J Grid Computing (2022) 20: 12
https://doi.org/10.1007/s10723-022-09604-3

A QoS-Aware IoT Service Placement Mechanism in Fog Computing Based on Open-Source Development Model

Defu Zhao · Qunying Zou · Milad Boshkani Zadeh

Received: 10 June 2021 / Accepted: 18 March 2022 / Published online: 11 April 2022

Abstract With rapid developments of the Internet of Things (IoT) applications in recent years, their use to facilitate day-to-day activities in various domains for enhancing the quality of human life has significantly increased. Fog computing has been developed to overcome the limitations of cloud-based networks and to address the challenges posed by the massive growth of IoT devices. This paradigm can provide better Quality of Service (QoS) in terms of low energy consumption and fast response, and cope with latency and bandwidth limitations. Since IoT applications are offered in the form of multiple IoT services with different QoS requirements, it is essential to develop an efficient IoT service deployment mechanism in a fog environment with distributed fog nodes and centralized fog servers. This is referred to as the Fog Services Placement (FSP) problem. Hence, we propose a QoS-aware IoT services placement policy with different objectives as a multi-objective optimization problem. Given the proven effectiveness of meta-heuristic techniques in solving optimization problems, we have used the Open-source Development Model Algorithm (ODMA) to deploy IoT services on fog nodes called FSP-ODMA. FSP-ODMA uses the service cost, energy consumption, response time, latency, and fog resource utilization as objective functions to find the optimal IoT service placement plan. In addition, we propose a three-layer conceptual computing framework (i.e., cloud-fog-IoT) to describe the interactions between system components and the FSP problem-solving policy. The simulation results obtained demonstrate that the proposed solution increases the resource usage and service acceptance ratio and reduces the service delay and the energy consumption compared with the other metaheuristic-based mechanisms.

Keywords Fog computing · IoT applications · Service placement · ODMA optimization algorithm

D. Zhao (✉)
School of Date Science, Jiangxi Institute of Fashion Technology, Nanchang, Jiangxi 330201, China
e-mail: aolanzhou@163.com

Q. Zou
Business School, Jiangxi Institute of Fashion Technology, Nanchang, Jiangxi 330201, China
e-mail: zqywstl@163.com

M. Boshkani Zadeh
Department of Computer Engineering, Tangestan (Ahram) Branch, Islamic Azad University, Ahram, Iran
e-mail: mboshkanizade@gmail.com

1 Introduction

The Internet of Things (IoT) emerged as a concept in the early 2000s and is now making headlines around the world [1]. IoT refers to the billions of devices that are connected to the Internet around the world and have the ability to collect and exchange data with each other. These devices are used in various fields such as industry, smart homes, smart agriculture, traffic monitoring, animal tracking, etc. [2]. The number of devices connecting to the Internet is increasing every day. The number of IoT devices will reach more than 46 billion in 2021 and is projected to increase to more than 75 billion by 2025 [1]. These huge volumes of IoT devices generate a lot of data that requires a lot of cost, energy and memory to process in the endpoints. Processing this growing amount of data poses a major challenge to society.

Cloud computing technology can provide IoT device services to end users without worrying about storage, processing power and high costs [3]. In this regard, data is collected by IoT devices and transferred to cloud servers, and then activities such as data storage and processing are performed in the cloud environment instead of the devices themselves [4]. In the last one year, with the increase of telecommuting and the need to share data due to the start of the Corona pandemic, the use of cloud-based software is expanding.

Many IoT applications are based on real-time data processing and experience high latency when interacting with centralized cloud servers. This problem is due to the large distance between IoT devices and cloud servers. In addition, the massive growth of IoT devices has led to the cloud facing huge volumes of data for processing. Therefore, the current cloud infrastructure suffers from the problem of bandwidth, latency, response time, and network congestion. Hence, a new model called fog computing has evolved to overcome these limitations [5–7]. Fog computing is a distributed computing model that stores, calculates, and processes data generated by end users near IoT devices. Therefore, this paradigm provides cloud services at the edge of the network to reduce the latency associated with real-time applications. In addition to reducing latency, fog computing can reduce bandwidth through reducing the amount of data transmitted to the cloud. In general, fog computing provides better Quality of Service (QoS) in terms of latency, response time, bandwidth, network congestion, and energy consumption [4]. The main components of fog computing are IoT devices (service requesters) and fog nodes (service providers) [6].

Fog computing is considered complementary to cloud computing instead of replacement. There are a large number of heterogeneous nodes in fog computing (i.e., gateways, routers, set-top-boxes, access points, switches, etc.) that allow IoT applications (i.e., smartphones, vehicles, smartwatches, sensors, cameras, etc.) to execution near end devices in a decentralized manner without involving the cloud. However, fog nodes have limited processing power, storage capacity and memory compared to cloud servers [9]. Therefore, resource management in fog computing is challenging for reasons such as decentralization, heterogeneity, dynamics, and variability, and requires efficient orchestration policies. Orchestration involves activities to effectively manage resources in fog-cloud computing. In the discussion of resource management, there is a wide range of issues such as resource forecasting, task scheduling, service migration and service location, which we focused on the issue of services placement.

The problem of resource provisioning, also known as Fog Placement Services (FSP), refers to how autonomous deployment of IoT applications on fog nodes. Basically, IoT applications are developed based on a microservice architecture that can better harness the potential of distributed fog computing. Therefore, IoT applications include several IoT services with different QoS requirements that can be deployed on fog nodes with different resource capabilities [11]. FSP is an NP-Hard problem in which the state of resources in fog and the details of IoT services must be considered simultaneously [12]. In general, the FSP decides on the mapping between IoT services and fog nodes in order to improve QoS metrics (i.e., reliability, latency, response time, cost, availability, fog resource utilization, energy) [13]. Some characteristics of the fog environment such as decentralized, heterogeneity, limited storage resources and processing can add to the complexity of solving this problem.

In this paper, we propose a framework for QoS-aware IoT services placement in fog computing to solve the FSP problem. This framework formulates the FSP problem based on the autonomous MADE-k (monitoring, analysis, decision-making, and execution control loop with a shared knowledge base) model [14]. According to MADE-k, fog sources and IoT services are monitored first. Prioritization of services is done based on the deadline in the analysis phase. The services placement is determined at the decision-making phase and finally the decisions made are executed in the fog environment. In this paper, the FSP problem is solved in the decision-making phase of the MADE-k model by the Open-source Development Model Algorithm (ODMA) [15] as a meta-heuristic algorithm called FSP-ODMA. ODMA as an evolutionary algorithm focuses on solving a wide range of optimization problems and provides high convergence speed and local optimization avoidance during the search process compared to other evolutionary algorithms (i.e., Genetic Algorithm (GA), Particle Swarm Optimization (PSO), etc.). FSP-ODMA decides on the IoT services placement by compromising between different objectives such as service cost, energy consumption, response time, latency, and the fog resource utilization. Multi-objective FSP problem solving can improve the performance of deploying IoT services on fog nodes.

The main contributions to this work can be summarized as follows:

- Design a conceptual framework based on the autonomous MADE-k model to demonstrate the interaction between the main components of the system (i.e., IoT devices, fog nodes and cloud servers) and efficient resource management
- Development of ODMA meta-heuristic algorithm to solve FSP problem in multi-objective form
- Evaluation of the proposed algorithm by simulating fog environment in terms of various performance metrics such as service cost, energy consumption, response time, latency and fog resources utilization

The rest of the paper is organized as follows: Section 2 discusses a literature review related to the FSP problem. Section 3 introduces the ODMA meta-heuristic algorithm. Section 4 presents the proposed conceptual framework based on the autonomous MADE-k model. The proposed placement scheme for solving the FSP problem using ODMA is described in Section 5. The discussion of simulation is given in Section 6. Finally, Section 7 concludes this paper.

2 Related Works

Many studies have addressed the FSP problem with different methodologies to improve processing, load balance, energy consumption, data transfer speed and storage [24–30]. According to the studied studies, FSP problem solving methodologies including approaches based on graph partitioning [7], graph coloring [13], greedy algorithm [24], First Come First Served (FCFS) algorithm [25], Quantum-based approaches [26], mathematical model-based approaches [27], fuzzy logic-based approaches [28], blockchain-based approaches [29] and techniques based on artificial intelligence [30]. Because artificial intelligence techniques are more effective than other techniques [31], we examine the literature from this perspective. In general, solving FSP based on artificial intelligence techniques includes evolutionary algorithms, machine learning algorithms and hybrid algorithms [31].

Given the widespread acceptance of evolutionary algorithms for solving FSP as well as its use in this work, we discuss related studies in this area below. Common evolutionary algorithms used to solve FSP include GA, Ant Colony Optimization (ACO), PSO, Artificial Bee Colony (ABC), Memetic Algorithm (MA), Differential Evolution (DE), and Taboo Search (TS). According to evolutionary algorithms, authors try to optimize one or more performance metrics based on a set of limitations to perform the process of deploying applications or services. Performance metrics and problem limitations can be divided into five general categories including resources (resource utilization), energy (energy consumption), network (load scheduling, failure rate, reliability, load balance and bandwidth), cost (communications, Virtual Machine (VM), migration, execution, deployment and resource usage) and time (scheduling, processing, placement, latency, completion time, deployment, propagation, deadline, waiting, round trip, task execution, computation, response time) [31].

Hussein and Mousa (2020) used PSO and ACO algorithms to solve the FSP problem and improve IoT task scheduling [30]. The authors considered the service rate and network latency as the objective function and managed to significantly improve the load balance of the fog nodes and the response time of IoT services. Huang et al. (2020) introduced the Multi-Replicas Pareto Ant Colony Optimization (MRPACO) to solve the problem of service replicas placement in fog environment [32]. MRPACO formulates the problem as a multi-objective problem considering services latency and deployment costs. The results show that by choosing the flow direction greedily, better quality solutions are obtained. Gill and Singh (2020) solved the problem of container placement on VMs in fog computing using ACO [33]. Optimal placement of containers in fog as a lightweight preference can reduce migration time and improve response time. The authors solve this problem by minimizing makespan and guarantee the QoS to end users.

Ghalehtaki et al. (2019) proposed an approach to solve the problem of micro-cache placement close to end users on fog-based Content Delivery Networks (CDNs) [34]. The authors formulated this problem on Set-Top Boxes (STBs) as an optimization problem and solved it with ABC algorithm to improve QoS in terms of cost and latency. Sharma and Saini (2019) proposed a cost-effective approach that can balance load in a fog environment by prioritizing user demand [35]. Here, the prioritization process is done with ABC based on the fitness function. The authors considered various QoS-related objectives (such as VM schedule length, schedule runtime, and energy consumption) in the fitness function. Nabavi et al. (2022) proposed an ABC-based multi-objective deployment approach for VMs placement in edge-cloud data centers [36]. This approach is called TRACTOR. TRACTOR is an energy-efficient and traffic-aware approach that considers VMs as fog tasks. The simulation results show that TRACTOR can significantly reduce energy consumption and network traffic without affecting other QoS parameters.

Javanmardi et al. (2021) proposed fog task scheduling for mobile IoT devices by combining PSO and fuzzy logic algorithms [37]. The authors simultaneously use network utilization and application loop latency to manage resources. Here, PSO is used for placement work and fuzzy logic is used to calculate the fitness function. Djemai et al. (2019) used Discrete PSO (DPSO) to solve the FSP problem [38]. DPSO provides suitable computing capacity for IoT services in the cloud-fog-IoT ecosystem by considering latency and energy consumption. In addition, the authors proposed an architecture of infrastructures and IoT applications in fog computing to illustrate the interactions between system components. Baburao et al. (2021) proposed a PSO-based Enhanced Dynamic Resource Allocation Method (EDRAM) for load balancing in fog computing [39]. EDRAM improves resource management in fog by taking into account bandwidth, latency, and makespan, as well as eliminating sleepy, unreferred, and inactive services.

Reddy et al. (2020) formulated resource management in a fog environment as an optimization problem and solved it by combining GA and Reinforcement Learning (RL) [40]. GA is applied to deploy IoT services on the minimum number of fog nodes and RL to optimize the period of fog nodes duty cycle by predicting sleep-wake cycles. The goal of this method is to manage resources in fog by minimizing latency and energy consumption for context-aware smart cities. Maia et al. (2021) formulated the FSP as a multi-objective problem by considering the distribute workloads of IoT applications [41]. The authors proposed an improved GA to solve the problem, in which the initial population is created to overcome the complexity with random-heuristic solutions. The objectives of this approach include Service Level Agreement (SLA) violations, availability of services, operational costs, and response time deadline violation. Bourhim et al. (2019) proposed a GA-based approach for containers placement in fog environment [42]. In this approach, the authors emphasize the impact of heterogeneous inter-container network communication technologies on IoT applications in fog computing. Also, in order to estimate the ensure application response time requirements, the overlay and host mode is considered for inter-container communication.

Hussain and Beg (2021) proposed a DE-based computation offloading approach for task allocation for stationary and mobile fog nodes in Vehicular Fog Computing (VFC) known as CODE-V [43]. CODE-V dynamically develops multi-hop computation offloading to select the optimal path. In addition, the authors formulate the problem as a Quadratic Integer Problem (QIP), taking into account the limitations on computing capacity, latency, hop-limit, task execution (local or remote). Sami et al. (2020) proposed a Kubeadm clustering approach for containerized micro-services placement using MA [44]. This approach makes it possible to containers placement in a distributed, multi-objective, resource-aware, and efficient context with minimal time and cost. In addition, the authors introduce a hybrid multi-layered networking architecture to maintain reachability between available vehicular fog cluster and the requesting user. Nardelli et al. (2019) focused on the problem of distributed placement of data stream processing applications in the fog computing infrastructure [45]. The authors analyze two solutions to the problem, including model-based and non-model-based. They used different techniques for the first solution and compared the three meta-heuristic approaches for the second solution, including local search, greedy first-fit, and TS. All of these methods take into account the heterogeneity of network and computational resources, even for big data samples. The evaluation results have proven the efficiency of TS for executing data stream processing applications. The studies examined are summarized in Table 1.

