application_3$	12	13	156
	$Application_4$	16	13	208

Table 10: Description of fog nodes and IoT services within the third scenario.

Fog Graph	Application Name	ServiceSize	Number of Edge Fog Node	Total Service
FG(57, 26)	$Applicatin_1$	4	26	104
	$Application_2$	8	26	208
	$Appication_3$	12	26	312
	$Appication_4$	16	26	416

5.5 Performance metrics

We have defined the evaluations and comparisons of the proposed algorithm according to the following metrics: energy consumption, tuple processing execution delay, IoT program execution delay, cost function, network usage, and execution time. This section describes these metrics as the main comparison results.

- **Energy consumption:**

Equation (15) calculates this metric as the overall energy usage of the active fog nodes running IoT applications, including the energy used for Computation and data communication across all sources in a fog network with heterogeneous nodes.

- **Application Loop Delay:**

An IoT application in a fog computing infrastructure consists of a sequence of services distributed across multiple nodes, referred to as an application loop. The delay time required to execute a path of independent services in the application defines this metric. The delay represents the time to

receive input, process it through the services, and produce output. Equation (20) calculates the loop delay.

$$Total\ Tuple\ Delay = \sum_{i=1}^{N} (Execution\ Delay_i + Communication\ Delay_i) \quad (20)$$

Where N denotes the total number of services within the path, $Execution\ Delay_i$ indicates the CPU delay at service i, and $Communication\ Delay_i$ denotes the delay from service i to service $i+1$.

- **Tupel CPU execution Delay:**

Tuple CPU execution delay refers to the time a processing unit, such as a cloud server or fog node, takes to execute a computational task, known as a "tuple," after receiving it. This delay depends on the task's size and the node's processing power. We define this metric as the delay in executing the tuple process and calculate it according to Equation (21).

$$Execution\ Delay_{CPU} = \frac{Tuple\ length(MI)}{Proceesing\ Power\ of\ Node(MIPS)} \quad (21)$$

Where Tuple length represents the number of million instructions required to process the tuple, and the *Processing Power of the Node* denotes the million instructions per second (MIPS) that the fog node or cloud server can handle.

- **Network Usage:**

Network usage is the total data transmitted and received between IoT nodes, fog, and cloud during a period. IoT devices transmit their generated data as byte-sized tuples forwarded to fog nodes or higher-layer cloud servers. This metric depends on the request size and the overall transmission delay between the originating device and the destination node. We calculate it according to Equation (22).

$$Network\ usage = Latency * TupleNewSize \quad (22)$$

Where *Latency* and $TupleNewSize$ represent the delay for data transmission at each fog node and the corresponding size of every transmitted tuple, respectively.

- **Total Execution Time:**

Equation (23) defines the overall execution time for every optimal placement as the difference between the completion time of the project execution ($ETime_{proj}$) and the start time of the simulation ($STime_{proj}$) using iFogSim.

$$Total\ execution\ time = ETime_{proj} - STime_{proj} \quad (23)$$

5.6 Analysis of the results

This section discusses the simulation results and comparisons. We assess the efficiency of the VASLA-FSP algorithm by comparing it against a set of other modern service placement algorithms in fog computing environments, including DLA-FMP (Abofathi et al.), ODSPS (Azizi et al., 2024), DCSP (Lin et al., 2024), as well as WOA-FSP (Ghobaei-Arani & Shahidinejad, 2022). The evaluation metrics, including energy consumption, delay, cost function, tuple CPU execution latency, network resource usage, and runtime, have been utilized in the evaluation results.

Experiment 1

We measured and compared the energy efficiency of the proposed algorithm to that of other algorithms in Scenarios 1, 2, and 3. Figure 12 (a) shows the energy consumption of each service placement algorithm according to the number of services in Scenario 1. As illustrated in Figure 12 (a), energy consumption increases with the number of IoT services in the system. The results show that the VASLA-FSP method achieves lower energy usage than the other approaches. This improvement is primarily due to its optimal utilization of computational resources, which effectively minimizes the total energy consumption of fog nodes in the fog computing infrastructure.

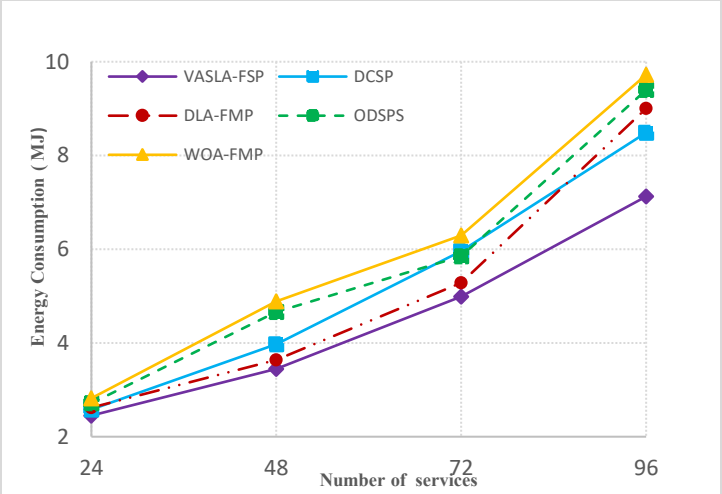


Fig. 12(a): Comparison of the energy consumption for different algorithms in scenario 1.

Table 11 presents the average energy consumption of fog nodes in Scenario 1, comparing the proposed algorithm against the other algorithms. The evaluation results demonstrate that the proposed VASLA-FSP algorithm reduces energy consumption by 9.49%, 24.5%, 17.18%, and 28.54% on average for service sets of 24, 48, 72, and 96, respectively, compared with DLA-FMP, DCSP, ODSPS, and WOA-FMP. Based on these values, Scenario 1 demonstrates that the proposed algorithm can improve energy efficiency by an average of 19.93%.

Table 11: Average energy consumption values across various algorithms in Scenario 1.

Number of fog nodes	Number of services	VASLA-FSP	DLA-FMP	DCSP	ODSPS	WOA-FMP	Percentage of improvement
FG(13, 6)	24	2451001	2620000	2581000	2713002	2821030	*9.49%*
	48	3445020	3637500	3968750	4661250	4887500	*24.5%*
	72	4990010	5285040	5957500	5849167	6295002	*17.18%*
	96	7125030	9002500	8486950	9397004	9722010	*28.54%*

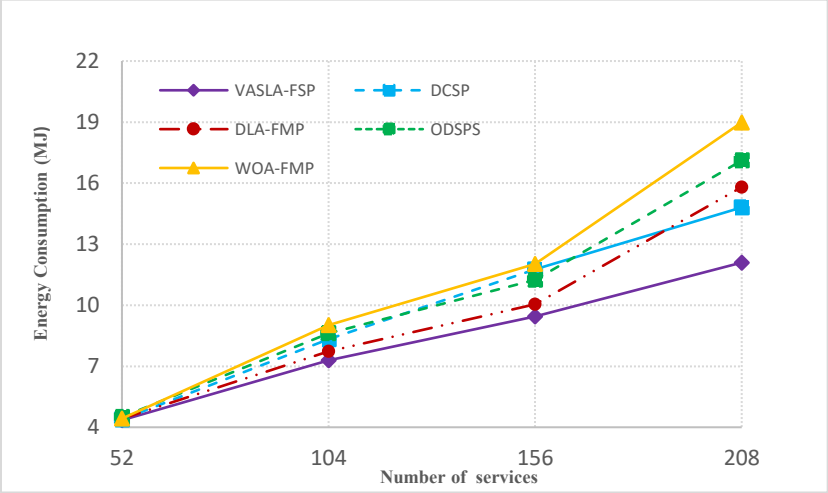


Fig. 12(b): Comparison of the energy consumption across various algorithms in Scenario 2.

