Hindawi
Mobile Information Systems
Volume 2020, Article ID 8830294, 22 pages
https://doi.org/10.1155/2020/8830294

Research Article

A Novel Genetic Service Function Deployment Management Platform for Edge Computing

David Chunhu Li,[1] **Bo-Hun Chen**,[2] **Chia-Wei Tseng**,[2] **and Li-Der Chou**[2]

[1]*Information Technology Management Program, Ming Chuan University, Taoyuan 333, Taiwan*
[2]*Department of Computer Science and Information Engineering, National Central University, Taoyuan 32001, Taiwan*

Correspondence should be addressed to Li-Der Chou; cld@csie.ncu.edu.tw

Received 29 May 2020; Revised 27 August 2020; Accepted 21 September 2020; Published 9 October 2020

Academic Editor: Konglin Zhu

The various applications of the Internet of Things and the Internet of Vehicles impose high requirements on the network environment, such as bandwidth and delay. To meet low-latency requirements, the concept of mobile edge computing has been introduced. Through virtualisation technology, service providers can rent computation resources from the infrastructure of the network operator, whereas network operators can deploy all kinds of service functions (SFs) to the edge network to reduce network latency. However, how to appropriately deploy SFs to the edge of the network presents a problem. Apart from improving the network efficiency of edge computing service deployment, how to effectively reduce the cost of service deployment is also important to achieve a performance-cost balance. In this paper, we present a novel SF deployment management platform that allows users to dynamically deploy edge computing service applications with the lowest network latency and service deployment costs in edge computing network environments. We describe the platform design and system implementation in detail. The core platform component is an SF deployment simulator that allows users to compare various SF deployment strategies. We also design and implement a genetic algorithm-based service deployment algorithm for edge computing (GSDAE) in network environments. This method can reduce the average network latency for a client who accesses a certain service for multiple tenants that rent computing resources and subsequently reduce the associated SF deployment costs. We evaluate the proposed platform by conducting extensive experiments, and experiment results show that our platform has a practical use for the management and deployment of edge computing applications given its low latency and deployment costs not only in pure edge computing environments but also in mixed edge and cloud computing scenarios.

1. Introduction

People have now entered the era of 5G, Internet of Things (IoT), and big data. As a typical application of IoT in the transportation field, the Internet of Vehicles (IoV) can integrate sensing technologies, 5G telecommunication networks, vehicle big data, artificial intelligence technologies, and other advanced technologies. Edge computing [1] is a key technology of 5G networks that support IoV applications, such as autonomous driving [2], vehicle safety [3], intelligent traffic management [4, 5], and in-vehicle infotainment [6, 7]. With edge computing, various application services can be deployed on mobile devices located close to vehicles, such as base stations and roadside units. Through local data processing and encrypting, real-time and highly reliable IoV services can be realised. The European Telecommunication Union has formulated standards for the establishment of a multiaccess edge computing (MEC) working group and accelerates the establishment of a MEC ecosystem. Through these standards, 5G emerging (e.g., augmented/virtual reality) and vehicle services (e.g., IoV) can be widely deployed at the edge of the network [8–10].

Processing IoT/IoV big data are inseparable from cloud computing. Despite its power, cloud computing also has its limitations [11, 12]. For instance, the timeliness of its computing power can lead to delayed data feedback. Such a problem has been mainly ascribed to mass data transmission. Specifically, a limited network bandwidth will block data transmission and subsequently extend the response time. Another limitation lies in the effectiveness of computing power. When all data are transmitted to the cloud data centre and when some of these data lack any use or value (due to lack of preprocessing), these data will lead to a waste of cloud computing power. Edge computing has been proposed to improve the timeliness and effectiveness of data processing [13]. Table 1 shows the differences between edge cloud computing and traditional cloud computing in terms of their architecture, computing resources, and network latency. The synergy between cloud computing and edge computing forms the future IoT architecture.

Edge computing is a complex ecosystem. By deploying services in different locations between the access and core networks, edge computing can reduce the intermediate levels of the network to varying degrees, simplify the network architecture, and meet low-latency business requirements [14]. When the data are only locally valuable, edge computing can process and store the original data, thereby avoiding network processing latency. When massive amounts of data need to be processed, edge computing reduces the amount of data sent to the cloud data centre via local filtering and preprocessing, which not only reduces networking costs but also reserves the limited network bandwidth for processing more important data packets.

The deployment of edge computing is application-oriented and is affected by various factors, such as delay, bandwidth, data security, and edge infrastructure. On the one hand, edge computing involves a balanced consideration of meeting business indicators and considers both investment benefits and operation and maintenance requirements. The deployment of edge computing should not blindly meet network performance indicators but also need to achieve a balance between network performance and usage costs [15]. Effectively reducing user network usage costs is also an important factor that needs to be considered in the deployment of edge computing. This issue has received much attention in light of the recent industry discussions on how to widely apply edge computing.

On the other hand, service function (SF) deployment or service placement in cloud computing is a mature technology that is realised through different applications, such as OpenStack [16] and Kubernetes (K8s) [17]. Deploying services in the MEC platform allows low-latency services to meet the network needs of service providers and compensate for the shortcomings of cloud computing. However, in edge computing environments, SF deployment is still in a standardisation process. Given the decentralised architecture of these environments, the geographical location of users should be considered, which would increase the complexity of function deployment. In edge computing, many microdata centres (MDCs) are deployed in the edge network to provide localised services with limited computing resources, thereby reducing network latency in the cloud. However, given its limited computing resources, this architecture may be unable to address the sudden demand for services when such demand reaches its peak [18, 19]. In this case, the data volume should be transferred to nearby MDCs or remote cloud computing should be utilised for further processing. Each data volume transfer increases the network delay and affects the service quality for users [20, 21]. Deploying SFs also incurs costs, and choosing between efficiency and cost presents a problem. Service providers must also decide which MEC platforms they should deploy and how many services should be deployed on these platforms. Therefore, service providers should examine how services can be reasonably and effectively deployed in the edge network [22, 23].

Previous studies [23, 24] have mostly focused on improving network performance and resource utilisation when deploying services in edge computing. However, only a few of these studies have examined how to improve service deployment efficiency and reduce service placement costs to achieve a performance-cost balance. Considering both the network performance and usage costs of service placement, this paper proposes a new genetic algorithm-based service deployment algorithm for edge computing (GSDAE) and designs a novel edge computing service deployment management platform that comprises an SF deployment management system. The core of this platform uses the highly efficient GSDAE algorithm for SF deployment, reduces the average network latency for a client who accesses a service for multiple tenants that rent computing resources, and consequently lowers the cost of SF deployment. GSDAE can also be used not only in pure edge computing environments but also in mixed edge and cloud computing scenarios.

The major contributions of this article are as follows:

(i) We propose GSDAE to efficiently achieve a balance between service applications performance and networking costs when offloading services between cloud computing and the edge computing network. GSDAE considers both user experience in service offloading and network operational costs in developing a dynamic and flexible service deployment management scheme.

(ii) We develop a genetic algorithm-based service deployment management testbed platform to achieve a dynamic service deployment in an edge computing network. The design issues involved in integrating various service placement management components to implement the system are also explained in detail.

(iii) To verify the effectiveness and performance of the proposed platform and algorithm, we perform extensive experiments in seven scenarios. Experimental results show that our proposed approach improves the network applications offloading efficiency in terms of cost and end-to-end latency.

The rest of this paper is organised as follows. Section 2 discusses the related work. Section 3 presents the system model, defines the research problem, and proposes GSDAE. Section 4 presents the system architecture, the service deployment management platform, and the system implementation in detail. Section 5 evaluates the proposed platform and algorithm through extensive experiments. Section 6 concludes the paper and proposes some suggestions for future work.

Table 1: Comparison between cloud and edge computing.

Attributes	Cloud computing	Edge computing
Latency	Relatively high (50 ms ~ 55 ms)	Relatively low (1 ms~6 ms)
Security control	Nonlocally controllable	Locally controllable
Access type	Wide area network (WAN)	Local area network (LAN)
Servers locations	Mainly in the data centres	Mainly in the microdata centre
Architecture	Centralized	Decentralized
Suitable applications	High computing capacity	Lower latency
Services management	Cloud service providers	Edge applications managers
Computing resources	Relatively high (>100 origin servers)	Relatively low (1~10 edge servers)