3 Background

In this section, we depict a brief overview of the Open-source Development Model Algorithm (ODMA) technique and autonomic computing methodology.

3.1 Open-Source Development Model Algorithm

Many decision problems can be expressed as constrained optimization problems [15]. Basically, there

Table 1 Overview of the analyzed studies

Reference	Algorithm	Performance metrics	Methodology
[30]	PSO and ACO	Response time, communication cost and load balancing	An efficient scheduling model for deploying IoT services on fog nodes under response time and communication costs
[32]	ACO	Latency and deployment cost	Service replicas placement in fog using Pareto-based ACO and multi-objective optimization
[33]	ACO	Completion time (makespan) and response time	ACO based container placement in fog environment with makespan minimization to ensure QoS and improve response time
[34]	ABC	Latency and cost	Using ABC to micro-cache placement close to end users on the STBs in Fog-based CDNs to improve cost and latency
[35]	ABC	VM schedule length, runtime and energy consumption	Proposed an ABC-based cost-effective approach to load balancing in fog environment by considering various QoS parameters and prioritizing user demand
[36]	ABC	Energy consumption and network traffic	Development of an ABC-based approach for VMs placement in edge-cloud data centers by considering energy and traffic
[37]	PSO and fuzzy	Latency and network utilization	Proposed mobility-aware approach to improve the fog tasks scheduling on IoT devices by combining PSO and fuzzy logic based on latency and network utilization
[38]	PSO	Latency and energy consumption	Proposed a discrete version of PSO to solve the FSP problem and develop an architecture to describe the interactions between the ecosystem components
[39]	PSO	Bandwidth, latency and makespan	Improving quality of experience (QoE) in fog environment by PSO-based dynamic resources allocation and removing sleepy, unreferred and inactive services
[40]	GA and RL	Latency and energy consumption	An approach to improve service allocation to fog nodes in context-aware smart cities by combining GA and RL
[41]	GA	SLA, cost, response deadline and service availability	An improved multi-objective GA with random-heuristic initialization for load distribution and service placement in edge computing by minimizing SLA violations
[42]	GA	Response time, resource utilization and task runtime	A GA-based approach to aware container placement in fog computing based on the evaluation of inter-container communication
[43]	DE	Latency, task runtime and energy consumption	A DE-based multi-hop computation offloading approach to improve task allocation in VFC by considering the different objectives of QoS
[44]	MA	Latency, cost and runtime	A Kubeadm-based clustering approach to deploy on-demand microservices with the least time and cost using MA and Docker containerization technology
[45]	TS	Resource utilization and response time	Comparison of meta-heuristic approaches (i.e., local search, greedy first-fit and TS) to solve the problem of DSP applications placement on fog infrastructure

are two classes of algorithms for solving hybrid problems, including exact and approximate [9]. Exact algorithms guarantee optimal solution finding but are not satisfactory for solving NP-hard problems. Approximate algorithms can be used when the optimal solution cannot be achieved with exact algorithms. Approximate algorithms seek to find near-optimal solutions and are commonly known as heuristic algorithms. The two main problems of heuristic algorithms are entrapment in local minima and premature convergence [15]. Meta-heuristic algorithms have been proposed to solve these problems and are used in a wide range of problems. These algorithms significantly increase the ability to find quality solutions to NP-Hard optimization problems. Various classes of this type of algorithm have been developed in recent decades. Examples of meta-innovative algorithms include Genetic Algorithm (GA), Particle Swarm Optimization (PSO), Harmonic Search

Algorithm (HSA), Whale Optimization Algorithm (WOA), Cuckoo Search Algorithm (CSA), Gray Wolf Optimization (GWO), ODMA and so on.

Most current meta-heuristic algorithms mimic natural phenomena, for example, GA is inspired by Charles Darwin's theory of natural evolution and was proposed by John Holland (1962) [18]. The PSO proposed by Kennedy and Eberhart (1995) is a meta-heuristic algorithm based on the concept of swarm intelligence, which is inspired by groups of birds and fish [19]. Each solution in the PSO is a particle of the population that evolves its position based on three different factors, including the previous position, *pbest* (best particle personal experience) and *gbest* (best whole swarm experience). HAS is a new meta-heuristic algorithm proposed by Geem et al. (2001) [20]. This algorithm uses musical process and perfect state of harmony modeling for optimization. HAS does not require initial values for decision variables and is derived from a stochastic random search based on pitch adjusting rate and harmony memory. WOA is an evolutionary technique based on hunting behavior of whales that has been introduced by Mirjalili and Lewis (2016) [21]. This algorithm performs the optimization work using three evolutionary operators namely encircling prey, exploitation, and exploration. CSA is an optimization algorithm designed by Yang and Deb (2009) [22]. This algorithm is inspired by the obligate brood parasitism of some cuckoo species by laying their eggs in the nest of host birds of other species. GWO is one of the newest meta-heuristic algorithms proposed by Mirjalili et al. (2014) [23]. The basis of this algorithm is a hierarchical structure that models the social behavior of gray wolves during hunting.

ODMA is a new meta-heuristic algorithm for solving NP-Hard optimization problem proposed by Hajipour et al. (2016) [15]. This algorithm is inspired by the open-source development model and communities, which considers each solution as a software. ODMA takes advantage of both evolutionary and swarm intelligence algorithms without worrying about the complications of combining them. Therefore, every software in this algorithm has memory, data sharing, competition and collaboration. The software population in ODMA falls into two groups: promising and leading. Leading software is more popularity in user communities than promising software. The main operations of evolution in this algorithm include (i) moving towards the leading softwares, (ii) evolution of the leading softwares based on their history, and (iii) removing weak softwares and forking new softwares based on the leading softwares [15].

Figure 1 shows the ODMA flowchart [15]. As illustrated, the initial population of softwares is first randomly assigned. The softwares is then sorted according to a fitness function, so that the first software has the best value of the fitness function. After that, three steps related to softwares evolution are applied. Finally, the above steps are repeated until the stop condition is satisfied. This paper uses ODMA to optimize the deployment process of IoT services on fog nodes.

3.2 Autonomous Systems

The autonomic computing paradigm was introduced by IBM in 2001 to describe the activities of computer systems, where it was able to adapt to changes in the environment without user intervention [35]. Autonomic computing can enable computer systems to manage

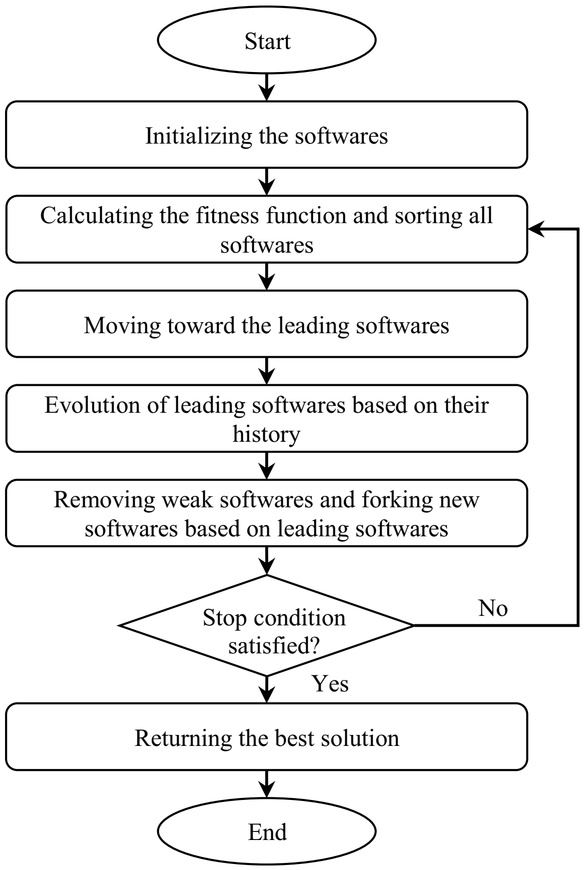
Start
Initializing the softwares
Calculating the fitness function and sorting all softwares
Moving toward the leading softwares
Evolution of leading softwares based on their history
Removing weak softwares and forking new softwares based on leading softwares
Stop condition satisfied?
No
Yes
Returning the best solution
End

Fig. 1 ODMA flowchart

independently through self-management. Each autonomic system has the characteristics of self-optimization, self-protection, self-healing, and self-configuration. Self-optimization refers to identifying opportunities to improve QoS according to defined objectives [36]. Self-protection is related to the privacy and security of computer systems by identifying types of attacks. Self-healing can provide the reliability and availability of computer systems by detecting unexpected errors. Finally, self-configuration refers to setting the basic parameters and operating conditions of computer systems automatically.

IBM introduced the MADE-k model (monitoring, analysis, decision-making, and execution control loop with a shared knowledge base) to achieve self-management capabilities and achieve independent management of computer systems, as shown in Fig. 2. In the monitoring phase, data is captured by network elements and sensors from a dynamic environment. The analysis phase is responsible for collecting and converting data in order to create a system status recognition model. The decision-making phase involves planning based on event-condition-action rules that can put the system in the desired state. Finally, the execution phase executes the decisions made by the decision-making phase by influencing the managed element. In addition, knowledge is a common database among the various phases for storing and exchanging events.

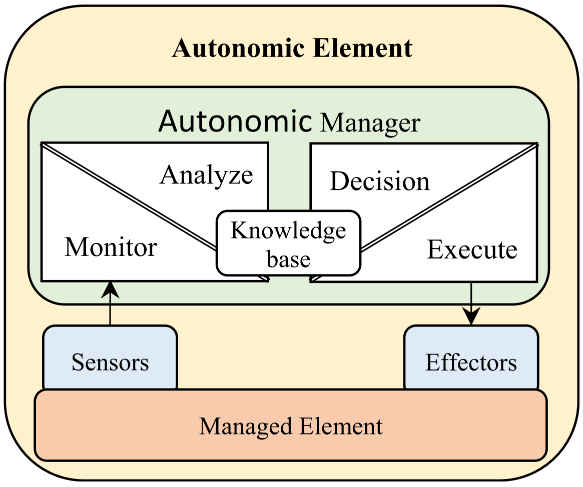

Fig. 2 IBM MADE-k control loop

4 Proposed Framework

In this section, a three-layer conceptual computing framework (i.e., cloud-fog-IoT) is proposed to describe the interactions between system components and the FSP problem-solving policy. This framework formulates the FSP problem based on the autonomous MADE-k model [14]. The FSP problem determines which IoT services are deployed on which fog nodes to execute. Let n applications be available for processing by IoT devices, where $A = \{a_1, a_2, \ldots, a_i, \ldots, a_n\}$ and a_i refer to the i^{th} application. Each IoT application contains a number of tasks. Each task is considered as an IoT service request created by IoT devices. The whole of all services from n IoT applications includes the r service, where $S = \{s_1, s_2, \ldots, s_j, \ldots, s_r\}$ and s_j refer to the j^{th} service. On the other hand, FCL includes m decentralized fog nodes, where $F = \{f_1, f_2, \ldots, f_k, \ldots, f_m\}$ and f_k refer to the k^{th} fog node.

The $s_j \in a_i$ service can be subdivided into a set of tasks, each of which is assigned to a fog node for execution. Thus, the f_k fog node can execute a subset of parsed tasks from separate services. Each node contains a set of VMs that can host a subset of parsed tasks. For convenience, we assume services are inseparable and perform all tasks of a service on an independent fog node. Also, each service is hosted by only one fog node until the end of the execution process.

As shown in Fig. 3, the FSP problem refers to the services placement from IoT applications on decentralized fog nodes, so that the time deadline of the applications is guaranteed. Each application contains several services indicated by ellipses. A IoT service is assigned to only one fog node. However, a fog node can support more than one IoT service due to its free resources, because the resources in the fog nodes are provided by different VMs. Links between IoT services indicate that they can exchange data according to the Directed Acyclic Graph (DAG) model [46]. IoT services execute in the FCL and close to data resources, as each service requires specific resources at a reasonable time. Properly allocating IoT services to limited resources is a challenge that defines the FSP problem. Therefore, a proper placement policy is needed to optimize resources and ensure QoS, for which a conceptual computing framework based on the autonomous MADE-k model is proposed in this paper.

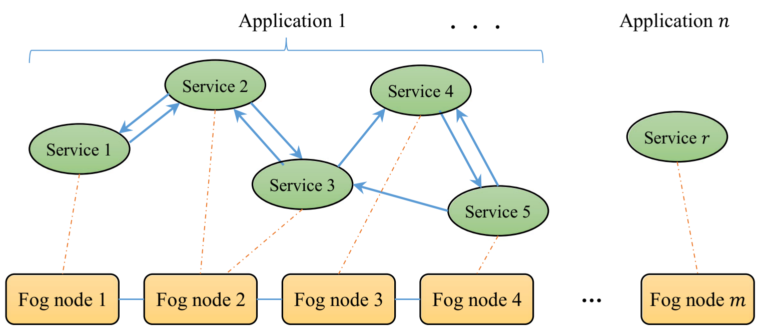

Fig. 3 Overview of the FAP problem

As shown in Fig. 4, the proposed framework consists of three layers: cloud servers (i.e., CCL), fog domains (i.e., FCL), and IoT devices. The CCL is at the highest level, then the FCL, and the IoT device layer is at the lowest level. The IoT device layer consists of a large volume of low-cost smart devices that collect endpoint information and send it to the FCL for processing. FCL has limited computing and storage resources provided by fog nodes. Fog nodes can be set-top-boxes, access points, base stations, servers, gateways, switches, routers, and more. Meanwhile, IoT devices such as smartphones, sensors, smartwatches, smart vehicles, cameras, etc., also show end users. The FCL receives requests from the IoT device layer to provide the resources they need. When the FCL fails to provide these resources, the request must be forwarded to the CCL for execution. CCL consists of a set of high-powered storage and computing servers capable of processing complex requests.