Figure 12(b) illustrates the energy consumption in Scenario 2. As shown in the figure, energy consumption remains relatively stable across all 52 and 104 service methods, with each technique showing a proportional increase. However, for 156 and 208 services, the proposed approach significantly decreases energy usage compared to the other approaches. Table 12 presents the mean energy usage of fog nodes in Scenario 2, comparing the proposed algorithm with various algorithms. The obtained results demonstrate that the VASLA-FSP algorithm, under scenarios with 52, 104, 156, and 208 services, reduces energy consumption on average by 1.95%, 17.45%, 9.05%, and 37.86%, respectively, compared with DLA-FMP, DCSP, ODSPS, and WOA-FMP. Based on these values, the proposed method demonstrates an average improvement of 16.58% over the other techniques in Scenario 2.

Table 12: The average energy consumption of various algorithms under Scenario 2.

Topology	Number of services	VASLA-FSP	DLA-FMP	DCSP	ODSPS	WOA-FMP	Percentage of improvement
FG(28, 13)	52	4349114	4381126	4423092	4503552	*4428080*	1.95%
	104	7297917	8622548	8331086	8015356	9313605	17.45%
	156	9646720	10763970	10039080	10247160	11029130	9.05%
	208	12095523	14805392	15807074	17108964	18984655	37.86%

Figure 12 (c) shows the energy consumption results of various algorithms in Scenario 3 based on the number of IoT services. The results demonstrate that as the number of services increases, the proposed algorithm exhibits lower energy consumption than the other algorithms.

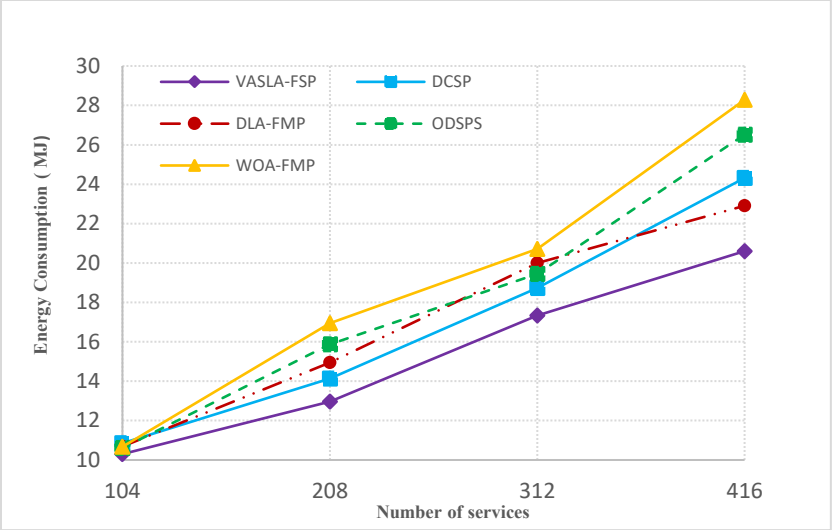


Fig. 12(c): Energy consumption analysis for various algorithms under Scenario 3.

Table13: The average energy consumption for various algorithms under Scenario 3.

Topology	Number of services	VASLA-FSP	DLA-FMP	DCSP	ODSPS	WOA-FSP	Percentage of improvement
$FG(57,26)$	104	11300200	11851732	11685220	11602000	11670540	3.56%
	208	13967720	14140151	15938320	16873840	17941400	16.09%
	312	18334890	19728570	21001420	20445680	21711260	13.0%
	416	21602060	25316990	23904520	27517520	29283120	22.7%

Table 13 presents the average energy consumption of fog nodes in Scenario 3, comparing the proposed algorithm with the other algorithms. According to the obtained results, the VASLA-FSP algorithm, when executed with 104, 208, 312, and 416 services, reduces energy consumption on average by 3.56%, 16.09%, 13.0%, and 22.7%, respectively, compared with DLA-FMP, DCSP, ODSPS, and WOA-FMP. Based on these values, the proposed method demonstrates an average improvement of 13.84% over the other techniques in Scenario 3. Evaluating energy consumption across the three scenarios reveals that the proposed algorithm achieved a 16.78% reduction in energy consumption compared to different approaches.

Experiment 2

Application loop delay refers to the total duration required for data to traverse through the independent service-oriented services of an IoT application and return the result to its source. The overall latency in various scenarios of the proposed algorithm, as shown in Figures 13(a), 13(b), and 13(c), is analyzed and compared with that of other existing algorithms. According to Figures 13(a), 13(b), and 13(c), the proposed VASLA-FSP algorithm exhibits the lowest latency among other algorithms, regardless of service size. The proposed method reduces communication latency by optimally placing application services at sources close to the network's edge. As a result, communication latency reduces due to the shorter transmission distance.

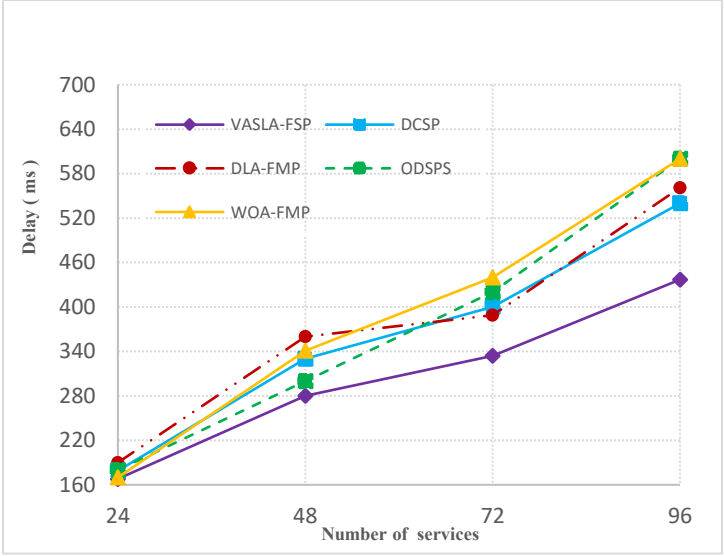


Fig. 13(a): Comparison of the delay for different algorithms in Scenario 1.