2. Related Works

With the advent of the 5G era and the optimisation management of resources related to edge computing, the application of efficient service deployment methods has received much attention. Many studies have then begun to examine the service deployment of edge computing, including its network resource efficiency and latency. For instance, Xu et al. [24] proposed a novel uncoupled model for edge computing that separates service providers from edge computing infrastructure providers. In the coupled state, service providers only offer services within their respective infrastructures. Under this premise, the sharing of resources is restricted, and the overall performance, including service response time, resource usage, and success rate, is reduced. A weighted Voronoi diagram is then designed to make decisions related to the deployment of SFs. In this diagram, a data processing centre with a higher workload has a smaller scope of responsibility. Whilst showing some value, the proposed resource allocation method ignores the upper limit on the computing processing capacity of the MDC in the edge computing network environment. When the deployed service exceeds the load of the MDC, this service is forced to be offloaded to other MDCs, thereby increasing latency. Fan and Ansari [25] proposed a methodology for deploying a cloudlet in a wireless metro network. Specifically, they designed two deployment strategies, namely, the heaviest access point (AP) first placement (HAF) and the density-based clustering placement (DBC), to reduce the overall service deploying network delay. In HAF, the MDC is deployed next to the AP, and the greedy algorithm (GRE) collects the request rates of all APs before conducting power-down sorting according to the level of the request rate. Meanwhile, DBC is a high-density deployment approach that assumes that when *k* SFs need to be deployed, the deployment decision takes *k* rounds. An SF is deployed at each round, and the following steps are repeated each time: (1) select the cloudlet with the highest density to deploy an SF, (2) remove the user served by this cloudlet, and (3) repeat this procedure until the deployment is completed. Both HBF and DBC offer impressive contributions in reducing service deployment costs and delays. However, whether these methods can determine which MDC should be used to deploy services and how many services should be deployed based on user demand remains unclear. When a service is deployed to an MDC with a high load, this service will be forced to be reoffloaded to other MDCs, thereby increasing latency. Sun and Ansari [26] proposed an architecture that utilises cloudlets and software-defined network technologies to deploy computing resources at the edge of the network. They also designed a LEAD mechanism to reduce the network transmission delays and to allocate the application workload of mobile users (MUs) to a suitable cloudlet. In the GRE-based LEAD mechanism, the request rates of all MUs are initially sorted, and then, those MUs with high request rates are prioritised when processing the distributed workload to minimise the overall network latency. Whilst LEAD, which uses the ancient GRE to solve the new problem of edge computing service deployment, is commendable, this mechanism ignores the limited computing resources of MDCs, and how the service deployment methods are dynamically adjusted remains unknown. Mao et al. [27] found that areas with high computing resource requirements (e.g., commercial and financial areas) usually have a high number of requests, local rents, and deployment costs. Therefore, these factors should be considered when deploying SFs. Wang et al. [28] proposed ITEM, an iterative algorithm, to address the problem of deploying social virtual reality (VR) applications in edge computing. They constructed a graph to illustrate the costs related to VR application deployment and converted the cost optimisation problem into a graph-cut problem. They also evaluated the performance of ITEM by using real-world data and demonstrated that this algorithm has a fast convergence and outperforms the baseline approaches. Whilst the fundamentals of their theory are strong and solid, they do not discuss their system implementation in detail probably because of space limitations in their paper. Marjanovic et al. [29] designed an edge computing architecture to facilitate massive-scale mobile crowd deployments. Their proposed architecture introduces essential service overhead and service reconfiguration to reduce privacy threats and to allow users to control their contributed flow of sensor data. Whilst their idea of using the designed architecture as an edge computing network environment for trading sensing data and for aggregating or processing data is very forward looking, they neither explain how their designed system architecture is implemented nor evaluate its performance by using important evaluation metrics related to service deployment in edge computing, such as cost and latency. Wöbker et al. [30] proposed Fogernetes, a fog computing platform for managing and deploying fog applications with specific requirements. This platform comprises a labeling system to match the requirements of deployed service applications with edge computing device capabilities. They described their platform implementation in detail based on K8s and published their source codes and scripts on GitHub [31]. They added that their proposed platform can serve Fodeo (FogVideo), a surveillance application deployed for fog computing. Whilst these authors can open source their system implementation method, their article is only a short conference paper and therefore does not provide systems performance evaluations to prove the practicality of their approach. Martín–Pérez et al. [23] designed a hard-core repulsion Poisson model to help network operators flexibly deploy a MEC infrastructure and provide their users with low-latency multimedia services. Through this model, operators can determine how many MEC points of presence (PoPs) and base stations are needed and identify the locations of those MEC PoPs to which the base stations are assigned. They not only illustrated their rigorous modeling derivation process but also described how their designed model can be applied in real-life scenarios. Nevertheless, their simulation only focuses on a reasonable planning of the spatial location of MEC PoPs to improve signal reception strength whilst indirectly reducing the latency of multimedia services. Poularakis et al. [32] argued that many studies have recently addressed the problem of executing service tasks and routing service requests to corresponding edge servers by improving the utilisation efficiency of edge computing resources. However, their solutions may introduce asymmetric bandwidth requirements for many emerging services, such as augmented reality, network gaming, and autonomous driving. To address these problems, they applied joint service placement and requests routing and designed a randomised rounding algorithm to achieve a close-to-optimal performance. They also thoroughly analysed the complexity of three special cases and two general cases and then compared their algorithms with linear-relaxation and GREs to examine its storage, computation capacity, and bandwidth capacity performance. However, they did not evaluate the performance of their proposed algorithm in terms of its cost and latency of service deployment in edge computing networks. Chen et al. [33] proposed an edge cognitive computing (ECC) architecture that applies cognitive computing at the edge of the network. The ECC architecture employs a dynamic service migration mechanism to achieve an elastic, energy-efficient, and high-quality cognitive computing service deployment. They also designed a practical platform to evaluate their proposed mechanism, and experiment results show that such a mechanism demonstrates low latency and excellent user experience. Whilst integrating communication, computation, storage, and applications on the edge network into cognitive computing to achieve data and resource cognition is deemed valuable and promising, the above article does not provide any experimental results for service deployment costs at edge computing networks. Salaht et al. [34] comprehensively examined various service deployment problems in fog and edge computing networks and summarised the related works on resource management, fog and edge computing system architectures, and the main characteristics of each system. They also provided a taxonomy of service placement problems, discussed the most common strategies and approaches, major design objectives, and evaluation tools reported in the literature, and highlighted some open challenges and future research directions. Service deploying control plan design is one of the most important topics described in their service placement taxonomy. Two control plane models, namely, centralised and distributed coordination, are widely adopted to make decisions related to the deployment of IoT applications over fog and edge computing infrastructures. Moreover, two control manners, namely, offline and online placements, are designed to address the service deployment problems in fog and edge computing. The authors suggested that studying dynamic and online service deployment control methods is a valuable research area. Zhai et al. [35] argued that properly deploying services amongst resource-constrained edge servers has the potential in reducing latency for edge computing. They applied reinforcement learning, a deep learning technology, to predict user request patterns and the resource constraints of users, and then achieved an optimal service deployment for the 5G mobile edge computing infrastructure. They also depicted the system models as a Markov decision process and solved the problem by using the Dueling-Deep Q-network algorithm, which can learn service deployment from user requests. They compared this algorithm with other common service deployment algorithms to evaluate its performance in terms of response quantity on edge servers and total response time during the training and testing phases. Whilst using the artificial intelligence deep learning technology to solve the problem of edge computing network service deployment is novel, they did not disclose the performance of their designed algorithm deployed in a real production environment. Zhou et al. [36] argued that complex IoT applications may be implemented as a set of lightweight microservices and distributed amongst containers over a mobile edge computing network. They also proposed an approximation latency-aware microservice mashup approach (LMM) to achieve an optimal collocation of suitable microservices for an application. They depicted service access latency by using the M/M/1 queuing model and formulated an equation for calculating latency and network resource consumption. To evaluate its performance, they implemented LMM by using Java and conducted extensive experiments. Although the queue model is considered an ancient technology, the authors have used such technology in a valuable way to solve modern problems.

Most previous studies have designed new network architectures or algorithms to improve the network performance of edge computing service deployment. However, unlike these studies, this paper attempts to balance the efficiency and cost of SF deployment in an edge computing network. To the best of our knowledge, this paper is the first to examine the management efficiency and SF deployment cost for edge computing and to implement a practice management platform. An SF deployment management platform is designed and implemented to help users choose an optimal SF deployment strategy.

3. System Model

We describe a serviced deployment edge computing network N that comprises a set of connected MDCs and deployed SFs. We formulate N as $N = (S, V, E)$, where S denotes the set of deployed SFs, V denotes the set of MDCs in the edge network, and E denotes all direct links between two MDCs. Meanwhile, $|V|$ denotes the number of MDCs, whereas $|E|$ denotes the number of links. Each MDC has its own computation capacity, SFs have K types of services to be deployed, and each MDC is associated with the service deployment delay and cost for deployed service i.

3.1. Service Deployment Cost Model. When a service function is deployed from the cloud computing network to the edge computing network, we define the cost of service deployment C_{si} as follows as a summation of data transmission and computational costs:

$$C_{si} = \sum_{i\in I} p_i d_i + \sum_{i\in I}\sum_{j\in J} e_i t_{i,j}, \tag{1}$$

where p_i is the data transmission charge of deployed SF i. d_i is the data transmission capacity of SF i. e_i is the computational charge of the j^{th} type of deployed SF i. $t_{i,j}$ is the j^{th} type of deployed SF i, respectively.

3.2. Service Deployment Latency Model. When a service deployment request is issued, the service placement packets pass through the base stations and SDN-based cellular core networks and are sent to the edge computing layer. Therefore, service deployment latency involves the processing day of edge computing nodes, propagation, and transmission delay of service placement requests. Processing delay refers to the average amount of time that the edge server spends in processing user requests. Meanwhile, propagation and transmission delay refers to the average time that service deployment requests are propagated and transmitted via network links, respectively. If D_{si} denotes the service latency of deployed SF i, then D_{si} can be formulated as

$$D_{si} = \sum_{i\in I}\sum_{j\in J} A_{i,j} + \sum_{i\in I}\sum_{j\in J} X_{i,j} + \sum_{i\in I}\sum_{j\in J} Y_{i,j}, \tag{2}$$

where $A_{i,j}$ is the average processing time of edge computing node i that processes SF j and $X_{i,j}$ and $Y_{i,j}$ are the propagation and transmission delays of the network from cloud server i to edge server j, respectively.

3.3. Problem Definitions. An edge computing service platform should assign service deployment requests to the edge server on an MDC that can minimise the service latency and service placement costs. Although service operators can add edge computing servers to increase computing capacity and consequently reduce latency, doing so will also increase the service costs of operators. Moreover, edge servers are facing resource constraints given their limited hardware computing capacities. Therefore, we need to design an optimal edge computing service deployment method to optimise the trade-off between service placement cost and service latency. To meet this requirement, we propose GSDAE, which aims to minimise the service placement cost and latency in deploying SFs. GSDAE can be formulated as

$$\begin{aligned} P1\colon\ \min\Bigg(&\sum_{i\in I} p_i d_i + \sum_{i\in I}\sum_{j\in J} e_i t_{i,j} + \sum_{i\in I}\sum_{j\in J} A_{i,j} \\ &+ \sum_{i\in I}\sum_{j\in J} X_{i,j} + \sum_{i\in I}\sum_{j\in J} Y_{i,j}\Bigg). \end{aligned} \tag{3}$$

The MDCs in the edge computing network are located in different areas, which may be urban, suburban, or rural areas. When a service is deployed, the time for transmitting and processing the data and the cost of transferring data from the cloud data centres to the edge computing MDCs differs across each region. As shown in Figure 1, the edge computing network contains 10 MDCs that are distributed in the urban, suburban, and rural areas. When the data centre node A in the cloud needs to deploy a certain service to an MDC in the edge computing network, GSDAE performs calculations and analyses the cost and latency of deploying such service from node A to each MDC. GSDAE also calculates which MDC incurs the least service deployment delay and cost for deploying this service (e.g., deploying x to MDC M4, y to MDC M7, and z to MDC M8).

3.4. GSDAE. GSDAE uses the network topology as its input. As shown in Figure 2, the network topology comprises five MDCs and eight network connection links. The red letter indicates the ID of an MDC, whereas the number on the link indicates the corresponding network delay. The proposed SF management platform is assumed to provide infrastructure-as-a-service (IaaS) rental services to service providers that require low latency. We assume that the edge computing network platform has the available network performance monitoring tools for tracking network performance parameters in real time, such as the network latency amongst MDCs. The SFs have low latency and high bandwidth network requirements, and the MDC in the edge network is assumed to have limited computing resources and deployed services. Each MDC can also collect a certain number of user requests. Given their time-consuming nature, SFs should not be deployed frequently. The deployment result will be maintained for a certain period before being redeployed according to the next service demand.