In order to efficiently manage and improve link rates, fog nodes in the FCL are subdivided into non-overlapping domains, where each domain is managed by a dedicated node called FOCN. FOCN in each domain is randomly selected from the fog nodes and is responsible for domain workload management, authentication, and monitoring of the subordinate fog domain. In addition, fog domains can work together to respond to requests through FOCN. The proposed framework includes p of the fog domain, where $D = \{d_1, d_2, \ldots, d_j, \ldots, d_p\}$ and d_j refer to the j^{th} fog domain. Each domain is connected to a cloud environment through cloud gateways and includes a FOCN and several of fog nodes. o_j is the FOCN in the domain d_i, and $Res(d_i)$ represents the set of subordinate fog nodes in this domain. Here, $f_{i,\ j}$ refers to the i^{th} fog node in the j^{th} fog domain. In addition, the FCL has a Cloud-Fog Control Middleware (CFCM) that manages resource requests. Therefore, all resource requests generated by the IoT device layer are sent to the CFCM in the FCL through the fog gateway.

Communication links between ecosystem components have low latency rates. l_M^D is the communication link latency associated with the IoT devices and CFCM, $l_{f_i}{}^{o_j}$ is the communication link latency associated with j^{th} FOCN and the i^{th} fog node, $l_N^{o_j}$ is the communication link latency between j^{th} FOCN and the Nearest Neighboring Fog Domain (NNFD), $l_{o_j}^M$ is the communication link latency between CFCM and j^{th} FOCN, and l_R^M refers to the communication link latency between CFCM and R cloud resources. All fog nodes have storage capacity, processor and memory, denoted by S_{f_i}, P_{f_i} and M_{f_i}, respectively. Similarly, these factors for FOCN are denoted by S_{o_j}, P_{o_j} and M_{o_j}, respectively. On the other hand, every IoT application such as a_i includes r_i independent services that must be placed on fog nodes. Each $s_j \in a_i$ service has different storage, processor, and memory requirements, denoted by S_{s_j}, P_{s_j}, and M_{s_j}, respectively. Plus, each IoT service has a time deadline indicated by D_{s_j}, where D_{a_i} can be the time deadline of application a_i.

According to the proposed framework, the CFCM receives and processes resource requests by the Admission Control Unit (ACU). This unit detects applications that are sensitive to latency and real-time, and transfers other applications to CCL for execution. In addition to ACO, CFCM includes the

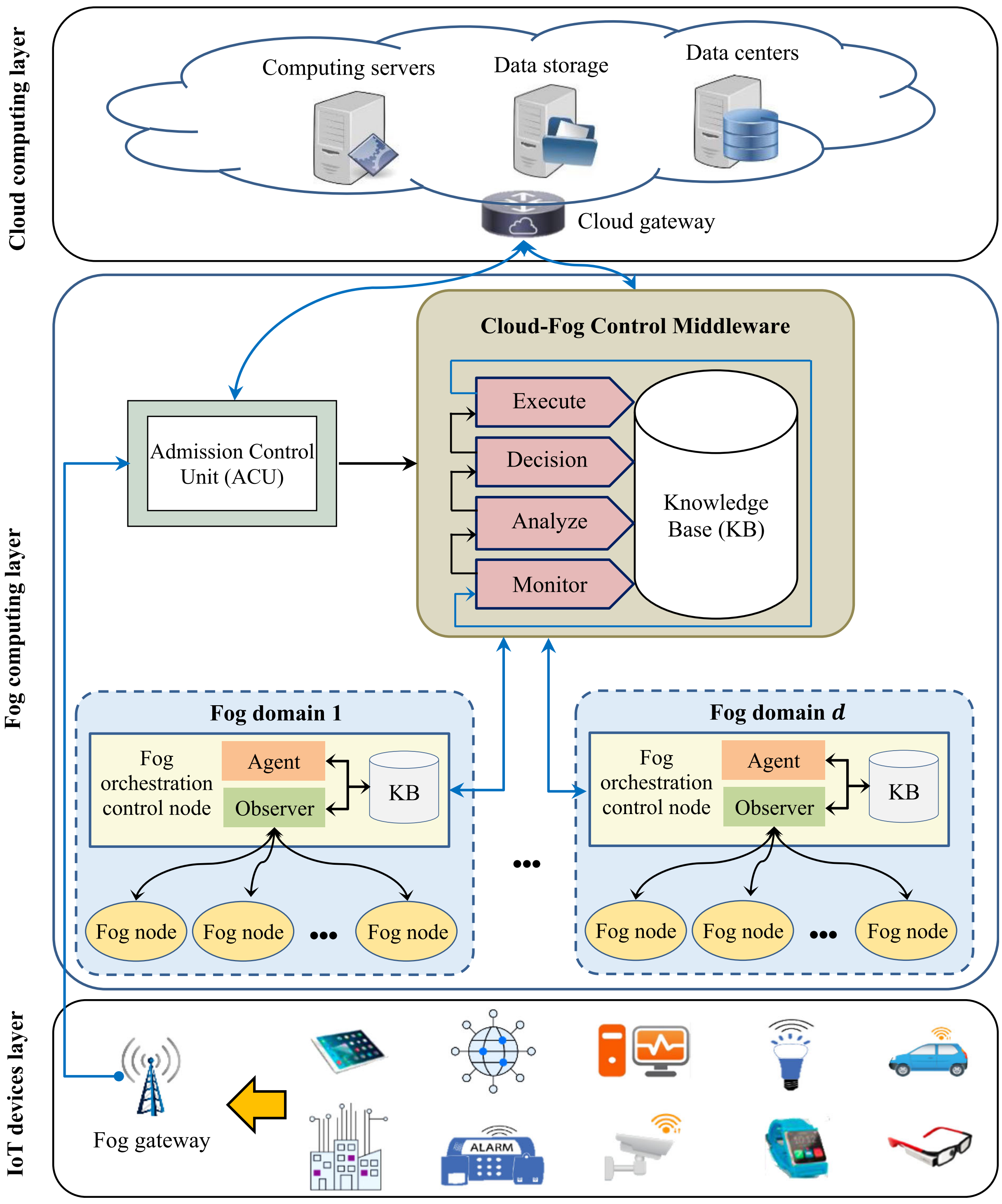

Fig. 4 Proposed conceptual computing framework

autonomous MADE-k model. MADE-k is a control loop model that can monitor, prioritize, placement, and finally execute resource requests. The CFCM performs the deployment of IoT services on fog nodes based on the MADE-k model and informs the FOCNs of the details of the planning. The data required by the MADE-k model for placement through the FOCN is sent to the CFCM at different time periods. Therefore, each FOCN makes details of subordinate fog domain resources available to the CFCM at specified timeslots. The proposed mechanism emphasizes the sending of this information at the beginning of each time period τ.

When none of the fog nodes in a domain can provide the resources needed for a request, the request is sent to the NNFD by FOCN. This can be repeated by other domains. Finally, if the required resources are not provided by any of the domains, FOCN notifies the CFCM of this issue and the CFCM sends the request to the CCL for execution. A description of all the variables and symbols associated with the problem is shown in Table 2. In addition, Table 3 shows the details of all the abbreviations used in this paper.

High-level descriptions of how the proposed framework works and the interactions between ecosystem components are provided below.

4.1 Fog Node

The main entity in the FCL are fog nodes. These nodes are decentralized computing entities that enable the deployment and execution of IoT services. Fog nodes communicate with IoT devices through various protocols such as SNMP and CoAP [3]. They have the ability to temporarily storage data and periodically send their performance reports to the CCL. A fog node has five components including listener, monitor, database, control manager and compute. The listener component is responsible for receiving the IoT service from FOCN. The monitor component monitors how the IoT service is executed in the compute component. Fog landscape status, monitored data, available node resources, and IoT service details are stored in the database component. The control manager component performs some operations on the fog node. Finally, the resources required for the IoT service are provided by the compute component to execute the service.

Each fog node as a computing resource has storage, processing, and memory capabilities for deploying and executing IoT services. Nodes in the fog domain consist of two types of thin and fat devices that only fat devices can process and calculate. Thin devices are used as sensors and actuators. Therefore, fat devices with different resource capabilities are considered as fog nodes and are used to provide IoT services according to various needs. A node among the available fog nodes in a fog domain is randomly considered as FOCN, which has various tasks to manage the fog domain.

4.2 Fog Domain

Each fog domain consists of a FOCN and several fog nodes with different resource capabilities. The fog domain is supported by a head element in the cloud (i.e., CFCM). In the proposed framework, there are several fog domains that regularly work timeslots at regular intervals to deploy IoT services on appropriate fog nodes according to QoS requirements. The fog domain manages the IoT services received from CFCM. The fog domain stores the capabilities of existing fog nodes and the QoS requirements associated with IoT services in its knowledge base. In addition, the fog domain is responsible for determining the appropriate scheduling to provide the resources needed by IoT services according to the autonomous computing model.

4.3 Fog Orchestration Control Node

FOCN monitors the execution of the IoT service on the dependent subordinate fog nodes. This node cooperates to execute services with other fog domains through low-latency communication links. In addition, FOCN can communicate with the CFCM or send some requests to the CCL. Resource utilization analysis and subordinate domain reconstruction are other FOCN tasks.

A FOCN has six components: listener, reasoner, propagation, watchdog, registry, and storage. The *listener* component is responsible for receiving IoT services from the CFCM. The *reasoner* component

Table 2 Description of all variables and symbols related to the problem

Variables	Description	Variables	Description
n	Number of IoT applications	S_{f_i}	Storage capacity of f_i
r	Number of IoT services	P_{f_i}	Processor capacity of f_i
m	Number of fog nodes	M_{f_i}	Memory capacity of f_i
p	Number of fog domains	S_{o_j}	Storage capacity of o_j
A	Set of applications	P_{o_j}	Processor capacity of o_j
S	Set of services	M_{o_j}	Memory capacity of o_j
F	Set of fog nodes	S_{s_j}	Storage capacity required by s_j
D	Set of fog domains	P_{s_j}	Processor capacity required by s_j
a_i	The i^{th} application	M_{s_j}	Memory capacity required by s_j
s_j	The j^{th} service	D_{s_j}	Time deadline of s_j
f_k	The k^{th} fog node	D_{a_i}	Time deadline of a_i
d_j	The j^{th} fog domain	RT_{a_i}	Execution time of a_i
o_j	The j^{th} FOCN	$P(a_i)$	The priority of a_i
$Res(d_i)$	Set of subordinate fog nodes in d_i	R_{a_i}	The time of sending the a_i
r_i	Number of independent services in a_i	$SC(a_i)$	Service cost factor for a_i
$f_{i,\,j}$	Fog node f_i in the domain d_j	$EC(a_i)$	Energy consumption factor for a_i
l_M^D	Link latency between devices and CFCM	$RT(a_i)$	Response time factor for a_i
$l_{f_i}{}^{o_j}$	Link latency between o_j and f_i	$LT(a_i)$	Latency factor for a_i
$l_N^{o_j}$	Link latency between o_j and NNFD	$FU(a_i)$	Fog resource utilization factor for a_i
$l_{o_j}^M$	Link latency between CFCM and o_j	ξ_{SC}	Weight of service cost factor
l_R^M	Link latency between CFCM and cloud	ξ_{EC}	Weight of energy consumption factor
l_M	Latency processing of service workloads in CFCM	ξ_{RT}	Weight of response time factor
l_O	Latency processing of service workloads in FOCN	ξ_{LT}	Weight of latency factor
l_f	Latency processing of service workloads in fog node	ξ_{FU}	Weight of fog resource utilization factor
l_N	Latency processing of service workloads in NNFD	$X_{s_j}^{dv}$	Binary variable for s_j deployment on device dv
l_R	Latency processing of service workloads in cloud	$CM_{s_j}^{dv}$	Execution time of service s_j on the device dv
φ	Free resource rate of a fog node	$CP_{s_j}^{dv}$	Execute cost of service s_j on the device dv
τ	The time period	$E_{AC}(s_j)$	Energy consumed by ACU for s_j
t	System current time	$E_{ST}(s_j)$	Energy consumed by FOCN for s_j
x_O	binary decision variables for FOCN	$E_{FG}(s_j)$	Energy consumed by fog node for s_j
x_f	binary decision variables for fog node	$ET(a_i)$	Execution time of a_i
x_N	binary decision variables for NNFD	$WT(a_i)$	Waiting time of a_i
x_R	binary decision variables for cloud	$CT(a_i)$	Communication time of a_i

Table 3 Details of all abbreviations used in this paper

Acronyms	Definition
QoS	Quality of Service
FSP	Fog Services Placement
ODMA	Open-source Development Model Algorithm
MADE-k	Monitoring, Analysis, Decision-Making, And Execution-knowledge base
FCL	Fog Computing Layer
CCL	Cloud Computing Layer
VM	Virtual Machine
FOCN	Fog Orchestration Control Node
NNFD	Nearest Neighboring Fog Domain
CFCM	Cloud-Fog Control Middleware
ACU	Admission Control Unit

analyzes the details of deploying IoT services. The *propagation* component is responsible for transferring services whose resources are not available on the subordinate fog domain. IoT services are sent to the CFCM and then to the CCL, or to the NNFD depending on the resources available in other domains. FOCN can find the NNFD by checking the communication link latency. In the *watchdog* component, QoS parameters and resources consumed in the subordinate fog domain are measured. The *registry* component provides the allocation of services in the compute component of fog node. Finally, the deployment of IoT services in the fog domain is maintained by the *storage* component.

4.4 FSP Problem Limitations

We define two limitations including the execution time deadline and the resources utilization for the FSP problem. The first limitation refers to the execution time deadline of an application, which varies according to the type of application. In general, the execution time of the application in the ecosystem should not exceed the time deadline of that application. Let RT_{a_i} be the execution time of a_i on the ecosystem and D_{a_i} the time deadline of this application. In this case, the time deadline limitation is defined by Eq. (1).

$$RT_{a_i} \le D_{a_i}, \quad \forall a_i \in A \tag{1}$$

The second limitation refers to the amount of storage capacity, processor and memory utilization of the fog nodes by IoT services. Therefore, the resources required by IoT services should not be more than the resources available for fog nodes, where this is also the case for FOCN. Let f_k be the fog node assigned to the s_j IoT service. In this case, the limitation of the resource's utilization is defined by Eq. (2).

$$S_{s_j} \le \varphi S_{f_k} \& P_{s_j} \le \varphi P_{f_k} \& M_{s_j} \le \varphi M_{f_k}, \quad \forall s_j \in a_i, a_i \in A \tag{2}$$

Where, φ refers to the free resource rate of a fog node that can be used for internal activities and data exchange with other fog nodes.