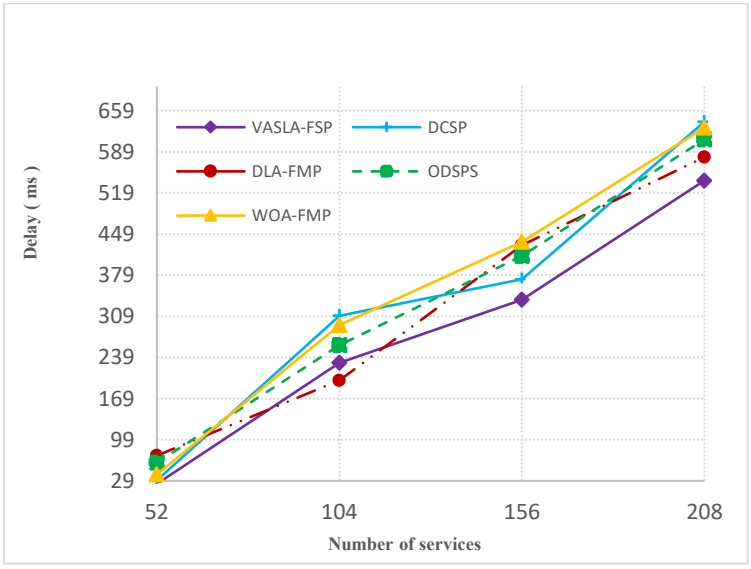


Fig. 13(b): Comparison of the delay for different algorithms in Scenario 2.

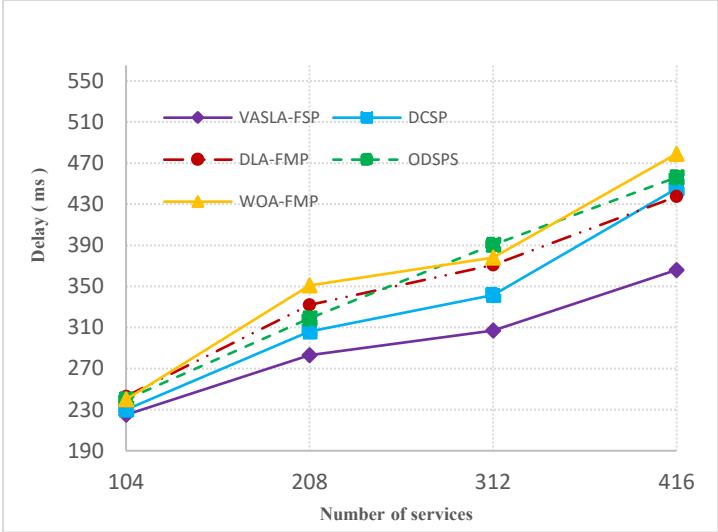


Fig. 13(c): Comparison of the delay for different algorithms in Scenario 3.

The average delays for executing services in networks of different sizes for Scenarios 1, 2, and 3 are shown in Tables 7, 8, and 9, respectively. According to these tables, under the $FG(13,6)$topology, the VASLA-FSP approach reduces the delay by up to 6.99%, 18.84%, 23.43%, and 32.77% compared to DLA-FMP, DCSP, ODSPS, and WOA-FMP, respectively. Under networks with the $FG(28,13)$ topology, the proposed approach decreases the delay by 26.14%, 15.65%, 22.4%, and 13.87% compared to DLA-FMP, DCSP, ODSPS, and WOA-FMP, respectively. Furthermore, in networks with the $FG(57,26)$ topology, the proposed approach decreases the delay by 5.98%, 22.25%, 19.27%, and 24.10% compared to DLA-FMP, DCSP, ODSPS, and WOA-FMP, respectively.

Table 7: The average delay in networks for different algorithms in Scenario 1.

Topology	The number of services	VASLA-FSP	DLA-FMP	DCSP	ODSPS	WOA-FMP	Percentage of improvement
$FG(13,6)$	24	168	179	190	180	170	6.99%
	48	280	330	360	300	341	18.84%
	72	334	400	389	420	440	23.43%
	96	437	540	561	600	620	32.77%

Table 8: The average delay in networks for different algorithms in Scenario 2.

Topology	The number of services	VASLA-FSP	DLA-FMP	DCSP	ODSPS	WOA-FMP	Percentage of improvement
$FG(28,13)$	52	24	30	42	57	64	26.14%
	104	230	310	200	260	294	15.65%
	156	337	372	425	412	436	22.4%
	208	540	640	580	610	632	13.87%

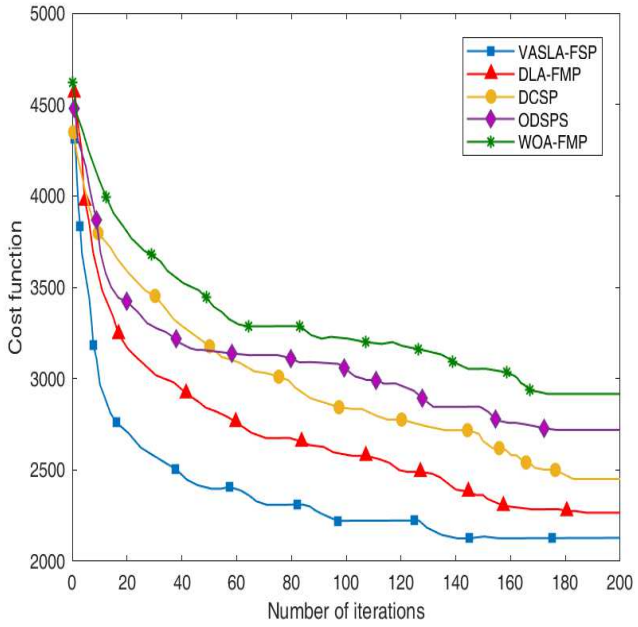
Table 9: The average delay in networks for different algorithms in Scenario 3.

Topology	The number of services	VASLA-FSP	DLA-FMP	DCSP	ODSPS	WOA-FMP	Percentage of improvement
$FG(57,26)$	104	225	230	243	240	245	5.98%
	208	283	306	332	319	351	22.25%
	312	307	342	371	391	378	19.27%
	416	366	445	438	456	479	24.10%

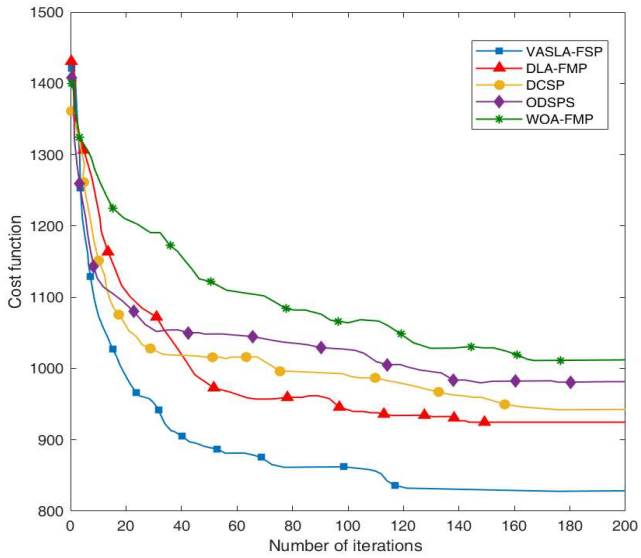
These results from different scenarios show that the proposed method improves the delay by 17.9% relative to existing approaches.