Based on the minimum network delay amongst MDCs, GSDAE is processed continuously in the form of a genetic algorithm. The genetic algorithm-based GSDAE initially carries out the population initialisation step, which corresponds to a service deployment scenario that involves making decisions regarding which type of service and how many services should be deployed to the edge network. This step initialises a $\textbf{Population}_\textbf{c}$ matrix. The population of GSDAE is denoted by $\textbf{Population}_\textbf{c}$, which is a 2D array. Table 2 presents an example of the GSDAE population. Two types of SFs (SF1 and SF2) are deployed in this example.

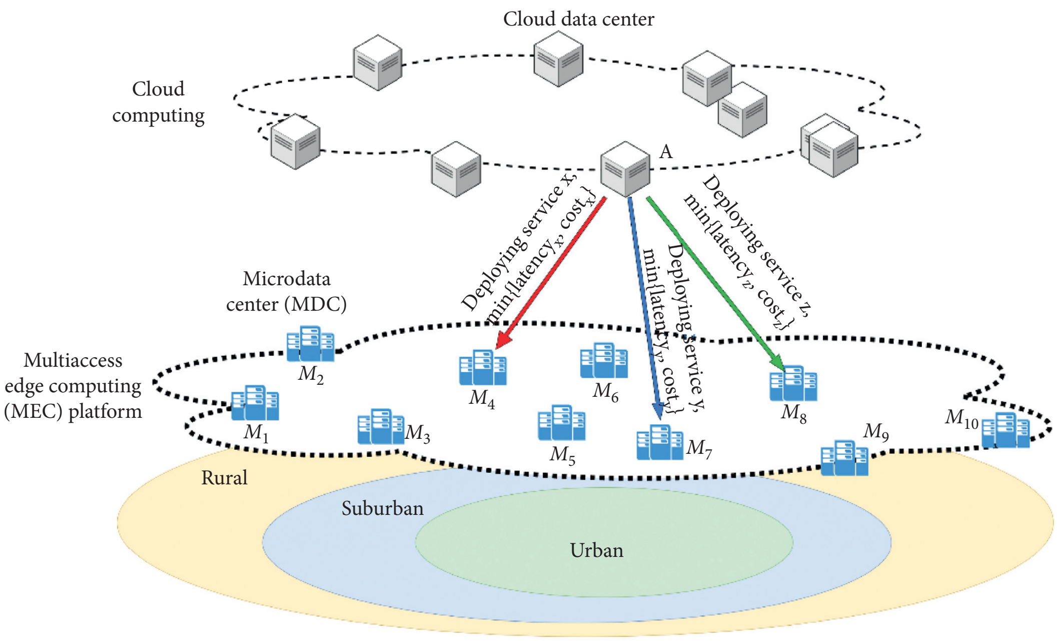
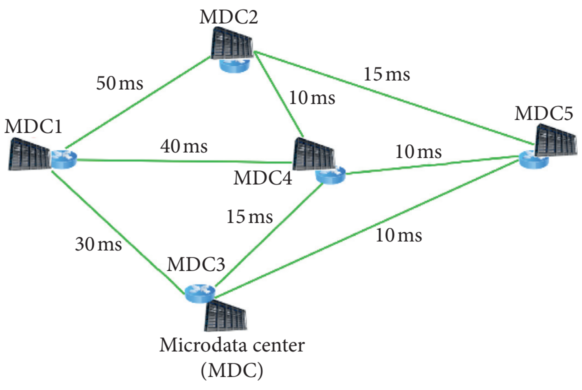

Cloud data center
Cloud computing
A
Deploying service x, min{latency_x, cost_x}
Deploying service y, min{latency_y, cost_y}
Deploying service z, min{latency_z, cost_z}
Microdata center (MDC)
Multiaccess edge computing (MEC) platform
M1
M2
M3
M4
M5
M6
M7
M8
M9
M10
Rural
Suburban
Urban

FIGURE 1: Problem of the service deployment strategy.



(a)

	MDC1	MDC2	MDC3	MDC4	MDC5
MDC1	0	50	30	40	40
MDC2	50	0	25	10	15
MDC3	30	25	0	15	10
MDC4	40	10	15	0	10
MDC5	40	15	10	10	0

(b)

FIGURE 2: Network topology in edge computing as the input of GSDAE: (a) network topology; (b) minimum network delay amongst MDCs.

$\mathbf{Population}_c$ is computed by using (4), where I is the number of MDCs, K is the number of functional service categories that need to be deployed, Gene_{ik} is the number of k-type functional services that need to be deployed on the i^{th} MDC, and V is the set of MDCs in the topology. i ranges from 1 to I, whereas k ranges from 1 to K:

$$\mathbf{Population}_c = \begin{bmatrix} \mathrm{Gene}_{ik} & \cdots & \mathrm{Gene}_{IK} \\ \vdots & \ddots & \vdots \\ \mathrm{Gene}_{ik} & \cdots & \mathrm{Gene}_{IK} \end{bmatrix}, \quad \cdot \begin{cases} I = |V|, \\ K = |\mathrm{SF}|, \end{cases} \tag{4}$$

$$\forall \mathrm{Gene}_{ik} \in \mathrm{Population}_c \begin{cases} \mathrm{Gene}_{ik} = 0, & vi \text{ has no } k^{\mathrm{th}} \text{ type SF}, \\ \mathrm{Gene}_{ik} > 0, & vi \text{ has } k^{\mathrm{th}} \text{ type deployed SF, and the amount is } \mathrm{Gene}_{ik}. \end{cases}$$

Table 2: GSDAE population example.

Location	MDC1	MDC2	MDC3	MDC4	MDC5
SF1	5	8	12	0	0
SF2	0	0	0	2	3

Given that the genetic algorithm randomly generates populations, these populations have a low survival rate. GSDAE undergoes seven special matrix transformations and calculations to increase such survival rate and to improve its efficiency and convergence speed. The seven matrix transformations are described as follows:

(1) GSDAE creates **SFMatrix**, which stores important information related to the deployed SFs, including the required computing resources, the maximum number of user requests that can be handled by an SF, and the number of required SFs. Table 3 shows a sample **SFMatrix** format where two SFs are deployed.

(2) GSDAE decomposes the **SFMatrix** and generates the $\mathbf{SF}_{\mathbf{capacity}}$ submatrix. $\mathbf{SF}_{\mathbf{capacity}}$ is transposed to store the computing capacity required for the deployed SF. Figure 3 presents a matrix transposition example for $\mathbf{SF}_{\mathbf{capacity}}$.

(3) To avoid generating an initial population that is unable to meet the resource requirements for the deployed SF, the population matrix needs to be normalised. As shown in Figure 4(a), although the tenant wants to deploy 20 type-1 services (SF1) to the edge network, the initial population matrix of GSDAE ($\mathbf{Population}_{\mathbf{c(limit\_region)}}$ in Figure 4) randomly generates 35 SF1 to be deployed to the MDCs of the edge network, hence not fulfilling the requirements to deploy type-1 services. In this case, the population matrix is normalised by using equation (5) to meet the SF deployment requirements. Figure 4(b) presents an example of population normalisation $\mathbf{Population}_{\mathbf{c(normalized)}}$. Through the matrix transformation indicated in equation (5), the survival rate of the progeny in GSDAE is improved:

$$\mathbf{Population}_{\mathbf{c(normalized)}} = \begin{bmatrix} \text{Gene(normalized)}_{ik} & \cdots & \text{Gene(normalized)}_{IK} \\ \vdots & \ddots & \vdots \\ \text{Gene(normalized)}_{ik} & \cdots & \text{Gene(normalized)}_{IK} \end{bmatrix},$$

$$\forall k \in K \begin{cases} \text{Gene(normalized)}_{ik} = \text{round}\left(\left(\dfrac{\text{Gene}_{ik}}{\sum_{i=1}^{I}\text{Gene}_{ik}}\right) \times \text{amount}_k\right), \\ \sum\limits_{i=1}^{I} \text{Gene(normalized)}_{ik} = \text{amount}_k. \end{cases} \tag{5}$$

(4) GSDAE uses the information from the MDC to generate **CapMatrix**, which records the computing resources owned by each MDC. Figure 5 shows an example of **CapMatrix**, where the first row stores the computing resources information for each MDC in a heterogeneous network.

(5) GSDAE continually uses $\mathbf{Population}_c$ and $(\mathbf{SF}_{\mathbf{capacity}})^{\mathrm{T}}$ to generate the $\mathbf{Consumption}_c$ matrix, which records the number of computing resources consumed by using the $\mathbf{Population}_c$ deployment strategy in each MDC. $\mathbf{Consumption}_c$ is calculated by using equation (6), and Figure 6 shows an example of $\mathbf{Consumption}_c$:

$$\mathbf{Consumption}_{\mathbf{c}} = \left(\mathbf{SF}_{\mathbf{capacity}}\right)^{\mathbf{T}} \times \mathbf{Population}_{\mathbf{c}}. \tag{6}$$

(6) Although the $\mathbf{Population}_{\mathbf{normalized}}$ matrix generated in step 3 meets the required number of SFs deployed for tenants, we have no guarantee that the MDC has sufficient computing resources for all SFs. Figure 7 presents an example of the required computing resources that exceed the limitations of the MDC. The deployed SF1 and SF2 require 40 units of computing resources, which exceed the limits of MDC 3. To address this problem, GSDAE verifies each SF to be deployed by using equation (7). Through this equation, GSDAE checks whether the MDC has sufficient computing resources for the deployment of SFs:

$$\forall m_{ij} \in \left(\mathbf{CapMatrix} - \mathbf{Consumption}_{\mathbf{c}}\right) \begin{cases} m_{ij} \geq 0, & \text{suitable deployment}, \\ m_{ij} < 0, & \text{unsuitable deployment}. \end{cases} \tag{7}$$

Table 3: **SFMatrix** format.

	Computing capacity required	Maximum requests	Numbers of service functions required
SF1	2	500	20
SF2	8	500	5

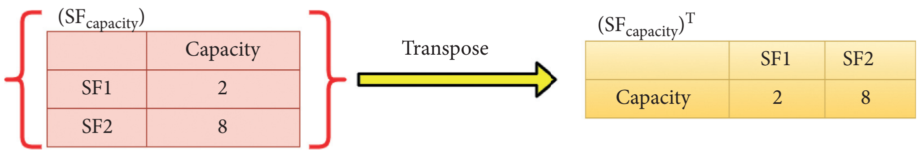


Figure 3: Matrix transposition of $\mathbf{SF_{capacity}}$.