4.5 Cloud-Fog Control Middleware

The CFCM, as the most important component in the FCL, is responsible for deploying and managing IoT services. CFCM operates on two main components including ACU and MADE-k model. Requests are received by the ACU from IoT devices. CFCM stores the capabilities of enabled fog nodes and information about the QoS requirements of IoT services in its knowledge base. Thus, the MADE-k model automatically decides whether to placement IoT services on fog nodes and send them to fog domains. In addition, CFCM monitors the utilization of cloud resources. Hence, there is a bridge between FCL and CCL through the cloud gateway.

CFCM periodically solves the FSP problem. The proposed mechanism is that at each τ time period, the IoT services are deployed on the fog nodes by the MADE-k model. Therefore, newly received services should be postponed until the nearest time period. Finally, the execution of the IoT services starts after the complete deployment of all received requests in a time period. It is important to note that CFCM deploys IoT services in a distributed manner for each fog domain. Therefore, details of the deployment of IoT services after placement are sent to the FOCNs in each fog domain.

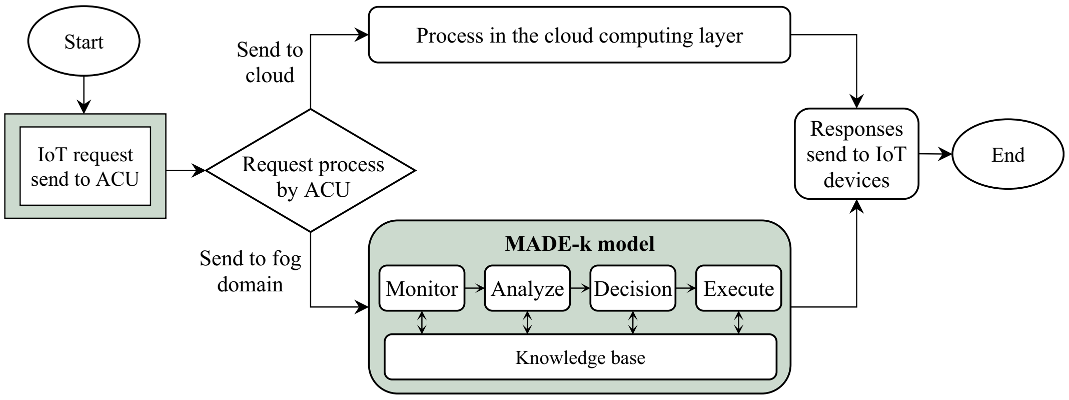

Fig. 5 The admission control function

4.5.1 Admission Control Unit

The CFCM has an ACU that receives requests from IoT devices through fog gateways at the edge of the network, as shown in Fig. 5. Based on the response time and time deadline of the applications, this unit detects applications that are sensitive to latency and real-time, and transfers other applications to CCL for execution. Plus, when the FCL is unable to provide the resources needed for a request, the ACU sends the request to the CCL. This unit has a queue that processes FCL-related requests at regular timeslots. The proposed mechanism is that in each τ time period, IoT services are sent from the queue to the planning model (i.e., MADE-k). This can ensure ecosystem dynamics for latency-sensitive and real-time applications.

4.5.2 Autonomous MADE-K Model

There is a CFCM in the FCL that deploys IoT services based on the MADE-k model. This model consists of four phases of monitoring, analysis, decision-making and execution control loop with a shared knowledge base. The MADE-k model deploys IoT services for each fog domain autonomously and distributed. When the service fails to execute on the fog domain, it must be sent to the NNFD or CCL, which will increase the latency. Therefore, the FSP problem-solving approach must deploy IoT services efficiently and taking into account the QoS requirements to minimize latency. In addition to latency, there are other metrics such as response time, energy consumption, fog resource utilization, cost, etc. in solving the FSP problem that must be considered to satisfy. Therefore, we formulate the FSP problem as a multi-objective optimization problem to solve it with a meta-heuristic approach. In this paper, the QoS-aware IoT services placement policy in FCL based on ODMA as a meta-heuristic algorithm is performed. Placement planning and request management for execution is done by autonomous MADE-k model. This model receives requests from the ACU and processes them in four phases as follows.

Monitoring phase The input of this phase is the data related to the available fog nodes from the subordinate fog domain and the set of IoT services existent in time period τ. This phase is responsible for monitoring free resources in the fog domain, FOCN status, resources required by IoT services, and fog node status. At this phase, the utilization of storage, processor and memory in each time period is monitored. Also, the monitoring phase is responsible for managing requests that are sent to the CCL due to lack of required resources. In addition, in this phase, details of IoT applications, resources required by IoT services, and fog nodes status are stored in the knowledge base.

Analysis phase This phase sets the priority of executing IoT services to reduce IoT service latency and improve the placement process. Given that each application includes IoT several services, so priorities are determined by the application. In this regard, all IoT services in an application have the same priority and are executed based on the priority set for their application. In this paper, priority is defined based on the time interval to the application deadline, as done in [4, 5]. Accordingly, applications with the shortest time to deadline has a higher priority for placement and execution. $P(a_i)$ refers to the priority of the application a_i, which is defined by Eq. (3).

$$P(a_i) = \frac{1}{D_{a_i} - (t - R_{a_i})} \tag{3}$$

Where, D_{a_i} is the execution time of the application a_i, t is the current time and R_{a_i} is the time of sending the a_i by the end user.

Decision-making phase The input to this phase is a set of IoT services that are prioritized according to their applications. In addition, details of fog nodes and free resources in the fog domain are available in this phase. Based on this data, the decision to IoT services placement is made at this phase. In this paper, the decision-making process is performed by ODMA based on multi-objective optimization. This algorithm provides a placement scheme for the fog domain independently, as described in Section 6. At the end of this phase, placement details created for IoT services are stored in the knowledge base.

Execution phase In this phase, according to the placement scheme created in the decision-making phase, IoT services are deployed and executed on fog nodes. In this phase, the results of executing IoT services on fog nodes are stored in the knowledge base.

Algorithm 1. Pseudocode of the proposed framework

```
    Input: IoT applications and services of each application, the available resources of the fog domain
    and the fog nodes status.
    Output: IoT service placement scheme.
1:  for each $\tau$ time period do
2:      for each application $a_i \in A$ do
3:          Monitor (set of IoT services requested for $a_i$).
4:          Monitor ($S_{s_j}, P_{s_j}, M_{s_j}, D_{s_j}$).
5:          Storing details of IoT services in the knowledge base.
6:      end
7:      for each fog node $f_k$ in the domain do
8:          Monitor (status of fog node $f_k$).
9:          Monitor ($S_{f_k}, P_{f_k}, M_{f_k}$).
10:         Storing details of fog node status in the knowledge base.
11:     end
12:     for each set of application $a_i \in A$ do
13:         Calculate the priority of $a_i$ with Eq. (3).
14:         Storing application prioritization details in the knowledge base.
15:     end
16:     for each set of IoT service $s_j \in a_i, a_i \in A$ do
17:         Using ODMA to determine the IoT service placement scheme based on application
            priority.
18:         Storing the placement details of IoT services in the Knowledge base.
19:     end
20:     for each set of IoT service $s_j \in a_i, a_i \in A$ do
21:         Deploy the IoT service $s_j$ based on the placement scheme created and execute it on the
            fog node.
22:         Storing the results of executing IoT services on fog nodes in the knowledge base.
23:     end
24: end
25: IoT service placement scheme created by ODMA as the output of the algorithm.
```

The pseudocode of the proposed framework for the cloud-fog-IoT ecosystem is shown in Algorithm 1. The loop embedded in line 1 is to execute the phases of the autonomous MADE-k model in each time period such as τ. Lines 2–6 as well as lines 7–11 perform monitoring phases for IoT services and the fog nodes status, respectively. In addition, this phase monitors storage, processor, and memory utilization on line 4 for IoT services and line 9 for fog nodes. Also, the details of the IoT services and the fog nodes status are stored in lines 5 and 10, respectively. Lines 12–15 relate to application prioritization in the analysis phase. Lines 16–19 relate to the decision-making phase, which carries out the QoS-aware placement scheme for deploying IoT services using ODMA. In lines 20–23, the created placement scheme is applied and IoT services are executed on the fog nodes. Finally, the output of the algorithm is described in line 25.

Algorithm 2. Pseudocode of the DMA-FSP approach to solve the FSP problem

```
Input: Details of services, details of system resources and ODMA parameter values such as $N_P$,
$Max_{iter}$, $z$, $\rho$, $n_\omega$ and $P_{iter}$.
Output: The best solution for IoT services placement.
1:  for $i = 1$ to $N_P$ do
2:      Solution [$i$] ← Creating a random solution based on defined encoding in Fig. 6.
3:      Fitness [$i$] ← Calculating the fitness of Solution [$i$] based on the Eq. (4).
4:      Score [$i$] ← 0.
5:  end
6:  for $iter = 1$ to $Max_{iter}$ do
7:      Sort all solutions based on the fitness function.
8:      $\phi$ ← Selection of leading solution indexes based on Eq. (21).
9:      for $j = z$ to $N_P$ do
10:         $soft_\phi$ ← Select a leading solution from $\phi$ at random.
11:         $soft_j$ ← Moving solution $soft_j$ toward solution $soft_\phi$ with probability $\rho$ according to
            Fig 8.
12:         $fit_j$ ← Calculating the fitness of $soft_j$ based on the Eq. (4).
13:         if $fit_j \leq$ Fitness [$j$] then
14:             Solution [$j$] ← $soft_j$.
15:             Fitness [$j$] ← $fit_j$.
16:             Score [$\phi$] ← Score [$\phi$] + 1.
17:             Score [$j$] ← Score [$j$] + 1.
18:         end
19:     end
20:     Sort all solutions based on the fitness function.
21:     for $j = 1$ to $z$ do
22:         $soft_j$ ← Moving solutions $soft_{cur}$ and $soft_{old}$ based on Eq (22).
23:         $fit_j$ ← Calculating the fitness of $soft_j$ based on the Eq. (4).
24:         if $fit_j \leq$ Fitness [$j$] then
25:             Solution [$j$] ← $soft_j$.
26:             Fitness [$j$] ← $fit_j$.
27:             Score [$j$] ← Score [$j$] + 1.
28:         end
29:     end
30:     Sort all solutions based on the fitness function.
31:     if $iter \% P_{iter} == 0$ then
32:         Removing $n_\omega$ poor solution based on average score and fitness.
33:         for $j = 1$ to $n_\omega$ do
34:             $soft_L$ ← Select a leading solution with the highest fitness.
35:             Solution [$j$] ← Forking new solution from $soft_L$ based on Eq (23).
36:             Fitness [$j$] ← Calculating the fitness of Solution [$j$] based on the Eq. (4).
37:         end
38:     end
39: end
40: Return the best solution based on the fitness function.
```

5 Proposed Placement Policy with ODMA

In this paper, the FSP problem is formulated as a multi-objective optimization problem considering some QoS requirements, which is an NP-Complete problem [28]. In order to prove the effectiveness of meta-heuristic techniques in solving optimization problems, in this paper, ODMA is used as a meta-heuristic algorithm to solve the FSP problem, which we call FSP-ODMA. FSP-ODMA is a QoS-aware placement policy in the MADE-k model that decides on the optimal deployment of IoT services by compromising between different objectives such as service cost, energy consumption, response time, latency and fog resource utilization.

Like other evolutionary algorithms, ODMA consists of three main steps: 1) encoding solution (as a software) and creating the initial population, 2) calculating the fitness of the solutions, and 3) population evolution to satisfaction of the stop conditions. Basically, steps 1 and 2 are similar for evolutionary algorithms, and it is this way of population evolution in step 3 that distinguishes the optimization process in the algorithms. The evolution process in ODMA is based on two groups of solutions, namely promising and leading, which is done in three steps: (i) moving towards the leading softwares, (ii) evolution of the leading softwares based on their history, and (iii) removing weak softwares and forking new softwares based on the leading softwares [15]. The details of ODMA to solve the FSP problem are described below. In addition, the pseudocode of the FSP-ODMA approach to solving the FSP problem is shown in Algorithm 2.

Lines 1–5 are dedicated to the initial population creation step. In line 6, the algorithm is repeated to the specified number. Line 7 refers to the sorting of solutions based on the fitness function. The selection of a subset of the leading solutions in line 8 is done. Lines 9–19 relate to the first step in the evolution of the ODMA algorithm. Lines 20–29 are related to the second step and lines 30–38 are related to the third step of evolution.

5.1 Encoding Solution and Creating the Initial Population

In ODMA, each iteration consists of a population of software, so that each software encodes a solution to the problem. Here, the initial population includes the N_P solution, which is encoded based on Integer Linear Programming (ILP) to assign IoT services to fog nodes. Figure 6 shows the proposed encoding for representations of solutions in FSP-ODMA. As illustrated, each solution contains r independent element as the number of fog services available for placement in time period τ. In this encoding, each element refers to a separate IoT service from a specific application. On the other hand, the value of each element refers to an available fog node. Consequently, $v_k \in F$ is a fog node assigned to the IoT service s_j. Due to the ability to host multiple services by one fog node, services can be duplicated in more than one fog node [4].