Experiment 3

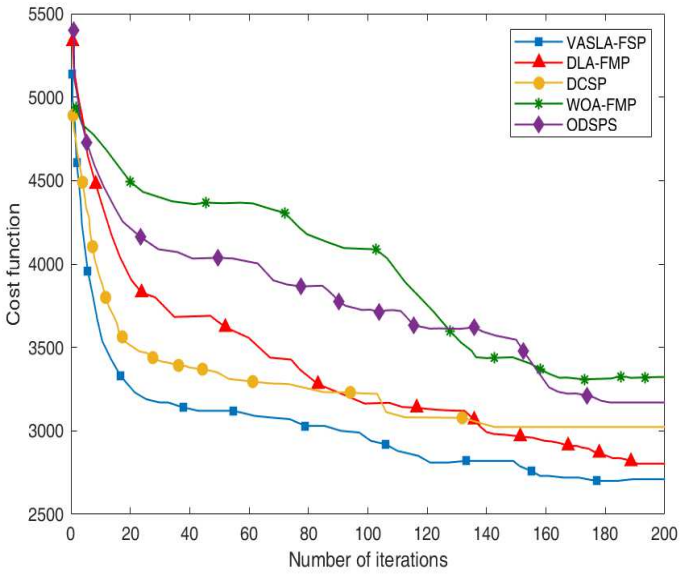
This experiment compares the cost functions of different algorithms to identify the optimal solution. In this experiment, Equation 11 is used as a cost function to compute the application loop latency and the total energy usage. Figure 14 illustrates the cost function values of the algorithms over 200 iterations for service sizes 24, 48, 104, 156, 312, and 416. As shown in Figure 14, the proposed method attained the minimum cost function value in fewer iterations for all scenarios. The comparison results indicate that the VASLA-FSP outperforms other algorithms in finding the optimal solution. This algorithm is superior because it does not explore the entire search space; it limits the search space, thereby increasing the convergence speed.



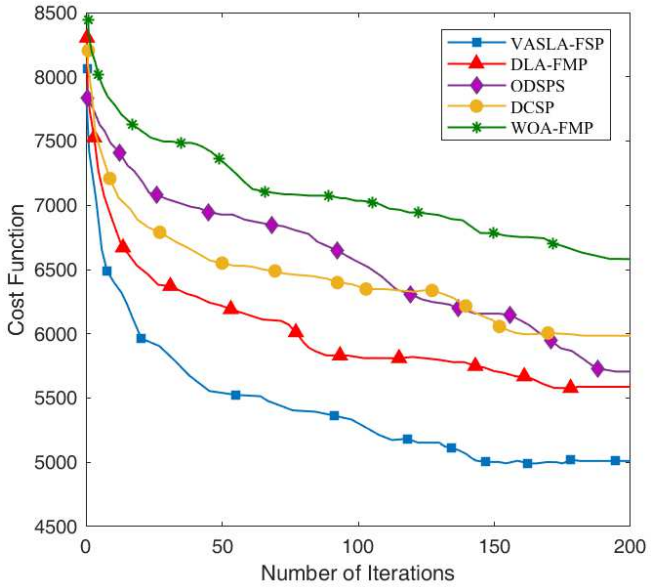
a: Convergence speed for 24 services



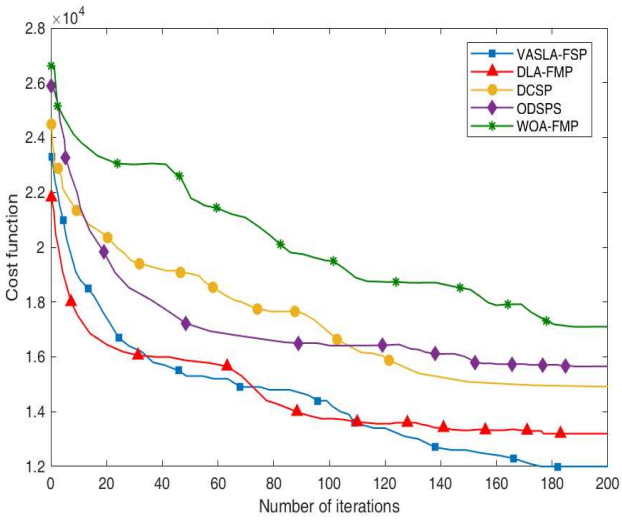
b: Convergence speed for 48 services



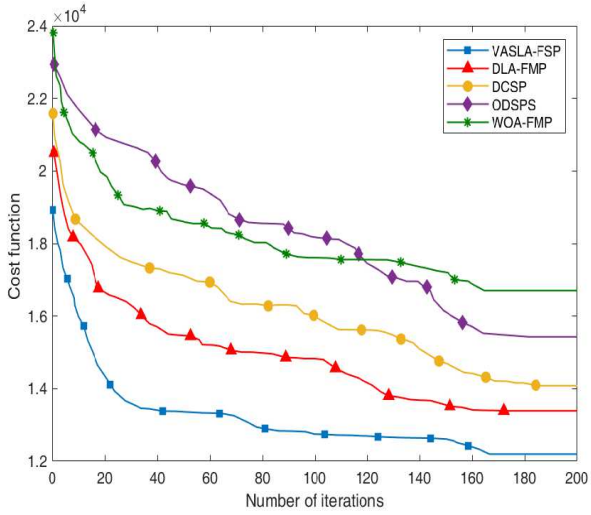
c: Convergence speed for 104 services



d: Convergence speed for 156 services



e: Convergence speed for 312 services



f: Convergence speed for 416 services

Fig. 14: Comparison of the cost function for different algorithms.

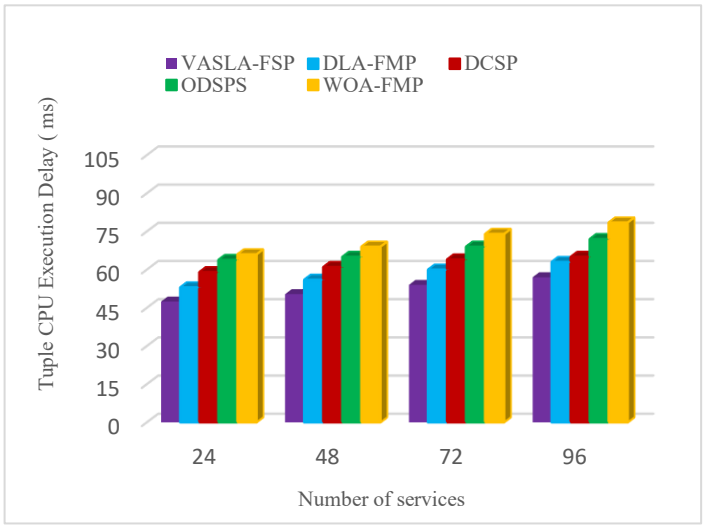
Table 10：Percentage decrease in cost function achieved by the proposed method relative to other methods.