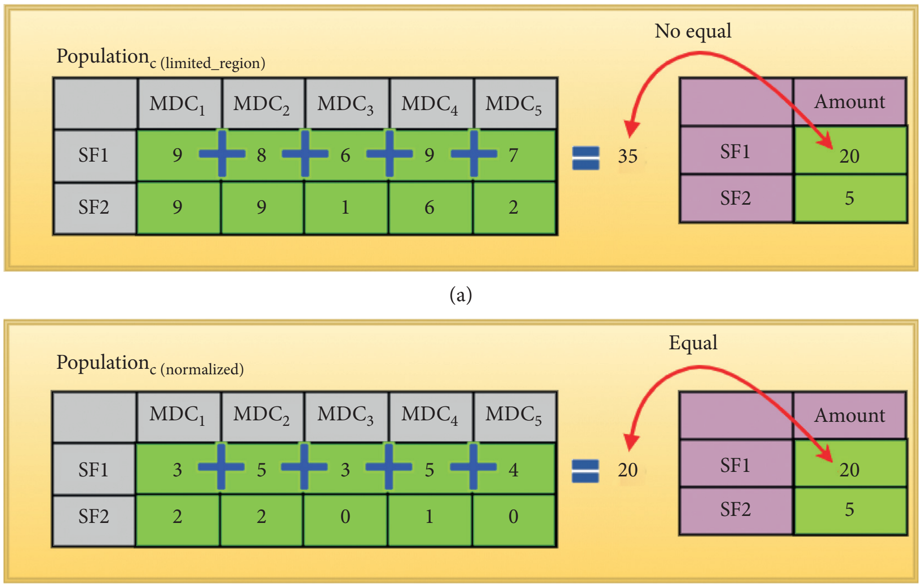


Figure 4: Matrix normalisation in the proposed GSDAE: (a) not fulfilling the SF deployment requirement; (b) population normalisation example.

MDC1	MDC2	MDC3	MDC4	MDC5
20	30	32	23	31

Figure 5: Example of **CapMatrix**.

(7) GSDAE uses GRE to address the overload in an MDC, which is redirected to a neighbouring MDC for processing. GSDAE uses network resource monitoring tools installed in the edge computing network platform to obtain the network connection delay and load information of each MDC in the edge network. GSDAE then greedily analyses all MDCs and decides which services are suitable for deployment. The fitness value of $\mathbf{Population_c}$ is calculated according to the average network delay score $\text{Delay}_{\text{score}}$ and overall deployment costs score $\text{Cost}_{\text{score}}$, which are calculated by using equations (8) and (9), respectively. DeploymentDelay_c represents the delay generated after deploying the service, $\text{Delay}_{\text{tenant}}$ represents the highest delay tolerated by the tenant when deploying a service, DeploymentCost_c represents the cost incurred when deploying a service, $\text{Total}_{\text{capacity}}$ represents the number of computing resources required to deploy a service, and $\text{Price}_{\text{max}}$ and $\text{Price}_{\text{min}}$ represent the maximum and minimum costs incurred by the IaaS resources rented by the services deployed by the tenant. The delay score ranges from 0 (longest) to 1 (shortest), whereas the cost score ranges from 0

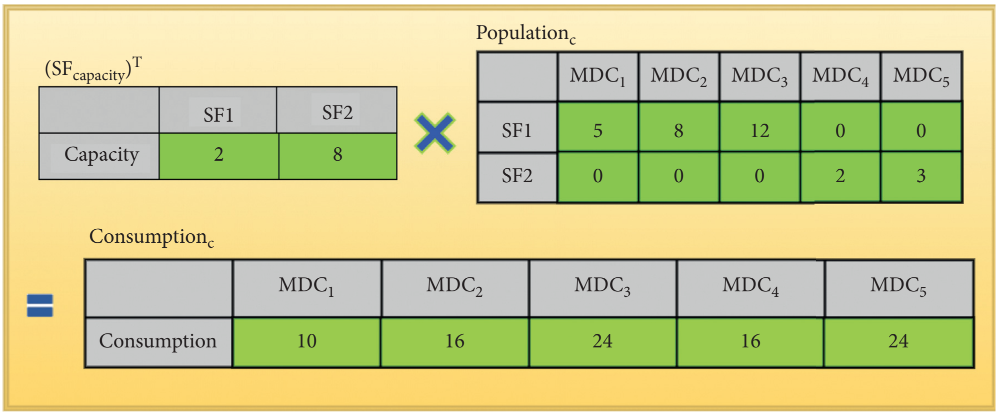


FIGURE 6: $\mathbf{Consumption}_c$ matrix generation.

(highest) to 1 (lowest). $\mathbf{c_1}$ and $\mathbf{c_2}$ in equation (10) represent the coefficients of these two scores. These parameters are adjustable and can reflect the degree to which lower network latency and deployment costs are valued when deploying SFs. $\mathbf{c_1}$ and $\mathbf{c_2}$ have a sum of 1, as shown in equation (10):

$$\text{Delay}_{\text{score}} = 1 - \frac{\text{DeploymentDelay}_c}{\text{Delay}_{\text{tenant}}}, \tag{8}$$

$$\text{Cost}_{\text{score}} = 1 - \frac{\text{DeploymentCost}_c}{\text{Total}_{\text{capacity}} \times ((\text{Price}_{\max} + \text{Price}_{\min})/2)}, \tag{9}$$

$$\text{Fitness}_{\text{value}} = c_1 \times \text{Delay}_{\text{score}} + c_2 \times \text{Cost}_{\text{score}}, \quad \text{where } c_1 + c_2 = 1. \tag{10}$$

GSDAE is summarised in Algorithm 1. We first initialise matrix $\mathbf{Population}_c$ (i.e., step 1) to store the information of SFs to be deployed and MDCs. The $\mathbf{Population}_c$ matrix contains the expected number of specific types of services to be deployed in each MDC, and K represents the total number of service types. When an iteration takes place, the algorithm generates **SFMatrix** to store the information of each type of deployed SF, including its required computing capacity, maximum user requests, and number of services required (Step 4). GSDAE then sequentially performs matrix transformation, decomposition, and submatrix generation and then determines whether a specific type of service is suitable for deployment in an MDC (steps 5 to 16). Afterwards, GSDAE calculates the fitness value of $\mathbf{Population}_c$ in each iteration (steps 17 to 21) and eventually determines which type of service is deployed to which MDC with the smallest deployment latency and cost.

After completing the above calculation, GSDAE uses three adjacency matrices to express the key attribute information of the topological structure of the edge computing network for service deployment. Take Figure 8 as an example, where (a) presents an edge computing network with 10 nodes, (b) is the adjacency matrix of network connectivity corresponding to the edge computing network topology (where 1 indicates the presence of a network connection between nodes and 0 indicates the absence of such connection), (c) is the adjacency matrix of service deployment delay corresponding to the edge computing network topology (where the number represents the minimum service deployment delay between nodes, and Inf represents the lack of any network connection between nodes; that is, the service deployment delay time is infinite), and (d) is the adjacency matrix of the service deployment cost corresponding to the edge computing network topology (where the number represents the lowest service deployment cost between nodes, and Inf indicates the lack of any network connection between nodes; that is, the service deployment cost is infinite). Take nodes 5 and 9 as example. Given the presence of a network connection between these nodes, the corresponding value of the network connection adjacency matrix (b) is 1. The value of the service deployment delay adjacency matrix (c) between nodes 5 and 9 is 0.8 s, which is the minimum delay time for deploying service x. The value of service deployment cost adjacency matrix (d) between nodes 5 and 9 is 73 units, which represents the lowest cost of deploying service x.

4. System Architecture and Implementation

This article designs a novel SF deployment management platform for edge computing. This platform comprises an SF deployment management system, which provides tenants with the most suitable SF deployment strategy whilst considering the number of user service requests and the limited edge computing resources of MDCs. We describe the architecture and implementation of this system in the following sections.

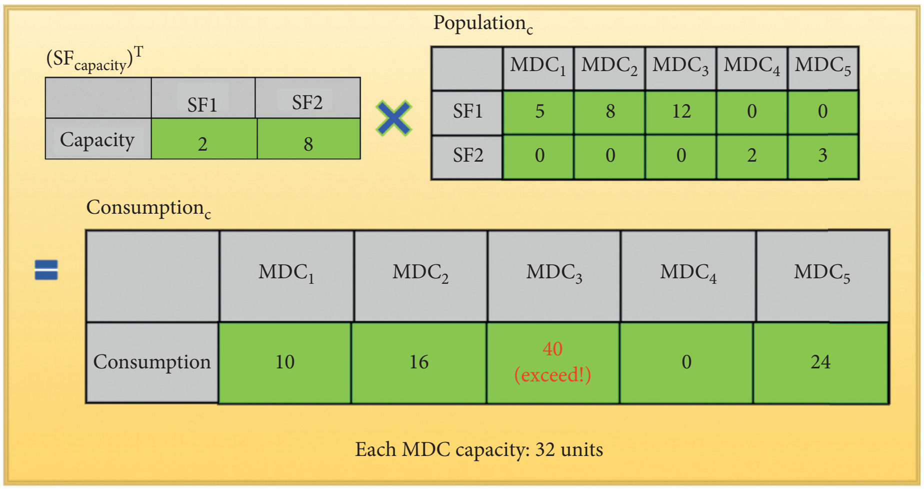
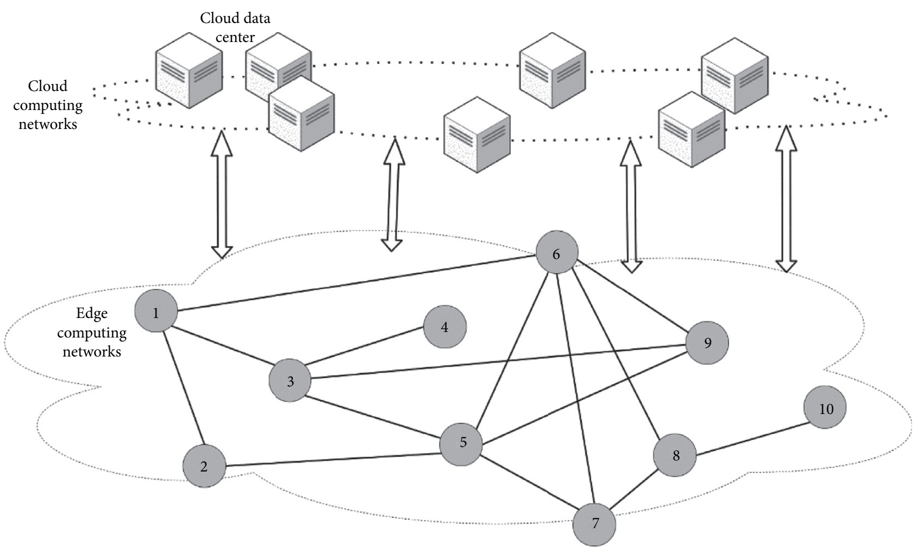


FIGURE 7: An example where the required computing resources exceed the limitations of an MDC.