This encoding ensures that all IoT services are deployed on fog nodes. After defining the encoding, the initial population containing the N_P solution is generated randomly. Figure 7 provides an example of a solution encoding process for better understanding, where 12 IoT services from 4 applications are assigned to 4 fog nodes. In this example, application a_1 contains the IoT services $\{s_1, s_2, s_3\}$, which are deployed on the fog nodes $\{f_1, f_1, f_2\}$, respectively. Application a_2 consists of two IoT services $\{s_4, s_5\}$, which are assigned to fog nodes $\{f_1, f_3\}$, respectively. The IoT services $\{s_6, s_7, s_8, s_9\}$ belong to the application a_3, which are mapped to the fog nodes $\{f_1, f_2, f_3, f_4\}$, respectively. Finally, IoT services $\{s_{10}, s_{11}, s_{12}\}$ of application a_4 are deployed on fog nodes $\{f_3, f_4, f_4\}$, respectively.

5.2 Fitness Function

In each iteration of the optimization process, the fitness of the solutions must be calculated. The fitness function (also known as the objective function) is used to evaluate solutions to the ideal solution. In this paper, the FSP problem is formulated as a multi-objective optimization problem, where it is rarely possible in theory to produce the ideal solution to such problems that satisfy all objectives at the same time [5]. Therefore, the ideal solution must be determined based on a compromise and agreement between the objectives.

Several objectives in the FSP problem have been explored by researchers to satisfy different QoS requirements, for example, number of services accepted, fog resource utilization, SLA violation, energy consumption, response time, service time, latency, throughput, service cost, etc. However, some objectives including service cost, energy consumption, response time, latency and fog resource utilization have been identified by many researchers as the main objectives in the FSP problem [4, 5]. Hence, we calculate the fitness of solutions with the aim of minimizing service costs, energy consumption, response time and latency, and maximizing the fog resource utilization. Thus, the fitness function for the FSP problem is defined by the five objectives considered by Eq. (4), where we seek to minimize it.

$$Fitness = \xi_{SC}.SC + \xi_{EC}.EC + \xi_{RT}.RT + \xi_{LT}.LT - \xi_{UF}.FU \tag{4}$$

Where, *SC*, *EC*, *RT*, *LT* and *FU* refer to the factors of service cost, energy consumption, response time, latency and fog resource utilization, respectively, and ξ_{SC}, ξ_{EC}, ξ_{RT}, ξ_{LT} and ξ_{FU} are the weights associated with these factors.

According to [7], the effect of all objectives is considered the same. The process of calculating *SC*, *EC*, *RT*, *LT* and *FU* based on the IoT application is discussed below, where the sum for all applications must be considered to calculate the final fitness. Meanwhile, Since the factors used in the objective function have different units of measurement, the values of each factor are normalized before calculation.

Service cost (SC): The service cost refers to the monetary cost associated with that service, which includes the communication service (i.e., the service execution time) and the computational cost (i.e., the cost of executing the service on the node) [48]. Each service can be deployed and executed on cloud *R*, FOCN or fog node. Let $X_{s_j}^{dv}$ be a binary variable that indicates the IoT service s_j is deployed on the device $dv \in \{R, O, f\}$. Since there is no priority to execute services owned by an application, we calculate the cost factor for an application. Eq. (5) defines the cost factor for the application a_i.

$$SC(a_i) = \sum_{s_j \in a_i} X_{s_j}^{R} CM_{s_j}^{R} CP_{s_j}^{R} + X_{s_j}^{O} CM_{s_j}^{O} CP_{s_j}^{O} + X_{s_j}^{f} CM_{s_j}^{f} CP_{s_j}^{f} \tag{5}$$

Where, $CM_{s_j}^{dv}$ and $CP_{s_j}^{dv}$ are the execution time and execute cost of service s_j on the device *dv*, respectively.

Energy consumption (EC) The energy consumption of each service requested in the cloud-fog-IoT ecosystem includes three types of energy: 1) energy consumed to service processing in the ACU, 2) energy consumed to transfer service to the FOCN via CFCM, and 3) energy consumed to service execution in the fog node. Therefore, the energy consumption of the IoT service s_j is calculated based on the sum of these three types of energy. Hence, the total energy consumption of the application a_i can be calculated based on the sum of the energies consumed by all the services belonging to a_i, as shown in Eq. (6).

$$EC(a_i) = \sum_{s_j \in a_i} E_{AC}(s_j) + E_{ST}(s_j) + E_{FN}(s_j) \tag{6}$$

Where, $E_{AC}(s_j)$, $E_{ST}(s_j)$ and $E_{FN}(s_j)$ refer to the energy consumed by ACU, FOCN and fog node for IoT service s_j, respectively.

The energy consumed for service processing in the ACU (i.e., E_{AC}) is measured based on the size of the service. E_{AC} is defined for the IoT service s_j in Eq. (7).

$$E_{AC}(s_j) = SS_{s_j}.\theta_{AC}.n_{AC} \tag{7}$$

Where, SS_{s_j} is the size of the IoT service s_j based on the number of bits, θ_{AC} is the energy consumed per processor cycle in the ACU, and n_{AC} is the number of cycles required by the processor to process one bit of data.

The energy consumed to transfer the service to FOCN via CFCM (i.e., E_{ST}) is measured based on the communication architecture. E_{ST} is defined for the IoT service s_j in Eq. (8).

$$E_{ST}(s_j) = \frac{SS_{s_j}.te_f}{bw_f.se_{net}} \tag{8}$$

Where, te_f is the transmission energy of the signal transmitter, bw_f is the output bandwidth of FOCN and se_{net} is the spectral efficiency of the communication link between the CFCM and FOCN. Considering te_f, se_{net} is calculated by Eq. (9) [10].

$$se_{net} = \log_2\left(1 + \frac{p_f.k_f.\gamma_f}{bw_f.N_0 + ip_f}\right) \tag{9}$$

Where, p_f is the probability of FOCN being idle, k_f is the shadowing factor in the communication link, γ_f is the channel loss rate, N_0 is the noise power spectral density, and ip_f refers to the interference power in the FOCN.

There are two types of fog nodes (i.e., thin and fat) in the proposed framework, where only fat nodes can perform calculations and processing. The energy consumed to execute the service in the fat type node (i.e., E_{FN}) is measured based on the processing time and the amount of processor utilization. E_{FN} is defined for the

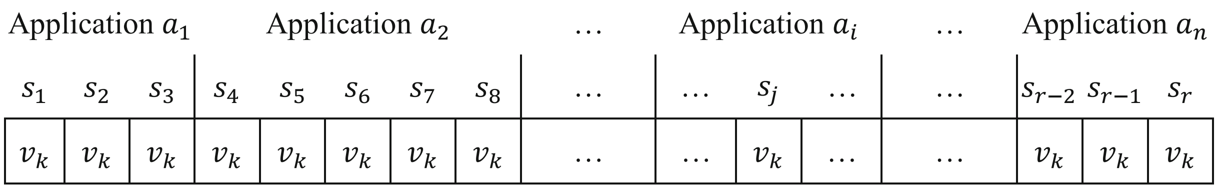
Application a_1
Application a_2
…
Application a_i
…
Application a_n
s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 … … s_j … … s_{r-2} s_{r-1} s_r
v_k v_k v_k v_k v_k v_k v_k v_k … … v_k … … v_k v_k v_k

Fig. 6 Proposed encoding to representations of solutions

IoT service s_j in Eq. (10).

$$E_{FN}\left(s_j\right)=\int_{t_\alpha}^{t_\beta}E_{s_j}(t)dt \tag{10}$$

Where, t_α and t_β are the start and end times of the IoT service s_j, respectively, and $E_{s_j}(t)$ is the energy consumed to execute the IoT service s_j at time t. E_{s_j} is calculated by considering the relationship between the FOCN and the fog node according to Eq. (8).

Response time (RT): The response time of an application is estimated based on the execution time, waiting time, and communication time of all IoT services belonging to that application, as defined in Eq. (11).

$$RT(a_i)=ET(a_i)+WT(a_i)+CT(a_i) \tag{11}$$

Where, $RT(a_i)$ is the response time of application a_i, and ET, WT, and CT refer to execution time, waiting time, and communication time, respectively.

Basically, the execution time of a IoT service depends on the hardware configuration of the fog node (i.e., total memory size, memory utilization, cache size, and processor power) and the software features of the services depend on the application (i.e., the computational time complexity of the application module). Here, the Whetstone criterion is used to calculate the service processing capability and the TPROF tool is used to determine the computational time complexity of the application module [47]. Let $Exe(s_j)$ be the execution time predicted for the IoT service s_j. Hence, the total execution time of the application a_i is estimated based on the sum of the execution times of all the IoT services belonging to application a_i, as shown in Eq. (12).

$$ET(a_i)=\sum_{s_j\in a_i}Exe\left(s_j\right) \tag{12}$$

Requests are sent from IoT devices to CFCM. Accordingly, three queues are defined in the ecosystem based on the FCFS. The Q_M queue is related to the incoming traffic flow to the CFCM. The Q_f queue refers to traffic flow in fog nodes, and the Q_R queue refers to traffic flow in the cloud. Based on these three queues, the waiting time of an IoT service can be calculated. We calculate the waiting time in Q_M based on time periods τ. However, the waiting time in Q_f is calculated based on the time between IoT services reaching the fog node, where the input rate is assumed by the Poisson distribution. Let λ be the input rate of requests and μ is the rate of distribution of IoT services by the MADE-k model. Accordingly, $WT\left(s_j^{FN}\right)$ is the waiting time in Q_f associated with the IoT service s_j defined by Eq. (13) [48].

$$WT\left(s_j^{FN}\right)=\frac{1}{\mu_i-\lambda_i}-\frac{1}{\mu_i} \tag{13}$$

Since the cloud has virtually unbounded resources, the waiting time of IoT services can be considered zero for Q_R. Therefore, the execution time of IoT service s_j is calculated based on Eq. (14).

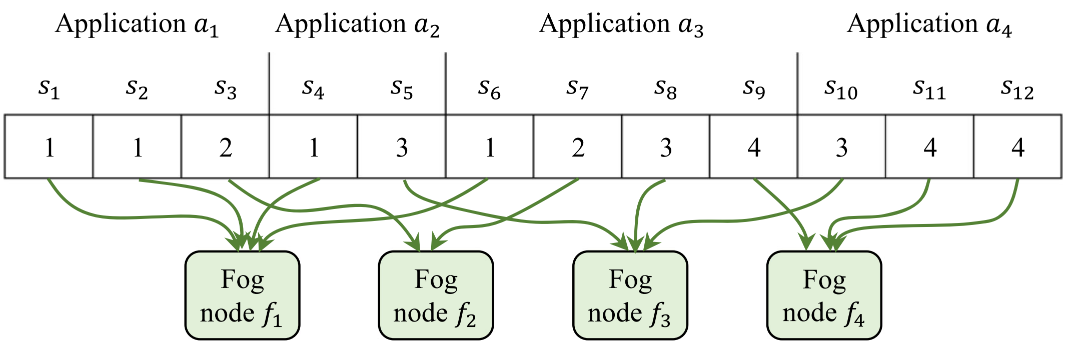

Fig. 7 An example of a solution encoding process

$$WT(s_j) = WT\left(s_j^f\right) + WT\left(s_j^M\right) \tag{14}$$

Hence, the total execution time of the application a_i is estimated based on the sum of the execution times of all the IoT services belonging to application a_i, as shown in Eq. (15).

$$WT(a_i) = \sum_{s_j \in a_i} WT(s_j) \tag{15}$$

The communication time required to transfer data between ecosystem components at different layers is calculated based on the latency of the communication links. Since IoT services are sent by CFCM to the FOCN and then to the fog nodes, the sum of the latencies $l_{o_j}^M$ and $l_{f_i}{}^{o_j}$ is considered as the communication time. The amount of these latencies is negligible, however the latency between components i and j is calculated based on bandwidth ($bw_{i,\,j}$), a constant propagation latency (l_{pd}), and the size of the data associated with the IoT service s_j (ds_{s_j}), as shown in Eq. (16). In addition, Eq. (17) shows the total communication time of the application a_i.

$$WT(a_i) = \sum_{s_j \in a_i} WT(s_j) \tag{16}$$

$$CT(a_i) = \sum_{s_j \in a_i} CT(s_j) \tag{17}$$

Latency (LT): latency factor indicates the time required to execute all services in an application. In the proposed framework, we consider two types of latency for each service: 1) communication link latency and 2) processing latency. Communication link latency include: latency in transmission of service workloads from IoT device to CFCM (l_M^D), latency in transmission of service workloads from CFCM to FOCN (l_O^M), latency in transmission of service workloads from CFCM to fog node (l_f^O), latency in transmission of service workloads from FOCN to NNFD (l_N^O), and latency in transmission of application workloads from CFCM to R-cloud resources (l_R^M). On the other hand, processing latency include: latency processing of service workloads in CFCM (l_M), latency processing of service workloads in FOCN (l_O), latency processing of service workloads in fog node (l_f), latency processing of service workloads in NNFD (l_N), and latency processing of service workloads in R-cloud resources (l_R). In this paper, communication link latency is avoided because communication link latency between ecosystem components is negligible due to the close distance to the network edge [5]. Therefore, the total latency of application a_i is calculated based on Eq. (18).

$$LT(a_i) = \sum_{s_j \in a_i} l_M(s_j) + x_O.l_O(s_j) + x_f.l_f(s_j) + x_N.l_N(s_j) + x_R.l_R(s_j) \tag{18}$$

Where, x_O, x_f, x_N, and x_R represent the binary decision variables for FOCN, fog node, NNFD, and cloud R, respectively.

Each factor is multiplied by a binary decision variable to determine the computing resource assigned to the IoT service s_j. Due to the processing of all services for placement in CFCM, l_M applies to all services. Given the time periods specified, we assume $l_M = \tau$. Also, given the strong computing capacity and rich resources in the cloud, $l_R = 0$ is assumed. The latencies associated with l_O, l_f and l_N are calculated similarly. Here, the latency l_f is calculated according to Eq. (19).

$$l_f(s_j) = \frac{SS_{s_j}.\varphi_f(s_j)}{fr_f} \tag{19}$$

Where, SS_{s_j} is the size of the IoT service s_j based on the number of bits, $\varphi_f(s_j)$ is the number of cycles required by the processor per fog node to execute the IoT service s_j, and fr_f is the frequency of the fog node.