Fog Graph	Application Name	ServiceSize	Number of Edge Fog Node	Total Service
FG(57, 26)	$\mathbf{Applicatin_1}$	4	26	104
	$\mathbf{Application_2}$	8	26	208
	$\mathbf{Appication_3}$	12	26	312
	$\mathbf{Appication_4}$	16	26	416

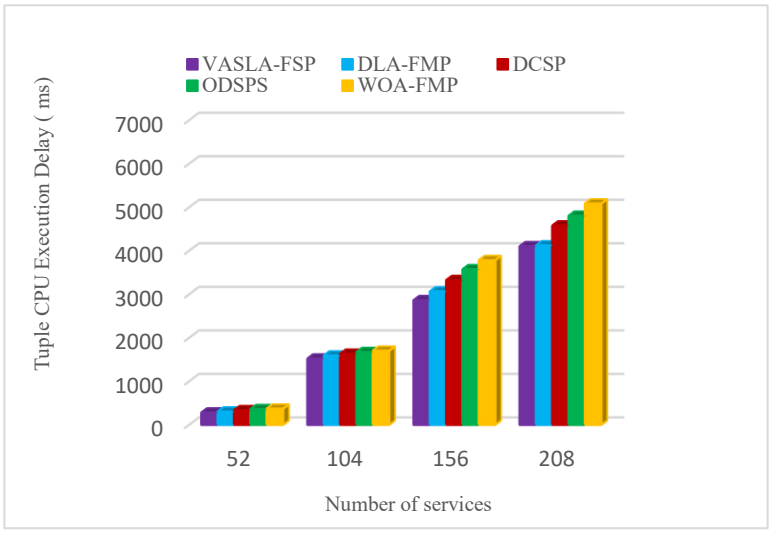
Table 10 shows the cost function results for the different algorithms. As shown, the proposed VASLA-FSP algorithm achieves improvements in convergence speed by up to 21.65%, 16.58%, 13.63%, 19.05%, 26.78%, and 22.16% for 24, 48, 104, 156, 312, and 416 services, respectively, compared to the DLA-FMP, DCSP, ODSPS, and WOA-FMP algorithms. Overall, these results indicate that the proposed method improves the cost function by an average of 19.98% compared to the other methods.

Experiment 4

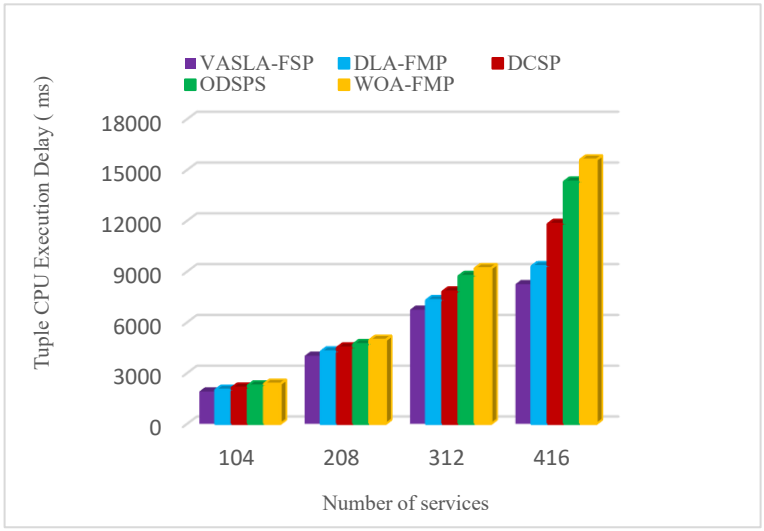
The computation of the tuple CPU execution delay is initiated when the sensor generates a tuple and continues until the system has produced all results. To illustrate how the algorithms perform under different service loads, the tuple CPU execution delay for each scenario is analyzed. The CPU delay for tuples with varying numbers of services is depicted in Fig. 15 under all three scenarios. We have noticed a significant increase in the execution latency of tuple processing. In scenarios 1, 2, and 3, the average tuple processing delays are 57.21 ms, 2369.25 ms, and 5429.25 ms, respectively. The results for different scenarios indicate that the proposed approach outperforms the other compared approaches. Because the proposed algorithm optimally places each service on fog nodes close to the data source, the tuple processing delay is reduced compared to other methods.



a: Tuple cup execution delay for scenario 1



b: Tuple cup execution delay for scenario 2

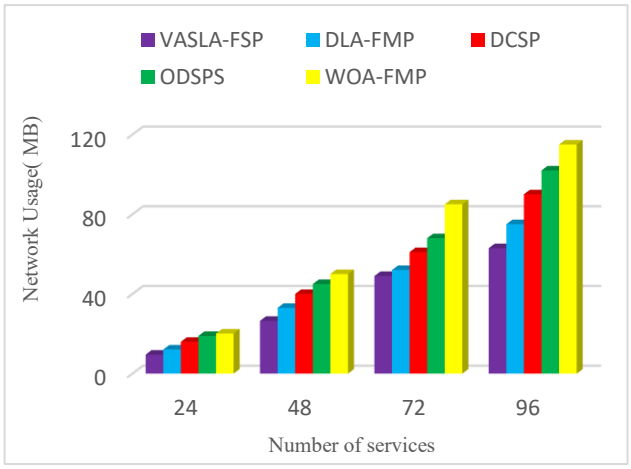


c: Tuple cup execution delay for scenario 3

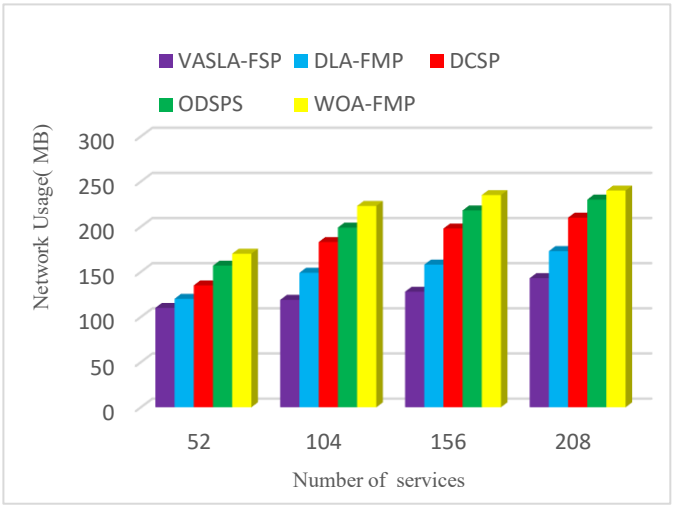
Fig. 15. Comparison of the tuple cup execution delay for different algorithms.

Experiment 5

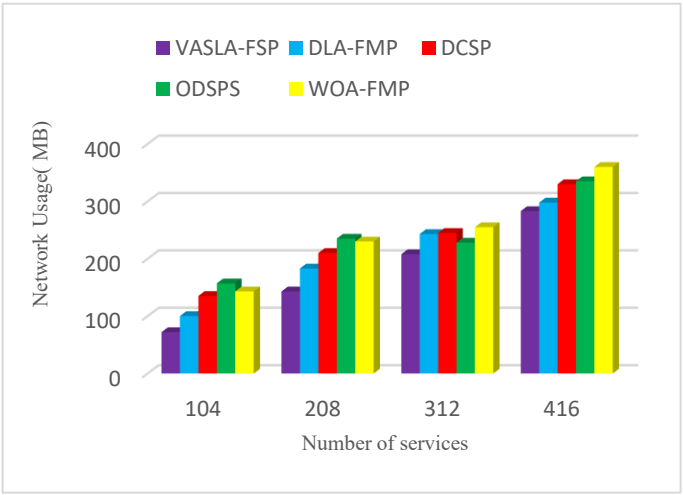
Figure 16 shows the network usage of various algorithms for scenarios 1, 2, and 3. As depicted, the proposed algorithm demonstrates lower network usage compared to other approaches when the number of application services increases. The proposed algorithm reduces network usage by optimally placing services on low-latency edge devices rather than on cloud servers. Therefore, this service placement efficiently minimizes the delay in establishing communication connections from source to destination nodes.