(a)

Connectivity attribute

Nodes	1	2	3	4	5	6	7	8	9	10
1	0	1	1	0	0	1	0	0	0	0
2	1	0	0	0	1	0	0	0	0	0
3	1	0	0	1	1	0	0	0	1	0
4	0	0	1	0	0	0	0	0	0	0
5	0	1	1	0	0	1	1	0	1	0
6	1	0	0	0	1	0	1	1	1	0
7	0	0	0	0	1	1	0	1	0	0
8	0	0	0	0	0	1	1	0	0	1
9	0	0	1	0	1	1	0	0	0	0
10	0	0	0	0	0	0	0	1	0	0

(b)

Service deployment delay attribute

Nodes	1	2	3	4	5	6	7	8	9	10
1	Inf	0.05	0.03	Inf	Inf	0.7	Inf	Inf	Inf	Inf
2	0.05	Inf	Inf	Inf	1.4	Inf	Inf	Inf	Inf	Inf
3	0.03	Inf	Inf	0.07	0.9	Inf	Inf	Inf	1.2	Inf
4	Inf	Inf	0.07	Inf	Inf	Inf	Inf	Inf	Inf	Inf
5	Inf	1.4	0.9	Inf	Inf	2.1	0.03	Inf	0.8	Inf
6	0.7	Inf	Inf	Inf	2.1	Inf	1.1	0.7	0.04	Inf
7	Inf	Inf	Inf	Inf	0.03	1.1	Inf	0.05	Inf	Inf
8	Inf	Inf	Inf	Inf	Inf	0.7	0.05	Inf	Inf	0.08
9	Inf	Inf	1.2	Inf	0.8	0.04	Inf	Inf	Inf	Inf
10	Inf	Inf	Inf	Inf	Inf	Inf	Inf	0.08	Inf	Inf

(c)

Service deployment cost attribute

Nodes	1	2	3	4	5	6	7	8	9	10
1	Inf	112	65	Inf	Inf	43	Inf	Inf	Inf	Inf
2	112	Inf	Inf	Inf	29	Inf	Inf	Inf	Inf	Inf
3	65	Inf	Inf	42	27	Inf	Inf	Inf	126	Inf
4	Inf	Inf	42	Inf	Inf	Inf	Inf	Inf	Inf	Inf
5	Inf	29	27	Inf	Inf	244	301	Inf	73	Inf
6	43	Inf	Inf	Inf	244	Inf	439	256	72	Inf
7	Inf	Inf	Inf	Inf	301	439	Inf	99	Inf	Inf
8	Inf	Inf	Inf	Inf	Inf	256	99	Inf	Inf	118
9	Inf	Inf	126	Inf	73	72	Inf	Inf	Inf	Inf
10	Inf	Inf	Inf	Inf	Inf	Inf	Inf	118	Inf	Inf

(d)

FIGURE 8: Adjacency matrices of the edge computing network produced by GSDAE: (a) network topology of the edge computing network; (b) network connectivity adjacency matrix; (c) service deployment delay adjacency matrix; (d) service deployment cost adjacency matrix.

4.1. System Architecture. Our system comprises a proxy server, computing server, and edge orchestrator, which are considered the main components in service deployment management. This system, whose architecture is illustrated in Figure 9, can simultaneously reduce the network transmission delay resulting from service deployment and the costs of such deployment. The network infrastructure of the proposed SF deployment management system is based on OpenFlow vSwitch (OVS) [37], which is used as a data plane to control the data flow. Various off-shelf network and application monitoring toolsets are also broadly used in cloud computing and services provider platforms, including

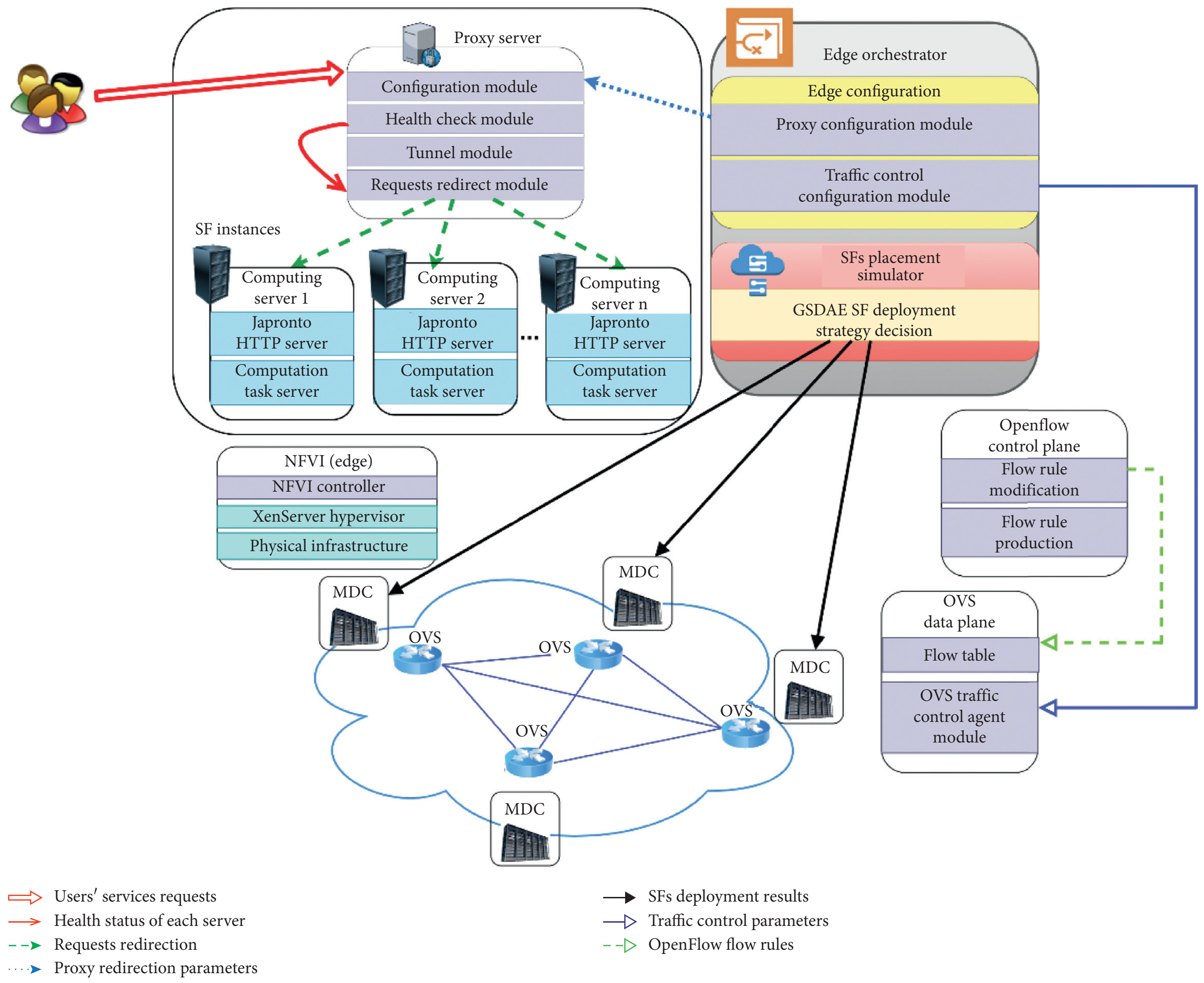


Figure 9: Architecture of the proposed SF deployment management platform.

software, tools, and utilities that continuously track and measure the computing resources for different services and monitor the system performance. In our proposed service deployment management platform for edge computing, we install some open source tools for Linux to obtain various metrics. The major toolsets are mtr, top, atop, htop, and glances, which allow our platform to collect important data, including the computing resources required by various services, average, minimum, and maximum network latency, memory, and CPU usage and edge computing server loading information (e.g., local volume and I/O activities).

4.2. System Implementation. To verify the effectiveness of our proposed edge computing SF deployment management system, the designed SF management platform was implemented on XEN servers. Tables 4 and 5 show the hardware and software specifications of the proposed genetic SF deployment management platform.

Due to limited experimental resources, we implemented our designed platform on two XEN servers, whose hardware specifications are shown in Table 4. Specifically, our platform uses two DELL R415 servers with an AMD Opteron Processor 4386, 32 GB memory, and 2 TB HDD storage space. We used the hypervisor solution of Citrix XEN Server [38] to build a virtual machine (VM) platform and to provide various services to the upper layer. Each hypervisor has its virtual network provided to the VM via the upper layer as a network connection, and all VMs are bridged to the physical network through the L2 bridge and are uniformly allocated virtual (private) and public IP addresses by the internal dynamic host configuration protocol/network address translation server to allow public IP forwarding to internal VMs and to access Internet services.

We implement OVS as the tenant IaaS server in both hypervisors and VMs to build the network topology environment in edge computing, as shown in Figure 10. Due to the limited hypervisor memory and CPU resources, our platform system environment consists of 5 OVS, 5 proxy servers, and 13 VMs, representing the IaaS servers of tenants. We also implemented the computation task server module on the tenant IaaS server to simulate the edge computing environment and to provide SFs that consume system

```
     Input:
(1)  Information of edge computing network topology
(2)  Information of all MDCs.
(3)  Information of all application SFs.
     Number of user requests. (2) $K$ represents the number of functional service categories that need to be deployed. (3) $Gene_{ik}$ denotes
     the number of $k$-type functional services that need to be deployed on the $i^{th}$ MDC. (4) The computing capacity required for SF $i$.
     (5) The number of required SFs.
     Output: the SF deployment of assignment matrices for all services
(1)  The population of GSDAE:
     $\mathbf{Population_c} = \begin{bmatrix} Gene_{ik} & \cdots & Gene_{IK} \\ \vdots & \ddots & \vdots \\ Gene_{ik} & \cdots & Gene_{IK} \end{bmatrix}$, and $\begin{cases} I = |V| \\ K = |SF| \end{cases}$
     $\forall Gene_{ik} \in Population_c \begin{cases} Gene_{ik} = 0, \ vi \text{ has no } k^{th} \text{ type SF} \\ Gene_{ik} > 0, \ vi \text{ has } k^{th} \text{ type deployed SF, and the amount is } Gene_{ik} \end{cases}$
(2)  $k = 1$;
(3)  While $k < K + 1$ do:
(4)      Create SFMatrix to store information related to the deployed SFs;
(5)      Decompose SFMatrix and generate the $\mathbf{SF_{capacity}}$ submatrix;
(6)      Transpose the $\mathbf{SF_{capacity}}$ matrix to generate $\mathbf{SF_{capacity}}^{\mathbf{T}}$, which stores the computing capacity required for the deployed SF;
(7)      Normalise the $\mathbf{Population_c}$ matrix:
     $\mathbf{Population_{c\,(normalized)}} = \begin{bmatrix} Gene\,(normalized)_{ik} & \cdots & Gene\,(normalized)_{IK} \\ \vdots & \ddots & \vdots \\ Gene\,(normalized)_{ik} & \cdots & Gene\,(normalized)_{IK} \end{bmatrix}$
     $\forall k \in K \begin{cases} Gene\,(normalized)_{ik} = round\left(\left(Gene_{ik} / \sum_{i=1}^{I} Gene_{ik}\right) \times Amount_k\right), \\ \sum_{i=1}^{I} Gene\,(normalized)_{ik} = Amount_k. \end{cases}$
(8)  Generate CapMatrix to record the computing resources owned by each MDC;
(9)  Calculate $\mathbf{Consumption}_c$ matrix to record the number of computing resources consumed by using the $\mathbf{Population}_c$ deployment
     strategy in each MDC;
         $\mathbf{Consumption_c} = (\mathbf{SF_{capacity}})^{\mathbf{T}} \times \mathbf{Population_c}$
(10) Check whether the deployment of SFs complies with the computing resources of the MDC;
         $M_{ij} = \mathbf{CapMatrix} - \mathbf{Consumption}_c$
(11) if $M_{ij} >= 0$ then
(12)     $Geni_{ik}$ is suitable to be deployed on $MDC_i$;
(13) End
(14) else
(15)     $Geni_{ik}$ is unsuitable to be deployed on $MDC_i$;
(16) End
(17) for each $Geni_{ik}$ of MDC in the graph do
(18) Use a greedy algorithm to calculate the overload of each MDC. Those MDCs with the lowest latency in the neighbouring networks
     and are not yet overloaded are initially chosen;
(19)     $Delay_{score} = 1 - (DeploymentDelay_c / Delay_{tenant})$
(20)     $Cost_{score} = 1 - (DeploymentCost_c / Total_{capacity} \times ((Price_{max} + Price_{min})/2))$
(21)     $Fitness_{value} = c_1 \times Delay_{score} + c_2 \times Cost_{score}$, where $c_1 + c_2 = 1$
(22) Return $MDC_{i,\,k}$ with the minimal latency and cost that is suitable for the deployment of $SF_{i,k}$.
```