Fog resources utilization (*FU*): This factor indicates the number of FCL locations utilized by IoT services. Therefore, maximizing the fog resources utilization to reduce energy consumption is emphasized. This factor is calculated based on the communication latency and placement associated with deploying the IoT service on FOCN, fog node, NNFD, or cloud R, as these can lead to breach of application deadlines. Eq. (20) calculates the fog resources utilization for the application a_i.

$$FU(a_i) = P(a_i) \sum_{s_j \in a_i} \left[x_O(s_j) + x_f(s_j) + x_N(s_j) + x_R(s_j)\right] \tag{20}$$

Where, $P(a_i)$ is the priority of the application a_i, which is calculated by Eq. (3). Also, x_O, x_f, x_N, and x_R are the binary decision variables associated with the IoT service s_j for FOCN, fog node, NNFD, and cloud R, respectively.

5.3 The Process of Population Evolution

After creating the initial population, the fitness of each solution is calculated. According to ODMA, solutions are sorted so that $soft_1$ is the best and $soft_{N_P}$ is the worst solution based on the fitness function. Then, z solutions with the highest fitness function are selected as the leading solutions, while other solutions are promising. Therefore, $LS = \{soft_1, soft_2, \ldots, soft_z\}$ a set of leading solutions and $PS = \left\{soft_{z+1}, soft_{z+2}, \ldots, soft_{N_P}\right\}$ is a set of promising solutions. Any solution as a software may develop and evolve over time or become obsolete and removed. The evolutionary process in ODMA consists of three main steps: (i) moving towards leading solutions, (ii) evolution of the leading solutions based on their history, and (iii) removing weak solutions and forking new solutions based on the leading solutions [15]. The details of the evolutionary steps are described below.

Moving toward the leading solutions: In this step, each promising solution is developed based on a leading solution. ODMA uses an open-source model-based mechanism to select the leading solution. In this mechanism, each solution has a score, where $score_i$ refers to the score of the i^{th} solution. Initially, the score is zero for all solutions, $score_i = 0, \quad \forall i = 1, 2, \ldots, N_P$. The solution scores are updated during the optimization process as follows:

- Adding a unit to the score of a solution that has improved its position based on the fitness function.
- Adding a unit to the score of the leading solution that has succeeded in improving the position of a promising solution.

According to the solution scores, the promising solution j^{th} selects a solution from among the z leading solutions for its development. The leading solution index selected by Eq. (21) is defined so that the promising solution score is effective in selecting the leading solution [15].

$$\phi = argRand\left\{1, \left((z+1) - \left(\frac{score_j}{\sum_{i=1}^{z} score_i}\right)(2 \times z)\right)\right\} \tag{21}$$

Where, ϕ the leading solution index is selected and *argRand* is a random function for selecting the solution index within the specified range.

Next, the promising solution $soft_j$ moves and develops based on the leading solution $soft_\phi$. Here, the moving process is performed in a discrete form with the definition of the probabilistic variable $\rho \in [0-1]$. Here, each element of $soft_j$ changes with probability ρ and with respect to its corresponding element in $soft_\phi$. Basically, $\rho = 1$ changes all elements $soft_j$ to $soft_\phi$ and $\rho = 0$ makes no change to $soft_j$. Figure 8 shows an example of moving towards the leading solutions, where changes have been made to elements 4, 5 and 10. These changes are made on the assumption that there is a probability of ρ for these elements.

Evolution of the leading solutions based on their history: In this step, each leading solution evolves based on its history, including the current position ($soft_{cur}$) and the old position ($soft_{old}$). $soft_{new}$ as a new position of a leading solution and is calculated by Eq. (22).

$$soft_{new}^{k} = \left\| \frac{soft_{cur}^{k} + soft_{old}^{k})}{2} \right\| + \Delta \quad \forall k = 1, 2, \ldots, r \tag{22}$$

Where, $\Delta \in \{-1, 0, 1\}$ is a random number in the defined range to create variation in the solution, and $\|*\|$ is a rounding function. k is known as the index of an element in the solution, and r refers to the solution size (i.e., the number of fog services).

According to the solution encoding, each element as the position of a service in the solution cannot be less than 1 and greater than r. Therefore, this constraint is checked after applying this step to the solution and the violations are corrected with the value of the previous state.

Removing weak solutions and forking new solutions: In this step, n_ω weak solution is removed after each P_{iter} iteration and replaced with new solutions, where $n_\omega \leq z$. Weak solutions are determined based on the suitability (i.e., average score and fitness function). For each solution removed, a new solution is forking of the leading solutions. Here, n_ω the leading solution with the highest suitability for forking is selected. Let $soft_L$ be a leading solution selected. The new forked solution from $soft_L$ is calculated by Eq. (23).

$$soft^k_{new} = \begin{cases} soft^k_L + \Delta & \rho \geq rand() \\ soft^k_L & otherwise \end{cases} \quad \forall k = 1, 2, \ldots, r \tag{22}$$

Where, $\Delta \in \{-1,\ 0,\ 1\}$ is a random number in the defined range and $\rho \in [0 - 1]$ determines the probability of change in each element of the solution.

After the end of an iteration in ODMA, the new population has evolved compared to the old population. This process causes the population to converge towards the optimal points in successive iterations. We now analyze the stop criteria for the practical implementation of the algorithm, as it is impossible to execute for infinite time. In this paper, two stop conditions are defined for ODMA: the conventional iteration method and the achievement of convergence. In the first method, the number of iterations is used as a stop condition, where the maximum number of iterations is defined as Max_{iter}. The second method can be defined by observing the change in the most optimal solution of each iteration. Convergence occurs when no change in the optimal fitness of the solution is reported in several consecutive iterations.

6 Simulation Results

In this section, the proposed algorithm (i.e., FSP-ODMA) is compared to evaluate the performance against other similar methods. This comparison is performed with FSP-GWO [2], FSP-CSA [5] and FSP-PSO [7] methods, where they solve the FSP problem with GWO, CSA and PSO algorithms, respectively. In addition, all comparable methods, like the proposed algorithm, formulate the FSP problem as a multi-objective problem. The source code link of all these algorithms is available as open-source in related papers. The simulation was performed on a testbed based on the iFogSim28 simulator [36].

All experiments run on an HP ENVY Laptop with an Intel® Core™ i7 processor with a speed of 4.7 GHz and 16 GB of RAM. Requests sent from IoT devices are received by the CFCM, and based on the MADE-k model, the FSP-ODMA automatically decides on the services placement. In this regard, Google Cloud Platform is used for cloud services and Raspberry Pi 3b + Hypriot OS is used for fog nodes [36]. FCL is configured based on the Spring and Java 8 frameworks. In addition, communication exchanges between ecosystem components are handled by JSON and REST API packages. This section discusses experimental setup, performance metrics, and comparison results.

6.1 Experimental Setup

The simulation is performed on a virtual testbed platform and the experimental setup are adjusted accordingly [48]. In this study, different algorithms are evaluated based on 5 consecutive time periods (i.e., τ = 5). The number of services requested in each period is according to Table 4. Tables 5 show the various parameters and resource details for cloud servers and fog nodes. Expenses related to storage, processor and memory are in dollars per Billing Time Unit (BTU). The simulation is performed for 3 fog domains, where the number of fog nodes in the domains are 4, 8 and 12, respectively. Since FOCN is randomly selected from the fog nodes, the processing cost in FOCN and other fog nodes is the same.

Requests are sent to the FCL as real-time microservice-based applications by IoT devices. Three microservice-based applications were used to evaluate FSP-ODMA compared to FSP-GWO, FSP-CSA and FSP-PSO methods. These applications include Smart-health, Teastore and Hipstershop. Smart-health is a microservice application based on smart healthcare with 4 services that uses ECG sensor for continuous patient monitoring. The application includes modules for extracting anomalies, analyzers and persistence that process data locally and without latency in the FCL. Teastore is an e-commerce industry application with 6 services that has been used by various researchers to evaluate the FSP problem. Hipstershop is also an e-commerce-based application with 10 services that give online customers features such as item browsing, adding items to the cart, and completing the shopping process. Application specifications include the time deadline for execution, details of services, required resources (i.e., storage, processor and memory) and services field (mandatory or non-mandatory) are given in Table 6. Time deadlines are expressed in seconds (s), storage and memory resources in mebibytes (M.B), and processor resources in millicores (M.C).

All the comparative methods (i.e., FSP-GWO, FSP-CSA and FSP-PSO) together with the proposed algorithm (i.e., FSP-ODMA) use an evolutionary approach to solve the FSP problem. The simulation parameters for

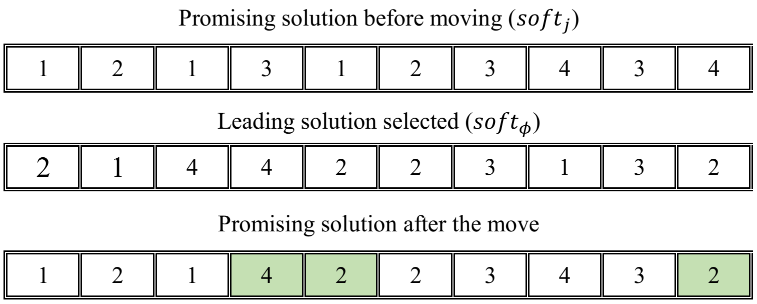
Promising solution before moving ($soft_j$)
1 2 1 3 1 2 3 4 3 4
Leading solution selected ($soft_\phi$)
2 1 4 4 2 2 3 1 3 2
Promising solution after the move
1 2 1 4 2 2 3 4 3 2

Fig. 8 An example of moving towards the leading solutions

all of them are set as follows: $N_P = 25$, $Max_{iter} = 100$, $\tau = 3s$, $l_M^D = l_R^M = 1s$, $l_{f_i}{}^{o_j} = l_{o_j}^M = 0.2s$, $l_N^{o_j} = 0.3s$.

In addition, some specific parameters related to FSP-ODMA are set as follows: $z = 10$, $\rho = 0.25$, $n_\omega = 3$, $P_{iter} = 25$.

6.2 Performance Metrics

The proposed algorithm is evaluated based on different performance metrics compared to other similar methods in the FSP problem. These metrics include the number of services performed, the number of failed services, the remaining services, convergence, runtime, service cost, energy consumption, response time, latency, and the fog resource utilization, which are discussed below.

Number of services performed This metric refers to the total number of services performed before the end of the specified time periods. Therefore, maximizing this metric is emphasized.

Number of failed services This metric refers to the number of services that failed to execute on the cloud or fog due to a time deadline violation. Therefore, minimizing this metric is emphasized.

Remaining services This metric refers to the number of services that failed to execute after the specified time periods. Therefore, minimizing this metric is emphasized.

Convergence This metric is related to evolutionary algorithms that stop evolution because exactly every solution is the identical in a population. An algorithm usually converges when there is no significant improvement in population fitness values from one iteration to the next. Convergence is measured by the fitness function relative to iteration, and the goal is to achieve maximum fit in the least iteration.

Runtime This metric for various algorithms refers to the duration of the FSP problem-solving policy, which is expressed in seconds.

Service cost Cost refers to the monetary that the user pays to application execution, which includes the cost of communications and calculations. The service cost is defined based on Eq. (5), where its minimizing is emphasized.

Energy consumption Sending a request by a user leads to the energy consumption of the entities involved in the ecosystem. In general, energy consumption depends on the type of data to be processed by the nodes and the

Table 4 Number of services requested in 5 consecutive time periods

Time period (τ)	Time (s)	Number of services requested
1	8	71
2	16	48
3	24	48
4	32	46
5	40	87

Table 5 Various parameters and resource details in cloud and fog

Parameters	Cloud computing	Fog computing
Probability of failure	0.003	0.00002
Storage (GB)	4 $	6 $
Processor (M.C)	4 $	5 $
Memory (GB)	4 $	6 $

Table 6 Specifications of Smart-health, Teastore and Hipstershop applications

Application	Time deadline (s)	Number of requests	Service name	Storage (M.B)	Processor (M.C)	Memory (M.B)	Service field
Smart-health	5	15	Analyser	100	200	100	Not mandatory
			Feature extractor	80	200	80	Mandatory
			Persistance	200	150	200	Not mandatory
			Client	200	200	100	Mandatory
Teastore	7	20	Auth	80	100	80	Mandatory
			Image	150	100	150	Not mandatory
			Persistence	100	200	100	Not mandatory
			Recommender	100	100	80	Mandatory
			Registry	150	100	150	Mandatory
			Webui	160	200	80	Not mandatory
Hipstershop	18	12	Shipping	64	100	64	Mandatory
			Recommend	220	100	220	Not mandatory
			Product	64	100	64	Mandatory
			Payment	64	100	64	Mandatory
			Frontend	64	100	64	Mandatory
			Email	64	100	64	Not mandatory
			Currency	64	100	64	Not mandatory
			Checkout	64	100	64	Mandatory
			Cart	128	200	64	Mandatory
			Adservice	180	200	180	Not mandatory

communications required to send the request/response. Energy consumption is defined based on Eq. (6), where its minimizing is emphasized.

Response time This metric refers to the time elapsed from the user's request to the receipt of results, which includes waiting time and service time. Response time can vary depending on the bandwidth of the network, the type and number of user requests. This metric is defined based on Eq. (11), where its minimizing is emphasized.

Latency A latency is the time it takes for a request to be sent to a processing component. This metric includes the time required to all services execution of application along with the time required for communication. Latency is defined by Eq. (18), where its minimizing is emphasized.

Fog resource utilization This metric means deploying the maximum number of services on fog nodes. Therefore, the utilization of more fog resources will increase the performance of the service. This metric is defined based on Eq. (20), where its maximizing is emphasized.