a: Network Usage for scenario 1



b: Network Usage for scenario 2

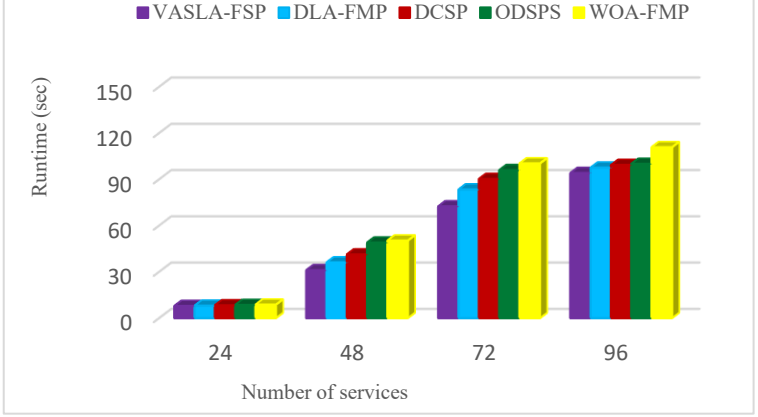


c: Network Usage for scenario 3

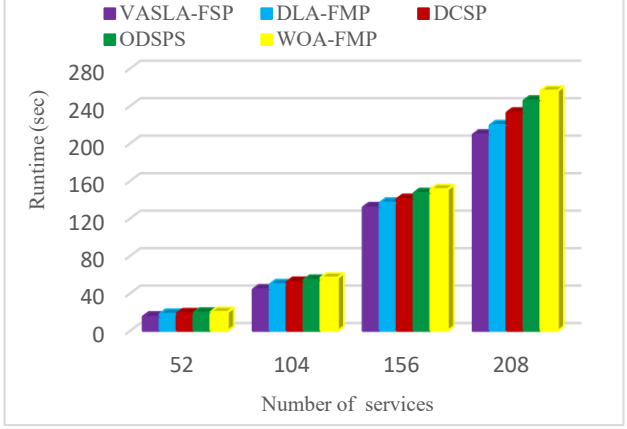
Fig. 16. Comparison of the network usage for different algorithms.

Experiment 6

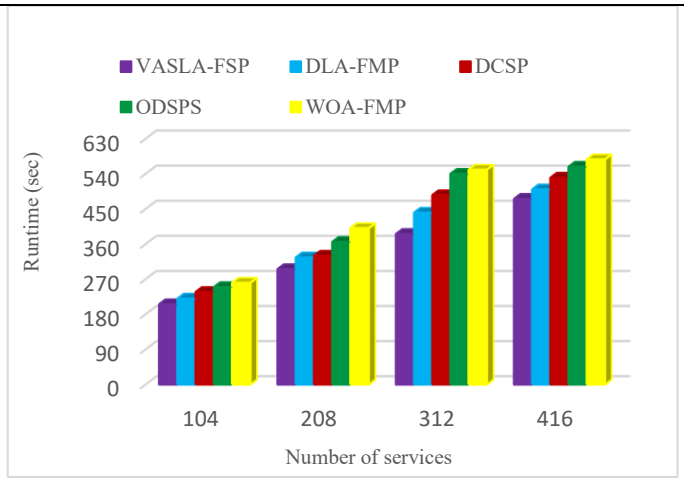
This experiment aims to identify an optimal service placement strategy that minimizes execution time. Figure 17 compares the total runtime of various service placement algorithms for scenarios 1, 2, and 3. Furthermore, increasing the number of IoT services deployed on the fog devices results in longer service placement and execution times. On average, the proposed method requires 58.09 seconds, 112.47 seconds, and 360.72 seconds to execute in scenarios 1, 2, and 3, respectively. As shown, the proposed VASLA-FSP algorithm achieves the lowest runtime compared to other methods.



a: Runtime for scenario 1



b: Runtime for scenario 2



c: Runtime for scenario 3

Figure 17: Comparison results of the runtime for different algorithms.

Experiment 7

In this section, we statistically analyzed the proposed VASLA-FSP algorithm using an unpaired t-test to measure the effectiveness of the proposed method compared to other algorithms. The statistical analysis of Scenario 1 is shown in Table 11. The objective function of all evaluation metrics in this study includes the cost function, energy consumption, and delay. The statistical features include the differences between p-value and t-value computed by comparing VASLA-FSP with other algorithms. In the experiment, we set the significance level α to 0.05. The p-value $p \leq 0.05$ indicates that the statistical test rejects the null hypothesis (H_0) at a 95% confidence level. A p-value of ≤ 0.05 suggests that the proposed algorithm is more energy-efficient than the other approaches, resulting in reduced delays and optimized cost function, as shown in Table 11.

Table 11. Statistical evaluation of the proposed VASLA-FSP algorithm in scenario 1 with 15 fog nodes.

Metric	Algorithms	95% Confidence Interval (CI)	t-statistic	p-value	Significance
Energy Consumption	DLA-FMP: VASLA-FSP	-1955992.02 to 689002.52	1.5244	0.2248	not statistically significant
	DCSP: VASLA-FSP	-1596395.06 to 104825.56	2.7903	0.0684	not quite statistically significant.
	ODSPS: VASLA-FSP	-2495116.59 to 190435.59	2.7311	0.0719	not quite statistically significant
	WOA-FMP: VASLA-FSP	-2881536.66 to 24296.16	3.1292	0.0521	not quite statistically significant
Delay	DLA-FMP: VASLA-FSP	-118.17 to 3.17	3.0162	0.0569	not quite statistically significant
	DCSP: VASLA-FSP	-138.66 to -1.84	3.2681	0.0468	statistically significant
	ODSPS: VASLA-FSP	-190.50 to 45.50	1.9552	0.1455	not statistically significant
	WOA-FMP: VASLA-FSP	-209.44 to 33.44	2.3061	0.1044	not statistically significant
Cost function	DLA-FMP: VASLA-FSP	-383.0904 to 148.7904	5.5972	0.1126	not statistically significant
	DCSP: VASLA-FSP	-1525.9368 to 1087.6068	2.1310	0.2793	not statistically significant
	ODSPS: VASLA-FSP	-3181.0825 to 2432.7825	1.6937	0.3395	not statistically significant
	WOA-FMP: VASLA-FSP	-4324.1242 to 3351.8342	1.6095	0.3539	not statistically significant

6. Conclusions

In this paper, we introduce a fog service placement approach called VASLA-FSP, based on Variable Action-Set Learning Automata (VASLA) for dynamic IoT environments. The proposed approach models the fog topology as a graph and maps it into a VASLA to efficiently identify optimal service placements. Unlike conventional static methods, which assume fixed fog nodes, communication links, and service requirements, VASLA-FSP operates effectively in dynamic environments where these conditions may all change during execution. In the proposed approach, only a subset of fog nodes is searched, thereby significantly reducing service placement delay and computational overhead. Simulation results across multiple scenarios confirm that VASLA-FSP outperforms existing methods, reducing energy consumption by 17%, delay by 18%, and improving the overall cost function by 20%. These results confirm that VASLA-FSP can efficiently use fog resources, even in dynamic IoT environments. However, in this study, user demand is assumed to be stable, with little or no mobility, and sudden demand surges are not considered. Additionally, the evaluations were conducted in simulation under predefined dynamics that may not fully

account for the complexities of real-world deployments. In the future, we intend to generalize VASLA-FSP to support user movement and time-varying service requests. Furthermore, we aim to develop more scalable and adaptive mechanisms to maintain low latency in large-scale, highly dynamic fog networks and demonstrate that VASLA-FSP is effective through real-world tests.