Algorithm 1: Genetic algorithm-based service deployment algorithm for edge computing (GSDAE).

Table 4: Hardware specifications of the implemented genetic SF deployment management platform for edge computing.

Hardware	Specification
Server manufacturer/model	DELL INC. PowerEdge R145
CPU	AMD Opteron(tm) processor 4386
Memory	DDR-3 1600 32G
Disk storage	WD20EARX 2.0TB
Server operating system	Citrix ®XenServer 7.1.0

computing resources. The computation task server simulates the access of users to system resources and then executes various services for these users. We simulated this computation task as a typical HTTP get request. The implementation of the computation task server was based on the Japronto HTTP server [39] developed by squeaky-pl. We used the Japronto server for our platform because of its multiprocess pipelines and better performance compared with other HTTP servers.

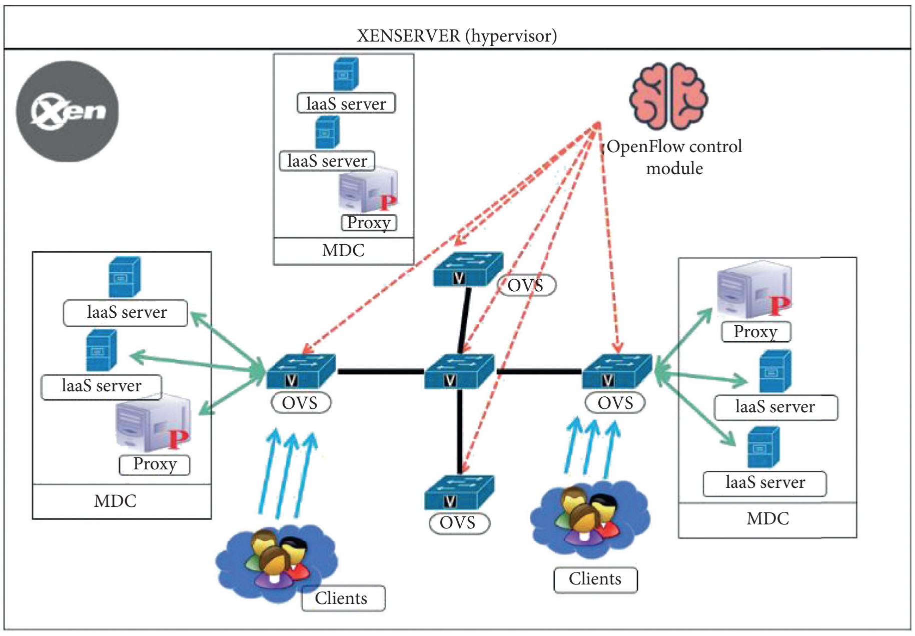


Figure 10: System architecture of the proposed genetic SF deployment management platform implemented on the XEN server for edge computing.

The OVS traffic control agent module was implemented on all OVSs of the platform system. This module directly communicates with the OVS kernel for traffic control to manage the network bandwidth and packet queuing rules. Meanwhile, the OpenFlow control module implemented in the OpenFlow control plane is responsible for the connectivity of the underlying network infrastructure. This module has two subfunctions, namely, the flow rules production and flow modification functions, both of which were implemented based on the Ryu SDN framework [40] developed and maintained by the Nippon Telegraph and Telephone Corporation in Japan. This framework is developed in the Python programming language to allow SDN network control. In the experiment, the OpenFlow rules generated by the flow rule production function were converted into the OpenFlow protocol, a packet-out command was issued to the underlying OVS, and this command was written by the OpenFlow control module to each corresponding OVS switch that supports OpenFlow in the network environment. In this way, the OpenFlow control module manages all networking packets, including ARP and IP packets.

Implemented in the proxy server, the tunnel module is mainly used as the processing interface of MDCs and is responsible for redirecting traffic flow. When the user service demand exceeds the load of an MDC, an overloading takes place. Accordingly, a tunnel was immediately established with GRE [41] or VXLAN [42] according to multimedia and nonmultimedia service applications in order to manage the SF offloading issued by the request redirect module. This tunnel can specify the network interface card and IP address used by an MDC for tunnel communication.

The health check module was also implemented in the proxy server to detect the service status in computation task servers every second. This server has UP and DOWN states, where the former indicates that the server is operating normally and the latter indicates that the server is not working. The request redirect module was also implemented in the proxy server for redirecting SF deployments. This module offloads the SFs according to the server load status detected by the health care module and the deployment strategy made by using GSDAE. When the MDC is about to overload, the request redirect module redirects those SFs that cannot be processed to other nearby MDCs.

Edge orchestrator was implemented to manage the core functions related to SF deployment. This orchestrator comprises a proxy configuration module, a traffic control configuration module, and an SF placement simulator that implements GSDAE. The SF placement simulator is the core component of the edge orchestrator and is developed by using C++. This simulator can adapt to different network topologies and makes deployment decisions related to the allocation of computing resources in different MDCs and the computing resource requirements of different SFs. When making a deployment decision, one must choose which SF should be deployed in which MDC and determine the number of SFs that need to be deployed in this MDC. Each MDC can then perform SF building and SF-oriented settings. The network topology of edge computing was converted into an undirected graph where the nodes represent the MDCs and the edges represent the network delay of physical lines amongst these MDCs. This undirected graph loads various information related to the number of computing resources of each MDC and the SF configuration that needs to be deployed. In the SFs placement simulator, various SF deployment strategies, such as GRE, DBC, and GSDAE, were implemented, and their performances across different scenarios were compared.

5. System Evaluation and Results

Seven experiments were conducted based on the implemented system management platform, and the performance of GSDAE was compared with that of two state-of-the-art SF deployment management algorithms in small- and large-scale edge computing networks with moderate and high loadings, respectively.

5.1. Experiment 1: SF Stress Test by Using Live Streaming Applications. By taking a real-time video streaming service as an example, experiment 1 aims to verify that each SF has its request processing limit. As shown in Figure 11, the main source of real-time audio and video streaming services is live streaming applications, such as digital TV, live videos, and webcasts. The audio and video contents are transmitted to a live stream broadcaster via network transmission. The network file system protocol, which allows different machines and operating systems to share individual files through a network, was adopted for the experiment. Through this protocol, the audio and video content in the video resource server was transmitted to the live stream broadcaster. Both the live stream broadcaster and video resource server have the same video and audio contents after the video transmission. After the audio and video content was received by the live stream broadcaster, such content was cut into several frames in time and sent to the live stream server via the real-time message protocol (RTMP) for users to watch. The users can then connect to the real-time streaming service website on the live stream server to access the real-time audio and video streaming content through any type of connecting network. Figure 12 shows that users can log in to the streaming website to watch streaming media content in real time during the experiment and that our service deployment management platform can simultaneously monitor the maximum number of service users in the background.

In this streaming service deployment stress test experiment, we used the Nginx RTMP module version 1.1.4 to set up a real-time streaming service web server. This module has three adjustable video playback bit rates corresponding to low (200 kbps–400 kbps), standard (401 kbps–1000 kbps), and high resolutions (1001 kbps–2200 kbps). To test the maximum number of users served by our service deployment management platform under the conditions of acceptable video playback quality, we set the video playback bit rate to 202 kbps (low resolution) and 937 kbps (standard resolution) when testing the deployment of 4K and 2K video streaming services, respectively. These parameters were set to test the maximum number of users of the platform without affecting the video playback quality.

The stress test software Flazr [43] was used to simulate a scenario where general users access the 3840×2160 (4K) high-quality and 1080P real-time audio and video streams on a single live stream server. As shown in Figure 13, this service deployment management platform can accommodate up to 1013 users watching the 1080P live stream at the same time. Given that a 4K image quality requires high encoding operations and transmission capacities, the maximum number of users watching the 4K live stream is 744, as shown in Figure 14.

5.2. Experiment 2: Response Time for SF Deployment. Experiment 2 aims to verify the practicality and feasibility of the proposed platform in terms of its deployment. The response time of GSDAE was compared with that of two other state-of-the-art service deployment methods, namely, GRE and DBC. Figure 15 presents the network topology, which comprises five MDCs. Underneath each MDC is an OVS-implemented router that is responsible for the transmission of network packets. The green line in Figure 15 represents the physical line in the topology, whereas the number on each line indicates the network delay set by the physical line. A large number of users access the service on each MDC and make service requests to the MDC through OVS.

In the experiment, we simulated a massive user access to the system by using the Sniper HTTP load generator [44] and by sending HTTP requests to MDCs through OVS. A total of 6,500 HTTP requests were received and randomly distributed on 5 MDCs, as shown in Figure 16. The performance of GSDAE was then compared with that of GRE and DBC in terms of the average response time of the deployed SF. GRE is a greedy algorithm where the service with the highest requests is chosen for deployment, whereas DBC performs multiround iterations according to the number of requests to be deployed. Given that 13 requests are to be deployed in the experiment, a total of 13 iterations were performed. Figure 17 presents the results of the three deployment strategies.