6.3 Comparison and Discussion

In this section, the methods of FSP-GWO, FSP-CSA, FSP-PSO and also FSP-ODMA as the proposed algorithm are compared and evaluated based on different performance metrics. Here, 300 services from 47 application applications enter the ecosystem for deployment in 5 different time periods. These requests consist of 3 different applications that must be placed and executed on 3 fog domains with the number of nodes 4, 8 and 12, respectively. An overview of FSP-ODMA results is reported in Table 7. These results are reported for each application based on the sum of all requests.

In addition, the statistical analysis of the proposed algorithm using paired t-test for different metrics is presented so that the results show that the proposed algorithm has a significant level in the results. Statistical parameters include mean and standard deviation (SD) of *t*-value and *p*-value. In this analysis, a significance level of $p < 0.05$ is considered for paired t-test. Therefore, when the p value is lower than 0.05, it can be concluded that the proposed algorithm for a particular criterion has a significant level.

According to Table 6, there are a total of 300 services for the three defined applications that enter the ecosystem in 5 different time periods. The aim of any placement algorithm is to deploy these services on fog nodes, NNFD or cloud, where the time deadline of the applications is not violated. Services that are executed before the time deadline are known as services performed. Failed services are services that could not be executed due to a time deadline violation. On the other hand, the resources required by some services may not be available in the specified time period and the placement of these services should be done in the next time period. These services are known as the remaining services. Figure 9 shows the results associated with the different methods for the metrics of number of services performed, number of failed services, and number of remaining services.

As illustrated, FSP-ODMA is superior to FSP-GWO, FSP-CSA and FSP-PSO methods with 271 services performed, 24 failed services and 5 remaining services by 8.4%, 4.9% and 25.6%, respectively. Prioritization to determine the order of application execution leads to the reduction of failed services. Thus, the superiority of FSP-ODMA can be explained by the fact that fog nodes select requests with minimum time deadline to execution based on priorities.

Convergence refers to the limitations of a process and can be a useful analytical tool when evaluating the expected performance of an optimization algorithm. In general, optimization is a process for producing candidate solutions, and convergence is a stable point at the end of the process that no further change is expected. All the methods compared use evolutionary approaches to solve the FSP problem, and since convergence is a phenomenon in evolutionary computations, it can compare existing methods based on this metric. In Fig. 10, the convergence for different methods in 100 iterations is calculated based on the fitness function.

As illustrated, FSP-ODMA has a better convergence rate than other methods, although this superiority is slight. FSP-ODMA converged to 853.5 in iteration 43, and the closest method to it is FSP-GWO, which converged to 854.9 in iteration 36. Due to the high computational complexity of the FSP-CSA in the optimization process, poor convergence was expected for this method.

Although the solution provided by FSP-ODMA has a better convergence rate than other methods, but the complexity of the methods in the optimization process must also be analyzed. Because most application require real-time resources and the execution time of the methods is important in solving the FSP problem due to the dynamic environment. Figure 11 shows the runtime of different methods for deciding on the IoT services placement during optimization. The results clearly show the inefficiency of FSP-CSA, because the evolution process of this method has many steps and, in each iteration, there is a need to calculate the fitness function sequentially. However, the runtime of the other methods is almost the same, although the FSP-GWO has a shorter runtime. In this experiment, the proposed algorithm has the best performance with an average runtime of 14.1 s after FSP-GWO.

Compared to other methods, FSP-ODMA has resulted in a minimized fitness function as well as better convergence speed. Therefore, the IoT services placement by FSP-ODMA is predictable with the lowest service cost. Figure 12 shows the results related to service costs for different methods after the end of time periods. Obviously, the service cost in the proposed algorithm is 8.2%, 24.6% and 21.7% better than the solutions obtained by FSP-GWO, FSP-CSA and FSP-PSO, respectively. This shows that FSP-ODMA can reduce the cost of IoT applications by an average of 18.2%. It is worth noting that with the IoT services placement on fog nodes, there is no need for any rental costs for cloud services. In addition, the proposed algorithm improves the service cost by considering the QoS requirements and the defined objective function.

Figure 13 shows the energy consumption of fog nodes after 5 time periods. FSP-ODMA with 21 J has the lowest energy consumption compared to other methods. After the proposed algorithm, FSP-PSO and FSP-GWO with 2.3 J have worthy energy efficiency and the worst results are related to FSP-CSA. In general, the proposed algorithm consumes 9.5% less energy to solve the FSP problem than FSP-PSO and FSP-GWO and 28.6% compared to FSP-CSA.

Table 7 Overview of FSP-ODMA results for Smart-health, Teastore and Hipstershop applications

Performance metrics	Smart-health	Teastore	Hipstershop
Number of services performed	54	107	111
Number of remaining services	6	13	9
Convergence	40	35	51
Execution time (s)	8.7	13.1	19.3
Service cost ($)	6.8	7.3	13. 4
Energy consumption (J)	1.05	1.36	2.06
Response time (s)	4.38	4.16	6.73
Latency (s)	0.17	0.18	0.31
Utilization of fog resources (%)	95.47	91.14	87.09
t-value	3.19	2.25	2.34
p-value	0.021	0.025	0.014

Response time refers to the total time elapsed from sending the request to receiving the response. Here the response time is measured in seconds (s) and the purpose of IoT service placement methods is to minimize it. Response time is calculated based on the sum of communication time, waiting time and execution time factors. Each application has a time deadline within which the user expects a response, and placement methods must ensure that resources are provided for IoT applications with tolerable response time. Figure 14 shows the response time based on the different time periods defined in Table 4 for all requests. These results are reported for each method based on the average response time on all applications.

As illustrated, FSP-ODMA has a better response time than all other methods in all comparisons and guarantees a tolerable response time. As time periods increase, access to the number of fog nodes also increases and response times for all methods decrease. Simulation observations show that in all cases the response time for the HipsterShop application is higher than for the Smart-Health and TeaStore applications, because the HipsterShop has more service modules. On average, the response time for FSP-ODMA is 5.64 s and it performs better with 2.93 s, 10.05 s and 8.84 s compared to FSP-GWO, FSP-CSA and FSP-PSO methods, respectively.

In the following, different methods are evaluated and compared based on latency performance metric. Latency in service execution indicates the amount of time spent on a service on the network, where it often depends on the latency of communication links in the network. Figure 15 shows the results of this comparison.

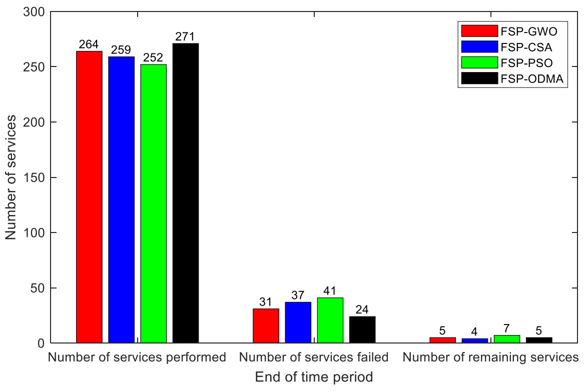

Fig. 9 Comparison of services performed, failed and remaining in different methods

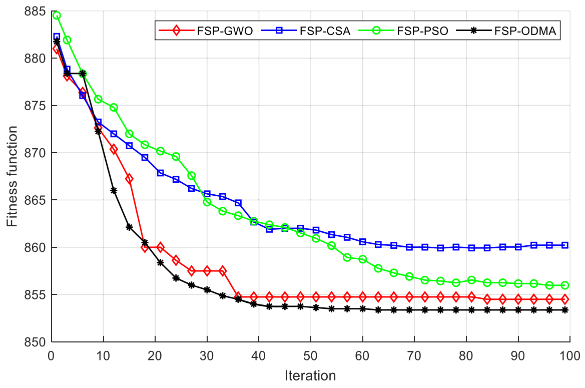

Fig. 10 Comparison of convergence speed in different methods

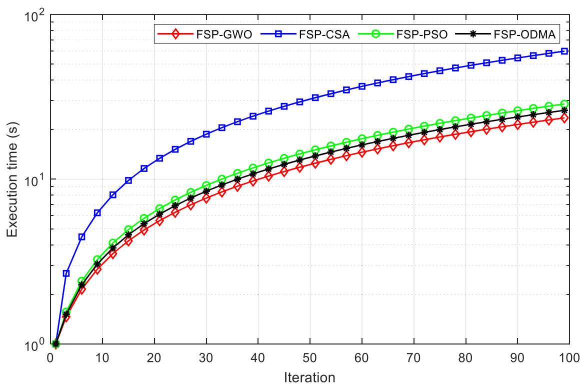

Fig. 11 Comparison of runtime in different methods

As illustrated, FSP-ODMA has less latency than other methods. These results are reported for 5 time periods (1 to 40s) for each service. The reason for the increase in latency for some services is that more requests are sent to the fog nodes at certain time periods. However, most services have relatively little latency because FSP-ODMA has been able to handle most requests in the FCL and a small number of requests have been sent to the cloud. Since most requests are processed by the proposed algorithm in the FCL, the average latency remains at 0.22 s. Meanwhile, the average latency of all services for FSP-GWO, FSP-CSA and FSP-PSO methods is 0.46 s, 0.76 s and 0.98 s, respectively.

The metric results of the fog resource utilization for different methods are presented in Fig. 16. This comparison was calculated and reported during 5 time periods (1 to 40s). According to the results, SFP-ODMA has increased the utilization rate of fog resources compared to other methods. As illustrated, SFP-ODMA improved this metric by 0.43% compared to SFP-GWO, 8.37% compared to SFP-CSA and 6.26% compared to SFP-PSO. These results show that the proposed algorithm provides the ability to suitable IoT services placement by increasing the fog resource utilization through the autonomous MADE-k model. Therefore, the obtained results indicate the acceptable performance of the proposed algorithm in terms of the fog resource utilization.

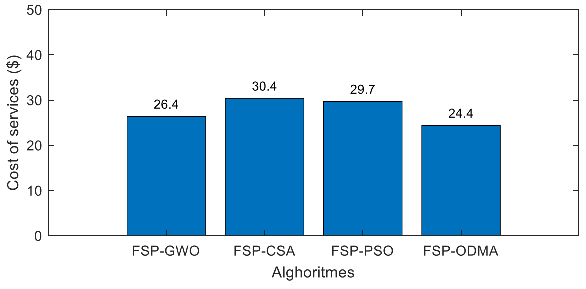

Fig. 12 Comparison of service costs in different methods

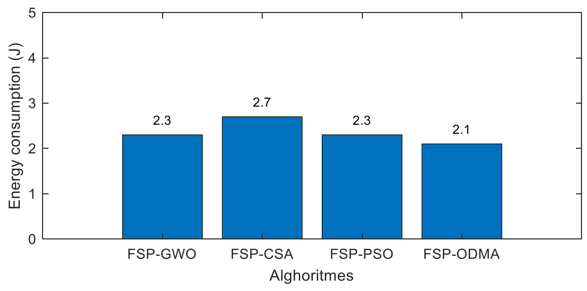

Fig. 13 Comparison of energy consumption in different methods

In general, the proposed algorithm (i.e., SFP-ODMA) offers better results in different performance metrics than other comparable methods (i.e., FSP-GWO, FSP-CSA and FSP-PSO). On average, SFP-ODMA results are 4.98%, 12.88% and 12.15% superior to FSP-GWO, FSP-CSA and FSP-PSO, respectively. In addition, comparing SFP-ODMA with other methods, it can be concluded that: (1) the solutions provided by FSP-ODMA have a better variety than other methods due to different steps in evolution and consideration of two types of leading and promising solutions, (2) the proposed algorithm has a better distribution in creating solutions, which improves the convergence speed, and (3) FSP-ODMA seems to work better in environments with a large number of nodes than other methods, especially FSP-COA. However, the comparison with runtime shows the superiority of FSP-GWO over FSP-ODMA with a slight difference.

7 Conclusion

Fog computing can be defined as a distributed computing paradigm that provides cloud services at the edge of the network. Therefore, due to their proximity to final devices, fog computing has the potential to provide service to latency-sensitive

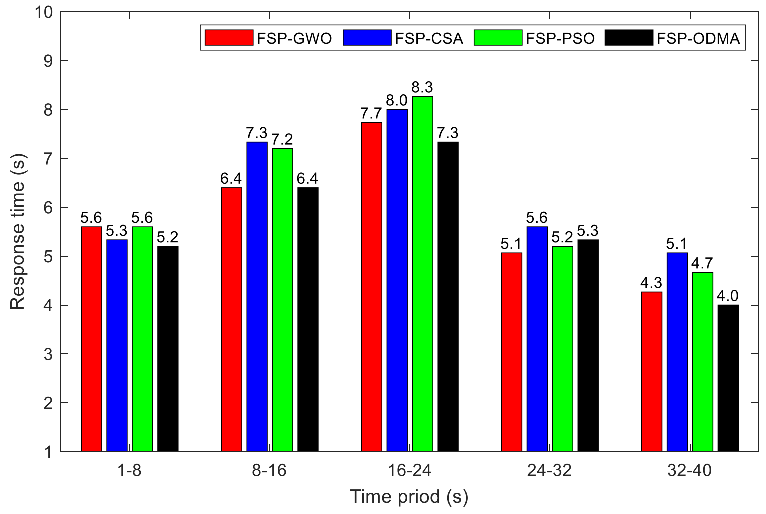

Fig. 14 Comparison of response times in different methods

applications. With the advent of fog computing, the challenge of deploying IoT services on fog nodes has attracted much attention, where it is often formulated as a multi-objective optimization problem. In this paper, an autonomous method of IoT services placement is proposed based on a conceptual framework consisting of four phases of monitoring, analysis, decision-making and execution control loop with a shared knowledge base (MADE-k). First, service details are monitored from IoT applications and fog resource status. Then, the services are analyzed and prioritized according to the time deadline. After that, the ODMA metaheuristic algorithm decides on the deployment of IoT services on fog nodes to optimize objectives such as service cost, energy consumption, response time, latency, and fog resource utilization. Finally, the decisions made are executed in a fog environment. The proposed policy improves service availability and QoS satisfaction by bringing cloud services and resources closer to the end devices. Fog landscape simulation using iFogSim shows that the proposed algorithm (i.e., FSP-ODMA) performs better than its counterparts such as FSP-GWO, FSP-CSA, and FSP-PSO by solving an optimization problem based on various performance metrics. For future work, we will develop our proposed policy based on the theory of subjective logic to ensure the reliability and safety of interactions in the fog environment in addition to latency.