Refrences

[1] R. Minerva, A. Biru, and D. Rotondi, "Towards a definition of the Internet of Things (IoT)," *IEEE Internet Initiative,* vol. 1, no. 1, pp. 1-86, 2015.

[2] M. Mansour *et al.*, "Internet of things: A comprehensive overview on protocols, architectures, technologies, simulation tools, and future directions," *Energies,* vol. 16, no. 8, p. 3465, 2023.

[3] M. Ghobaei-Arani, "A workload clustering based resource provisioning mechanism using Biogeography based optimization technique in the cloud based systems," *Soft Computing,* vol. 25, no. 5, pp. 3813-3830, 2021.

[4] A. Rezaeipanah, M. Mojarad, and A. Fakhari, "Providing a new approach to increase fault tolerance in cloud computing using fuzzy logic," *International Journal of Computers and Applications,* vol. 44, no. 2, pp. 139-147, 2022.

[5] N. Wang, B. Varghese, M. Matthaiou, and D. S. Nikolopoulos, "ENORM: A framework for edge node resource management," *IEEE transactions on services computing,* vol. 13, no. 6, pp. 1086-1099, 2017.

[6] R. Mahmud, S. N. Srirama, K. Ramamohanarao, and R. Buyya, "Quality of Experience (QoE)-aware placement of applications in Fog computing environments," *Journal of Parallel and Distributed Computing,* vol. 132, pp. 190-203, 2019.

[7] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, "Fog computing: A platform for internet of things and analytics," in *Big data and internet of things: A roadmap for smart environments*: Springer, 2014, pp. 169-186.

[8] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, 2012, pp. 13-16.

[9] A. V. Dastjerdi, H. Gupta, R. N. Calheiros, S. K. Ghosh, and R. Buyya, "Fog computing: Principles, architectures, and applications," in *Internet of things*: Elsevier, 2016, pp. 61-75.

[10] A. M. Rahmani, P. Liljeberg, J.-S. Preden, and A. Jantsch, "Fog computing in the internet of things," *Switzerland: Springer International Publishing,* pp. 3-13, 2018.

[11] F. Li, M. Vögler, M. Claeßens, and S. Dustdar, "Efficient and scalable IoT service delivery on cloud," in *2013 IEEE sixth international conference on cloud computing*, 2013: IEEE, pp. 740-747.

[12] S. Sarkar and S. Misra, "Theoretical modelling of fog computing: a green computing paradigm to support IoT applications," *Iet Networks,* vol. 5, no. 2, pp. 23-29, 2016.

[13] I. Stojmenovic, "Fog computing: A cloud to the ground support for smart things and machine-to-machine networks," in *2014 Australasian telecommunication networks and applications conference (ATNAC)*, 2014: IEEE, pp. 117-122.

[14] M. Yannuzzi, R. Milito, R. Serral-Gracià, D. Montero, and M. Nemirovsky, "Key ingredients in an IoT recipe: Fog Computing, Cloud computing, and more Fog Computing," in *2014 IEEE 19th international workshop on computer aided modeling and design of communication links and networks (CAMAD)*, 2014: IEEE, pp. 325-329.

[15] F. Karatas and I. Korpeoglu, "Fog-based data distribution service (F-DAD) for internet of things (IoT) applications," *Future Generation Computer Systems,* vol. 93, pp. 156-169, 2019.

[16] Y. Abofathi, B. Anari, and M. Masdari, "A learning automata based approach for module placement in fog computing environment," *Expert Systems with Applications,* vol. 237, p. 121607, 2024.

[17] T. Djemai, P. Stolf, T. Monteil, and J.-M. Pierson, "A discrete particle swarm optimization approach for energy-efficient IoT services placement over fog infrastructures," in *2019 18th international symposium on parallel and distributed computing (ISPDC)*, 2019: IEEE, pp. 32-40.

[18] N. Mehran, D. Kimovski, and R. Prodan, "MAPO: a multi-objective model for IoT application placement in a fog environment," in *Proceedings of the 9th International Conference on the Internet of Things*, 2019, pp. 1-8.

[19] I. Lera, C. Guerrero, and C. Juiz, "Analyzing the applicability of a multi-criteria decision method in fog computing placement problem," in *2019 Fourth International Conference on Fog and Mobile Edge Computing (FMEC)*, 2019: IEEE, pp. 13-20.

[20] B. Natesha and R. M. R. Guddeti, "Meta-heuristic based hybrid service placement strategies for two-level fog computing architecture," *Journal of Network and Systems Management,* vol. 30, no. 3, p. 47, 2022.

[21] M. Dadashi Gavaber and A. Rajabzadeh, "MFP: An approach to delay and energy-efficient module placement in IoT applications based on multi-fog," *Journal of Ambient Intelligence and Humanized Computing,* vol. 12, no. 7, pp. 7965-7981, 2021.

[22] A. M. Maia, Y. Ghamri-Doudane, D. Vieira, and M. F. de Castro, "An improved multi-objective genetic algorithm with heuristic initialization for service placement and load distribution in edge computing," *Computer networks,* vol. 194, p. 108146, 2021.

[23] Y. Qian, L. Hu, J. Chen, X. Guan, M. M. Hassan, and A. Alelaiwi, "Privacy-aware service placement for mobile edge computing via federated learning," *Information Sciences,* vol. 505, pp. 562-570, 2019.

[24] R. Oma, S. Nakamura, D. Duolikun, T. Enokido, and M. Takizawa, "An energy-efficient model for fog computing in the internet of things (IoT)," *Internet of Things,* vol. 1, pp. 14-26, 2018.

[25] A. Yousefpour, G. Ishigaki, R. Gour, and J. P. Jue, "On reducing IoT service delay via fog offloading," *IEEE Internet of things Journal,* vol. 5, no. 2, pp. 998-1010, 2018.

[26] M. M. Mahmoud, J. J. Rodrigues, K. Saleem, J. Al-Muhtadi, N. Kumar, and V. Korotaev, "Towards energy-aware fog-enabled cloud of things for healthcare," *Computers & Electrical Engineering,* vol. 67, pp. 58-69, 2018.

[27] T. Huang, W. Lin, C. Xiong, R. Pan, and J. Huang, "An ant colony optimization-based multiobjective service replicas placement strategy for fog computing," *IEEE Transactions on Cybernetics,* vol. 51, no. 11, pp. 5595-5608, 2020.

[28] M. Ghobaei-Arani and A. Shahidinejad, "A cost-efficient IoT service placement approach using whale optimization algorithm in fog computing environment," *Expert Systems with Applications,* vol. 200, p. 117012, 2022.

[29] R. Roy, S. Dehuri, and S. B. Cho, "A novel particle swarm optimization algorithm for multi-objective combinatorial optimization problem," *International Journal of Applied Metaheuristic Computing (IJAMC),* vol. 2, no. 4, pp. 41-57, 2011.