In the experiment, service requests were transmitted to the MDC through OVS based on the number of requests specified in Figure 16. A large number of service accesses were then sent to the system. The performance comparison results are shown in Figure 18, where the horizontal axis indicates the different deployment strategies, and the vertical axis indicates the average response time required for 6,500 requests. GSDAE outperforms the two other algorithms in terms of response time given that this algorithm considers not only the number of requests but also the network delay caused by the redirection of service. The average response time of GSDAE is about 12.77% shorter than that of GRE.

5.3. Experiment 3: SF Deploying Network Delay in a Small-Scale Network. Edge computing is a multitenant environment where many service providers want to deploy SFs on the edge network. In experiment 3, we simulated the multitenant situation and deployed various SFs in a small-scale edge computing environment. Three types of SFs were deployed, with each function requiring different computing resources. For instance, SF1 requires one unit of computing resources for a microservice, SF2 is an audio and video streaming service that requires two units of computing resources, and SF3 requires eight units of computing resources. The configuration of computing resources in this experiment was based on Amazon EC2 instance-type recommendations [45]. Amazon gives different instance-type recommendations for various services. For example, microservices use the T2 series of instances and require at

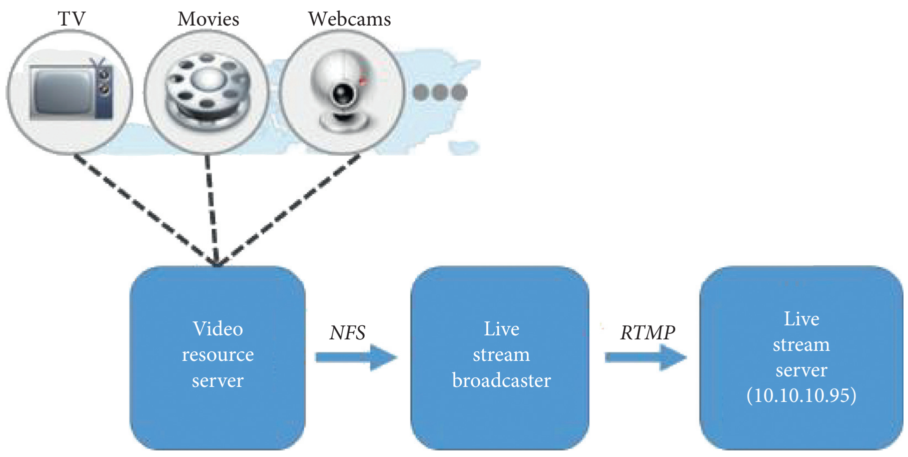


Figure 11: Real-time streaming SF deployment stress test workflow.

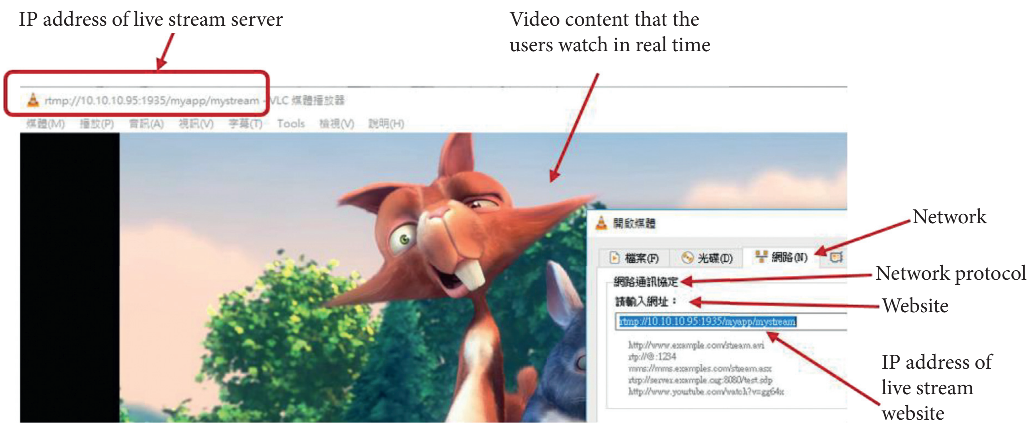
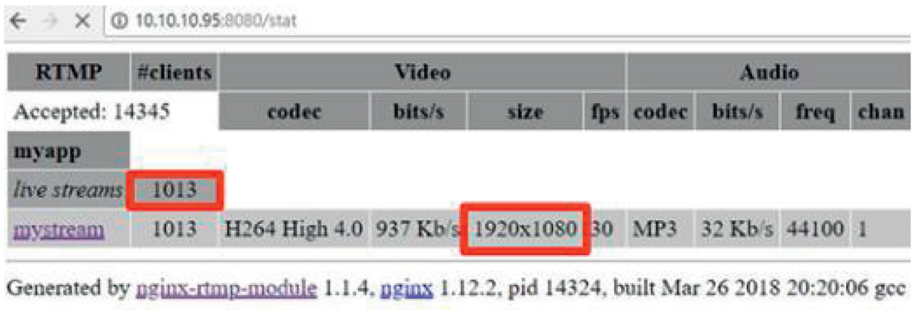

IP address of live stream server
Video content that the users watch in real time
rtmp://10.10.10.95:1935/myapp/mystream
VLC 媒體播放器
媒體(M)
播放(P)
音訊(A)
視訊(V)
字幕(T)
Tools
檢視(V)
說明(H)
開啟媒體
檔案(F)
光碟(D)
網路(N)
Network
網路通訊協定
Network protocol
請輸入網址：
Website
rtmp://10.10.10.95:1935/myapp/mystream
IP address of live stream website

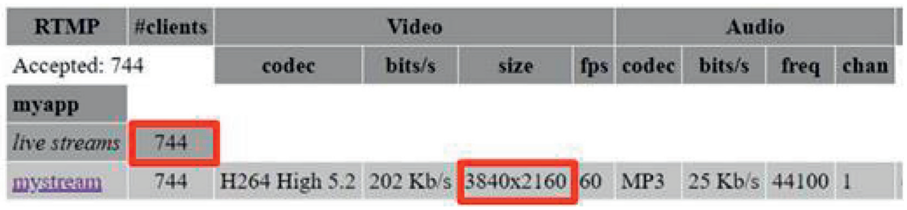
Figure 12: A user browsing the live streaming website and watching the video content in real time during the experiment.


10.10.10.95:8080/stat

RTMP	#clients	Video				Audio			
Accepted: 14345		codec	bits/s	size	fps	codec	bits/s	freq	chan
myapp									
live streams	1013								
mystream	1013	H264 High 4.0	937 Kb/s	1920x1080	30	MP3	32 Kb/s	44100	1

Generated by nginx-rtmp-module 1.1.4, nginx 1.12.2, pid 14324, built Mar 26 2018 20:20:06 gcc

Figure 13: Service limits of the 1920 × 1080 video service application deployment.



RTMP	#clients	Video				Audio			
Accepted: 744		codec	bits/s	size	fps	codec	bits/s	freq	chan
myapp									
live streams	744								
mystream	744	H264 High 5.2	202 Kb/s	3840x2160	60	MP3	25 Kb/s	44100	1

Figure 14: Service limits of the 3840 × 2160 (4K) video service application deployment.

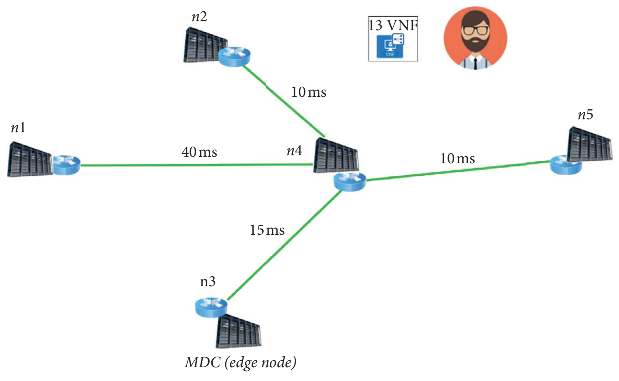


Figure 15: Network topology in experiment 2.

	*n*1	*n*2	*n*3	*n*4	*n*5
HTTP	1092	1358	3090	125	835

Figure 16: Number of service requests amongst various MDCs in experiment 2.

	MDC1	MDC2	MDC3	MDC4	MDC5
GSDAE	2	3	5	0	3
GRE	3	3	5	0	2
DBC	2	3	5	1	2

Figure 17: SFs deployed in experiment 2.

least 1 vCPU computing resource, video streaming and games use C5 series of instances and require at least 2 vCPU computing resources, and machine learning uses P3 series of instances and requires 8 vCPU computing resources.

The network topology in this experiment comprises 8 MDCs and 14 physical network connection lines that constitute a small-scale network topology. Each MDC in this topology has 32 units of computing resources, and the network delay of the connection line ranges from 10 ms to 90 ms. Table 6 shows the configurations of the small-scale edge computing network in this experiment.

Moderate and high loading scenarios were separately examined in the experiment. In equation (11), the load level is defined as the total computing resources required for the deployed SF. If the load level is 50% (i.e., at least 50% of the total computing resources are required), then moderate loading is observed. Meanwhile, if the load level is 75% (i.e., at least 75% of the total computing resources are required), then a high loading is observed. This SF deploying network delays for the three deployment strategies were compared under these two situations. Table 7 shows the moderate and high loadings of various types of SF deployment on a small-scale edge computing network:

$$\text{loading} = \frac{\text{total computing capacity of deployed SF}s}{\text{total computing capacity of MDC}s} \times 100\%. \tag{11}$$

Figure 19 presents the performance comparison results. The vertical axis shows the additional network delay required for SF deployment. Compared with the other two strategies, GSDAE can significantly reduce the network delays caused by the deployment of SFs. Under moderate network loading conditions, GSDAE experiences the shortest SF deploying network latency and outperforms the other deployment strategies. GSDAE can also reduce the network delay by approximately 38.20% compared with GRE. In the case of high network loading, GSDAE outperforms both GRE and DBC in terms of SF deploying network delay by approximately 43.99% and 22.01%, respectively.

5.4. Experiment 4: SF Deployment Cost in a Small-Scale Network. Urban areas, commercial areas, and financial districts often have a high number of requests and tend to have high local rent and deployment costs according to Porambage et al. [15]. Therefore, these factors should be taken into account when deploying SFs. In this experiment, the deployment cost per unit of computing resources was set between 20 and 50 units. A higher number of requests correspond to a higher deployment cost at the MDC, whereas a lower number of requests corresponds to a lower deployment cost.