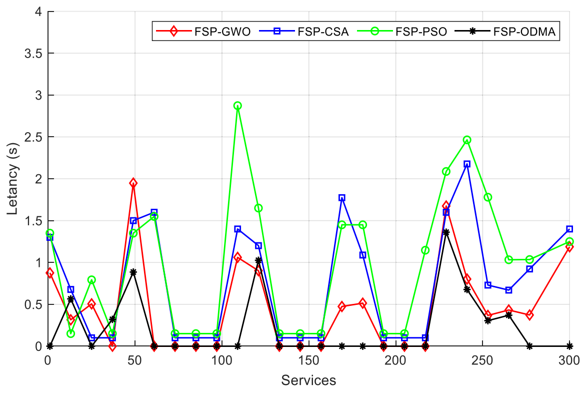

Fig. 15 Comparison of service latency in different methods

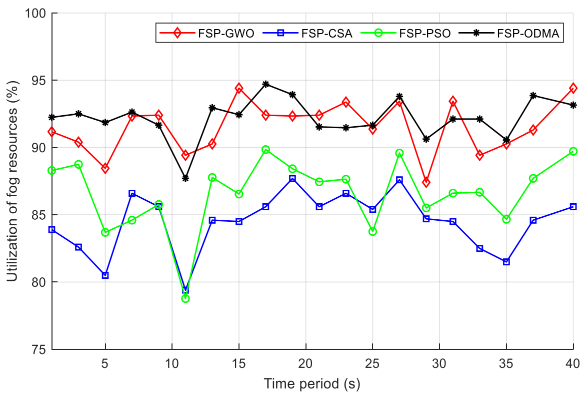

Fig. 16 Comparison of the fog resource utilization in different methods

Acknowledgements 1. The research results of the Ministry of Education's 2021 industry university cooperation collaborative education project "Research and practice of online and offline mixed teaching mode based on OBE concept for course Database Principle and Application" (Project NO.202101087015).

2. The research results of Natonal Vocational Education teaching reform research project: "Research on the path of Vocational Colleges to improve morality and build people from the perspective of curriculum ideology and politics" (Project No.2020QJG036).

Author Contribution Defu Zhao, Qunying Zou, Milad Boshkani Zadeh conducted this research. Defu Zhao: Methodology, Software, Validation, Writing original draft. Qunying Zou: Conceptualization, Supervision, Writing review & editing, Formal analysis, Project administration. Milad Boshkani Zadeh: Investigation, Resources, Data curation, Visualization.

Funding This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Data Availability Data sharing not applicable to this article as no datasets were generated or analyzed during the current study.

Declarations.

Conflict of Interest We certify that there is no actual or potential conflict of interest in relation to this article.

References

1. Nižetić, S., Šolić, P., González-de, D.L.D.I., Patrono, L.: Internet of things (IoT): opportunities, issues and challenges towards a smart and sustainable future. J. Clean. Prod. **274**, 122877 (2020)
2. Salimian, M., Ghobaei-Arani, M., Shahidinejad, A.: Toward an autonomic approach for internet of things service placement using gray wolf optimization in the fog computing environment. Software: Practice and Experience. **51**(8), 1745–1772 (2021)
3. Rezaeipanah, A., Mojarad, M., Fakhari, A.: Providing a new approach to increase fault tolerance in cloud computing using fuzzy logic. Int. J. Comput. Appl. **44**(2), 139–147 (2020)
4. Skarlat, O., Nardelli, M., Schulte, S., Borkowski, M., Leitner, P.: Optimized IoT service placement in the fog. SOCA. **11**(4), 427–443 (2017)
5. Liu, C., Wang, J., Zhou, L., Rezaeipanah, A.: Solving the multi-objective problem of IoT service placement in fog computing using cuckoo search algorithm. Neural. Process. Lett. 1–32 (2022) in press
6. Berahmand, K., Samadi, N., Sheikholeslami, S.M.: Effect of rich-club on diffusion in complex networks. International Journal of Modern Physics B. **32**(12), 1850142 (2018)
7. Salimian, M., Ghobaei-Arani, M., Shahidinejad, A.: An evolutionary multi-objective optimization technique to deploy the IoT Services in fog-enabled Networks: an autonomous approach. Appl. Artif. Intell. (2022). https://doi.org/10.1080/08839514.2021.2008149
8. Hosseinzadeh, M., Masdari, M., Rahmani, A.M., Mohammadi, M., Aldalwie, A.H.M., Majeed, M.K., Karim, S.H.T.: Improved butterfly optimization algorithm for data placement and scheduling in edge computing environments. Journal of Grid Computing. **19**(2), 1–27 (2021)
9. Rezaeipanah, A., Nazari, H., Ahmadi, G.: A hybrid approach for prolonging lifetime of wireless sensor networks using genetic algorithm and online clustering. J. Comput. Sci. Eng. **13**(4), 163–174 (2019)
10. Berahmand, K., Bouyer, A.: A link-based similarity for improving community detection based on label propagation algorithm. J. Syst. Sci. Complex. **32**(3), 737–758 (2019)
11. Aslanpour, M.S., Dashti, S.E., Ghobaei-Arani, M., Rahmanian, A.A.: Resource provisioning for cloud applications: a 3-D, provident and flexible approach. J. Supercomput. **74**(12), 6470–6501 (2018)
12. Selimi, M., Cerdà Alabern, L., Freitag, F., Veiga, L., Sathiaseelan, A., Crowcroft, J.: A lightweight service placement approach for community network micro-clouds. Journal of Grid Computing. **17**(1), 169–189 (2019)
13. Ghobaei-Arani, M., Shahidinejad, A.: An efficient resource provisioning approach for analyzing cloud workloads: a metaheuristic-based clustering approach. J. Supercomput. **77**(1), 711–750 (2021)
14. Goudarzi, M., Palaniswami, M., Buyya, R.: A distributed application placement and migration management techniques for edge and fog computing environments. In: 2021 16th Conference on Computer Science and Intelligence Systems (FedCSIS), pp. 37–56. IEEE, Sofia, Bulgaria (2021)
15. Hajipour, H., Khormuji, H.B., Rostami, H.: ODMA: a novel swarm-evolutionary metaheuristic optimizer inspired by open-source development model and communities. Soft. Comput. **20**(2), 727–747 (2016)
16. Rezazadeh, Z., Rahbari, D., Nickray, M.: Optimized module placement in IoT applications based on fog computing. In: Electrical Engineering (ICEE), pp. 1553–1558. IEEE, Mashhad, Iran (2018)
17. Rezaeipanah, A., Amiri, P., Nazari, H., Mojarad, M., Parvin, H.: An energy-aware hybrid approach for wireless sensor networks using re-clustering-based multi-hop routing. Wirel. Pers. Commun. **120**(4), 3293–3314 (2021)
18. Holland, J.: Outline of control parameters for genetic algorithms. Journal of Association for Computing Machinery. **3**, 297–314 (1962)
19. Kennedy, J., Eberhart, R.: Particle Swarm Optimization. In Proceedings of ICNN'95-International Conference on Neural Networks, vol. 4, pp. 1942–1948. IEEE, Perth, WA, Australia (1995)
20. Geem, Z.W., Kim, J.H., Loganathan, G.V.: A new heuristic optimization algorithm: harmony search. *Simulation*. ***76***(2), 60–68 (2001)
21. Mirjalili, S., Lewis, A.: The whale optimization algorithm. Adv. Eng. Softw. **95**, 51–67 (2016)
22. Yang, X.S., Deb, S.: Cuckoo search via Lévy flights. In: 2009 World Congress on Nature & Biologically Inspired Computing (NaBIC), pp. 210–214. IEEE, Coimbatore, India (2009)
23. Mirjalili, S., Mirjalili, S.M., Lewis, A.: Grey wolf optimizer. Adv. Eng. Softw. **69**, 46–61 (2014)
24. Yang, L., Cao, J., Liang, G., Han, X.: Cost aware service placement and load dispatching in mobile cloud systems. IEEE Trans. Comput. **65**(5), 1440–1452 (2015)

25. Alghamdi, A., Alzahrani, A., Thayananthan, V.: Execution time and power consumption optimization in fog computing environment. International Journal of Computer Science & Network Security. **21**(1), 137–142 (2021)
26. Bhatia, M., Sood, S.K., Kaur, S.: Quantum-based predictive fog scheduler for IoT applications. Comput. Ind. **111**, 51–67 (2019)
27. Dai, Y., Xu, D., Maharjan, S., Zhang, Y.: Joint computation offloading and user association in multi-task mobile edge computing. IEEE Trans. Veh. Technol. **67**(12), 12313–12325 (2018)
28. Tavousi, F., Azizi, S., Ghaderzadeh, A.: A fuzzy approach for optimal placement of IoT applications in fog-cloud computing. Clust. Comput. **25**, 303–320 (2021)
29. Gill, S.S., Tuli, S., Xu, M., Singh, I., Singh, K.V., Lindsay, D., et al.: Transformative effects of IoT, Blockchain and artificial intelligence on cloud computing: evolution, vision, trends and open challenges. Internet of Things. **8**, 100118 (2019)
30. Hussein, M.K., Mousa, M.H.: Efficient task offloading for IoT-based applications in fog computing using ant colony optimization. IEEE Access. **8**, 37191–37201 (2020)
31. Nayeri, Z.M., Ghafarian, T., Javadi, B.: Application placement in fog computing with AI approach: taxonomy and a state of the art survey. J. Netw. Comput. Appl. **185**, 103078 (2021)
32. Huang, T., Lin, W., Xiong, C., Pan, R., Huang, J.: An ant colony optimization-based multiobjective service replicas placement strategy for fog computing. IEEE Transactions on Cybernetics. **51**(11), 5595–5608 (2020)
33. Gill, M., Singh, D.: ACO based container placement for CaaS in fog computing. Procedia Computer Science. **167**, 760–768 (2020)
34. Ghalehtaki, R.A., Kianpisheh, S., Glitho, R.: A bee colony-based algorithm for micro-cache placement close to end users in fog-based content delivery networks. In: 2019 16th IEEE Annual Consumer Communications & Networking Conference (CCNC), pp. 1–4. IEEE, Las Vegas, NV, USA (2019)
35. Sharma, S., Saini, H.: Efficient solution for load balancing in fog computing utilizing artificial bee Colony. International Journal of Ambient Computing and Intelligence (IJACI). **10**(4), 60–77 (2019)
36. Nabavi, S.S., Gill, S.S., Xu, M., Masdari, M., Garraghan, P.: TRACTOR: traffic-aware and power-efficient virtual machine placement in edge-cloud data centers using artificial bee colony optimization. Int. J. Commun. Syst. **35**(1), e4747 (2022)
37. Javanmardi, S., Shojafar, M., Persico, V., Pescapè, A.: FPFTS: a joint fuzzy particle swarm optimization mobility-aware approach to fog task scheduling algorithm for internet of things devices. Software: Practice and Experience. **51**(12), 2519–2539 (2021)
38. Djemai, T., Stolf, P., Monteil, T., Pierson, J.M.: 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), pp. 32–40. IEEE, Amsterdam, Netherlands (2019)
39. Baburao, D., Pavankumar, T., Prabhu, C.S.R.: Load balancing in the fog nodes using particle swarm optimization-based enhanced dynamic resource allocation method. Appl. Nanosci. (2021). https://doi.org/10.1007/s13204-021-01970-w
40. Reddy, K.H.K., Luhach, A.K., Pradhan, B., Dash, J.K., Roy, D.S.: A genetic algorithm for energy efficient fog layer resource management in context-aware smart cities. Sustain. Cities Soc. **63**, 102428 (2020)
41. Maia, A.M., Ghamri-Doudane, Y., Vieira, D., de Castro, M.F.: An improved multi-objective genetic algorithm with heuristic initialization for service placement and load distribution in edge computing. Comput. Netw. **194**, 108146 (2021)
42. Bourhim, E.H., Elbiaze, H., Dieye, M.: Inter-container communication aware container placement in fog computing. In: 2019 15th International Conference on Network and Service Management (CNSM), pp. 1–6. IEEE, Halifax, NS, Canada (2019)
43. Hussain, M.M., Beg, M.S.: CODE-V: multi-hop computation offloading in vehicular fog computing. Futur. Gener. Comput. Syst. **116**, 86–102 (2021)
44. Sami, H., Mourad, A., El-Hajj, W.: Vehicular-OBUs-as-on-demand-fogs: resource and context aware deployment of containerized micro-services. IEEE/ACM Trans. Networking. **28**(2), 778–790 (2020)
45. Nardelli, M., Cardellini, V., Grassi, V., Presti, F.L.: Efficient operator placement for distributed data stream processing applications. IEEE Transactions on Parallel and Distributed Systems. **30**(8), 1753–1767 (2019)
46. Gasmi, K., Dilek, S., Tosun, S., Ozdemir, S.: A survey on computation offloading and service placement in fog computing-based IoT. J. Supercomput. **78**, 1983–2014 (2021)
47. Bao, L., Wu, C., Bu, X., Ren, N., Shen, M.: Performance modeling and workflow scheduling of microservice-based applications in clouds. IEEE Transactions on Parallel and Distributed Systems. **30**(9), 2114–2129 (2019)
48. Paul Martin, J., Kandasamy, A., Chandrasekaran, K.: CREW: cost and reliability aware eagle-whale optimiser for service placement in fog. Software: Practice and Experience. **50**(12), 2337–2360 (2020)
49. Zhang, G., Shen, F., Liu, Z., Yang, Y., Wang, K., Zhou, M.T.: FEMTO: fair and energy-minimized task offloading for fog-enabled IoT networks. IEEE Internet Things J. **6**(3), 4388–4400 (2018)