[30] N. Sarrafzade, R. Entezari-Maleki, and L. Sousa, "A genetic-based approach for service placement in fog computing," *The Journal of Supercomputing,* vol. 78, no. 8, pp. 10854-10875, 2022.

[31] C. Liu, J. Wang, L. Zhou, and A. Rezaeipanah, "Solving the multi-objective problem of IoT service placement in fog computing using cuckoo search algorithm," *Neural Processing Letters,* vol. 54, no. 3, pp. 1823-1854, 2022.

[32] M. Salimian, M. Ghobaei-Arani, and A. Shahidinejad, "Toward an autonomic approach for Internet of Things service placement using gray wolf optimization in the fog computing environment," *Software: Practice and Experience,* vol. 51, no. 8, pp. 1745-1772, 2021.

[33] J. Taghizadeh, M. Ghobaei-Arani, and A. Shahidinejad, "A metaheuristic-based data replica placement approach for data-intensive IoT applications in the fog computing environment," *Software: Practice and Experience,* vol. 52, no. 2, pp. 482-505, 2022.

[34] M. Tekiyehband, M. Ghobaei-Arani, and A. Shahidinejad, "An efficient dynamic service provisioning mechanism in fog computing environment: A learning automata approach," *Expert Systems with Applications,* vol. 198, p. 116863, 2022.

[35] O. Skarlat, M. Nardelli, S. Schulte, M. Borkowski, and P. Leitner, "Optimized IoT service placement in the fog," *Service Oriented Computing and Applications,* vol. 11, no. 4, pp. 427-443, 2017.

[36] C. Guerrero, I. Lera, and C. Juiz, "Evaluation and efficiency comparison of evolutionary algorithms for service placement optimization in fog architectures," *Future Generation Computer Systems,* vol. 97, pp. 131-144, 2019.

[37] C. Canali and R. Lancellotti, "GASP: genetic algorithms for service placement in fog computing systems," *Algorithms,* vol. 12, no. 10, p. 201, 2019.

[38] C. Jian, L. Bao, and M. Zhang, "A high-efficiency learning model for virtual machine placement in mobile edge computing," *Cluster Computing,* vol. 25, no. 5, pp. 3051-3066, 2022.

[39] M. Ayoubi, M. Ramezanpour, and R. Khorsand, "An autonomous IoT service placement methodology in fog computing," *Software: Practice and Experience,* vol. 51, no. 5, pp. 1097-1120, 2021.

[40] M. Zare, Y. E. Sola, and H. Hasanpour, "Towards distributed and autonomous IoT service placement in fog computing using asynchronous advantage actor-critic algorithm," *Journal of King Saud University-Computer and Information Sciences,* vol. 35, no. 1, pp. 368-381, 2023.

[41] I. Lera and C. Guerrero, "Multi-objective application placement in fog computing using graph neural network-based reinforcement learning," *The Journal of Supercomputing,* vol. 80, no. 19, pp. 27073-27094, 2024.

[42] B. Wu, X. Lv, W. D. Shamsi, and E. G. Dizicheh, "Optimal deploying IoT services on the fog computing: A metaheuristic-based multi-objective approach," *Journal of King Saud University-Computer and Information Sciences,* vol. 34, no. 10, pp. 10010-10027, 2022.

[43] M. Ranjbari and J. A. Torkestani, "A learning automata-based algorithm for energy and SLA efficient consolidation of virtual machines in cloud data centers," *Journal of Parallel and Distributed Computing,* vol. 113, pp. 55-62, 2018.

[44] V. Yadav, B. Natesha, and R. M. R. Guddeti, "Ga-pso: Service allocation in fog computing environment using hybrid bio-inspired algorithm," in *TENCON 2019-2019 IEEE Region 10 Conference (TENCON)*, 2019: IEEE, pp. 1280-1285.

[45] M. Dadashi Gavaber and A. Rajabzadeh, "MFP: An approach to delay and energy-efficient module placement in IoT applications based on multi-fog," *Journal of Ambient Intelligence and Humanized Computing,* vol. 12, pp. 7965-7981, 2021.

[46] F. Khosroabadi, F. Fotouhi-Ghazvini, and H. Fotouhi, "Scatter: Service placement in real-time fog-assisted iot networks," *Journal of Sensor and Actuator Networks,* vol. 10, no. 2, p. 26, 2021.

[47] M. Azimzadeh, A. Rezaee, S. J. Jassbi, and M. Esnaashari, "Placement of IoT services in fog environment based on complex network features: a genetic-based approach," *Cluster Computing,* vol. 25, no. 5, pp. 3423-3445, 2022.

[48] K. S. Narendra and M. A. Thathachar, *Learning automata: an introduction*. Courier corporation, 2012.

[49] M. Thathachar and P. S. Sastry, "A hierarchical system of learning automata that can learn die globally optimal path," *Information sciences,* vol. 42, no. 2, pp. 143-166, 1987.

[50] K. S. Narendra and M. A. Thathachar, "On the behavior of a learning automaton in a changing environment with application to telephone traffic routing," *IEEE Transactions on Systems, Man, and Cybernetics,* vol. 10, no. 5, pp. 262-269, 2007.

[51] B. Anari, J. A. Torkestani, and A. M. Rahmani, "Automatic data clustering using continuous action-set learning automata and its application in segmentation of images," *Applied Soft Computing,* vol. 51, pp. 253-265, 2017.

[52] F. Jazayeri, A. Shahidinejad, and M. Ghobaei-Arani, "Autonomous computation offloading and auto-scaling the in the mobile fog computing: a deep reinforcement learning-based approach," *Journal of Ambient Intelligence and Humanized Computing,* vol. 12, no. 8, pp. 8265-8284, 2021.

[53] Z. Anari, A. Hatamlou, and B. Anari, "Automatic finding trapezoidal membership functions in mining fuzzy association rules based on learning automata," 2022.

[54] M. Khaksar Manshad, M. R. Meybodi, and A. Salajegheh, "A new irregular cellular learning automata-based evolutionary computation for time series link prediction in social networks," *Applied Intelligence,* vol. 51, no. 1, pp. 71-84, 2021.

[55] A. B. Hashemi and M. R. Meybodi, "A note on the learning automata based algorithms for adaptive parameter selection in PSO," *Applied Soft Computing,* vol. 11, no. 1, pp. 689-705, 2011.

[56] A. Yazidi, D. Silvestre, and B. J. Oommen, "Solving two-person zero-sum stochastic games with incomplete information using learning automata with artificial barriers," *IEEE Transactions on Neural Networks and Learning Systems,* vol. 34, no. 2, pp. 650-661, 2021.

[57] A. Yazidi, I. Hassan, H. L. Hammer, and B. J. Oommen, "Achieving fair load balancing by invoking a learning automata-based two-time-scale separation paradigm," *IEEE Transactions on Neural Networks and Learning Systems,* vol. 32, no. 8, pp. 3444-3457, 2020.

[58] M. Thathachar and B. R. Harita, "Learning automata with changing number of actions," *IEEE transactions on systems, man, and cybernetics,* vol. 17, no. 6, pp. 1095-1100, 1987.

[59] U. Arora and N. Singh, "IoT application modules placement in heterogeneous fog–cloud infrastructure," *International Journal of Information Technology,* vol. 13, no. 5, pp. 1975-1982, 2021.