Figure 20 presents the performance comparison results. The vertical axis shows the additional deployment cost required for deploying SFs. A lower value on this axis corresponds to a greater reduction in SF deployment costs. High and moderate network loadings were examined in this experiment. In a small-scale network environment, whether under high or moderate load, the deployment cost of GSDAE is equal to that of the other strategies and does not show any significant improvement.

5.5. Experiment 5: SF Deploying Network Delay in a Large-Scale Network. The performance of the aforementioned strategies in a large-scale edge computing network was compared in experiment 5. The large-scale network topology comprises 20 MDCs and 40 physical network connection lines. The large-scale edge computing network also has more nodes and more complex physical lines compared with the small-scale network. The SF deploying network delays of the three deployment strategies were compared under high and moderate network loading conditions. Table 8 shows the high and moderate loading configurations of various SF deployments in a large-scale network.

Figure 21 presents the performance comparison results. The vertical axis shows the additional network delay required for SF deployment. The lower the number of network delays, the better the performance of the corresponding service deployment management method, and the less the service deployment latency caused by this method. The network delay of large-scale networks is much longer than that of small-scale networks due to the topology of the former and the fact that the number of requests for SFs and deployed services are much higher in the former than in the latter regardless of the loading conditions. The network delay in the large-scale network is also much higher than that in the small-scale network. Even though the number of MDCs in a large-scale network environment has increased

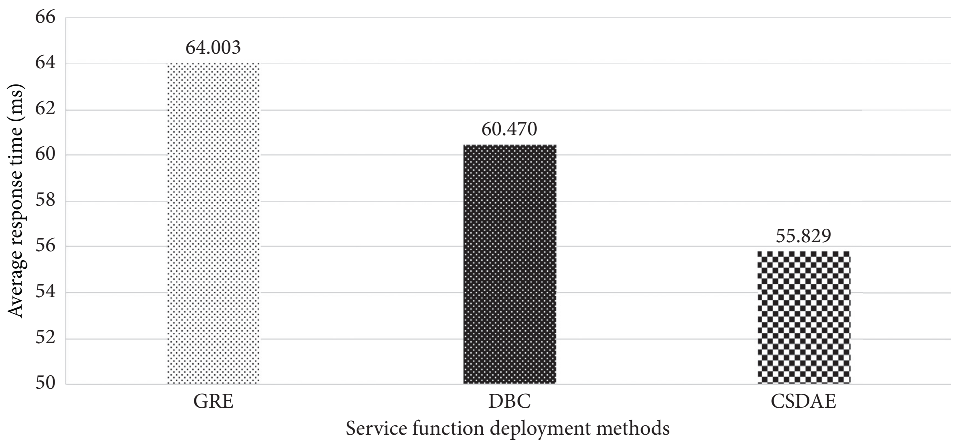


Figure 18: Comparison of the average response times of SF deployments.

Table 5: Software specifications of the implemented genetic SF deployment management platform for edge computing.

Software	Specification
HTTP server	Japronto
Operating system	Ubuntu Linux 16.04 64 bits
OpenFlow controller	Ryu 4.25
Open vSwitch	Support OpenFlow 1.4 switching capability
HTTP packet generator	Sniper
Audio and video converter	ffmpeg 2.8.3

Table 6: Configurations of the small-scale edge computing network in experiment 3.

Attributes	Values
Number of MDCs	8
Number of links	14
Link delay (ms)	[10, 90]
Computing capacity of each MDC	32 units
Maximum request handled by an SF	5
Deployment cost per computing unit	[20, 50] \$/unit

to 20, the number of three types of services deployed in a large-scale network has also increased by 450 (SF1: 200 > 500, SF2: 150 > 250, and SF3: 25 > 75). However, in a real network environment, various service deployment management algorithms combined with high-specification hardware and resources management can significantly reduce the delay time of service deployment.

Under a high-network loading condition, GSDAE achieves the lowest SF deploying delay amongst all compared strategies. The deploying latency caused by GSDAE is approximately 10.75% shorter than that caused by GRE. GSDAE also outperforms all the other strategies under moderate loading. Specifically, the SF deploying delays of GSDAE are 25.62% and 9.26% shorter than those of GRE and DBC, respectively.

In sum, in both small-and large-scale edge computing networks, GSDAE has a lower SF deploying network latency compared with the other two deployment strategies regardless of the loading conditions.

5.6. Experiment 6: SF Deployment Cost in a Large-Scale Network. In experiment 6, we compared the deployment costs of different SF deployment strategies in a large-scale network. The deployment cost per unit of computing resources in this experiment ranged from 20 to 50 units. A higher number of requests corresponds to a higher deployment cost of MDCs, and vice versa.

Figure 22 presents the experimental results. The vertical axis shows the additional cost for SF deployment. A lower additional deployment cost indicates a larger reduction in the deployment cost achieved by GSDAE. The experiment was conducted in high- and moderate-network loading conditions. Amongst the compared strategies, GSDAE incurs the lowest deployment costs for deploying SFs in both loading conditions.

5.7. Experiment 7: SF Deploying Delay and Cost in Mixed Cloud and Edge Computing. In experiment 7, we compared the SF deployment performance of GSDAE in a mixed edge and cloud computing environment. Table 9 presents the network environment configurations. The network nodes in this experiment were categorised into data centre (DC) and MDC. DC represents the cloud computing network infrastructure and, despite having a large number of computing resources, has a small number of nodes. Meanwhile, MDC represents the network infrastructure for edge computing and, despite having few computing resources, has a large number of nodes. Three types of SFs were deployed in this experiment, with each SF requiring different computing resources and several requests. Table 10 shows the configuration of the deployed SFs in the experiment.

The performance of GSDAE was compared with that of other deployment strategies in terms of network latency and deployment cost. Figure 23 presents the performance comparison results. The vertical axis of Figure 23(a) indicates the additional network delay required for SF deployment. In the mixed and edge computing networking environments, GSDAE outperforms the other two strategies and achieves the lowest network latency. Specifically, the SF deploying network delays of GSDAE are approximately 26.11% and 22.76% shorter than those of GRE and DBC,

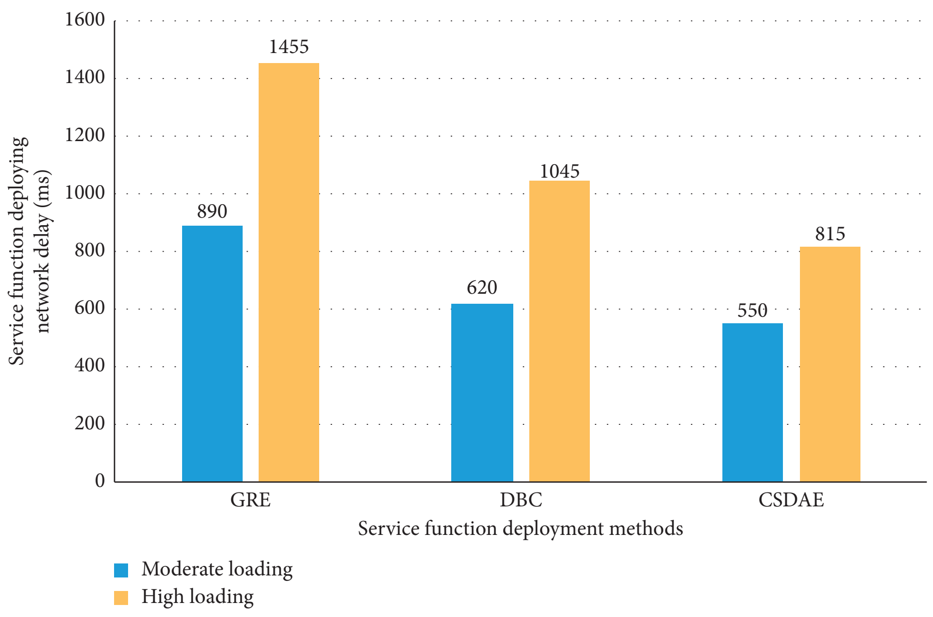


FIGURE 19: Comparison of SF deploying network delay in a small-scale edge computing network.

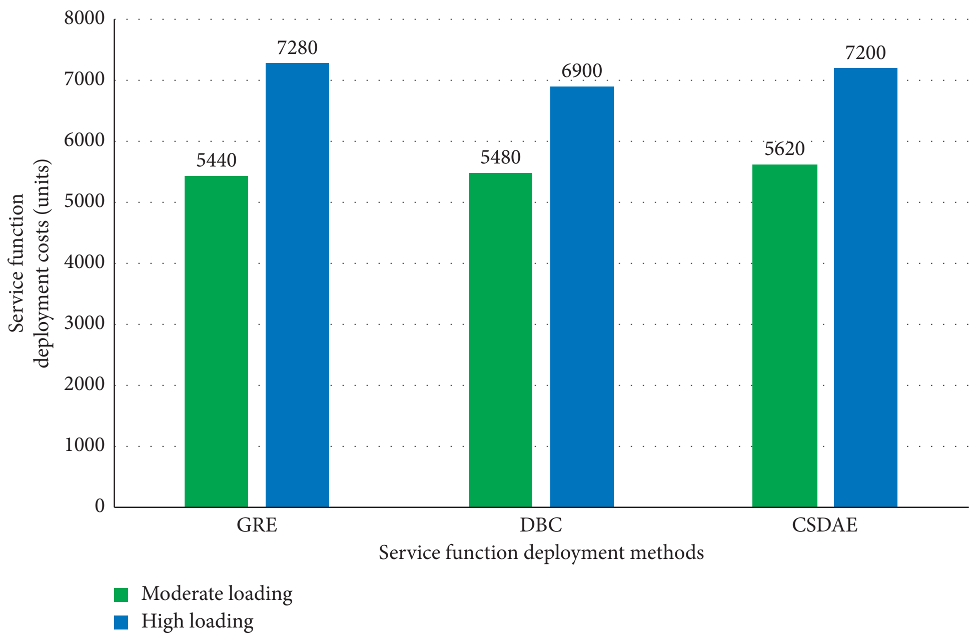


FIGURE 20: Comparison of SF deployment cost in a small-scale edge computing network.

TABLE 7: Moderate and high loadings of various types of SF deployments in a small-scale network.

	Request amounts	
	Moderate loading	High loading
SF1	200	250
SF2	150	150
SF3	25	50

respectively. Meanwhile, the vertical axis of Figure 23(b) shows the cost required for SF deployment. GSDAE also outperforms the other two deployment strategies in terms of deployment costs and saves approximately 10% greater costs compared with GRE.

The performance of the deployment strategies under the same number of requests for deployed SFs in a mixed cloud and edge computing network environment was then compared. Table 11 shows the configuration of the deployed SFs in the experiment, where each type of required SFs has the same number.

Figure 24 presents the performance comparison results. The vertical axis of Figure 24(a) shows the additional network delays required for SF deployment. GSDAE outperforms the other two strategies and can reduce the SF deploying network

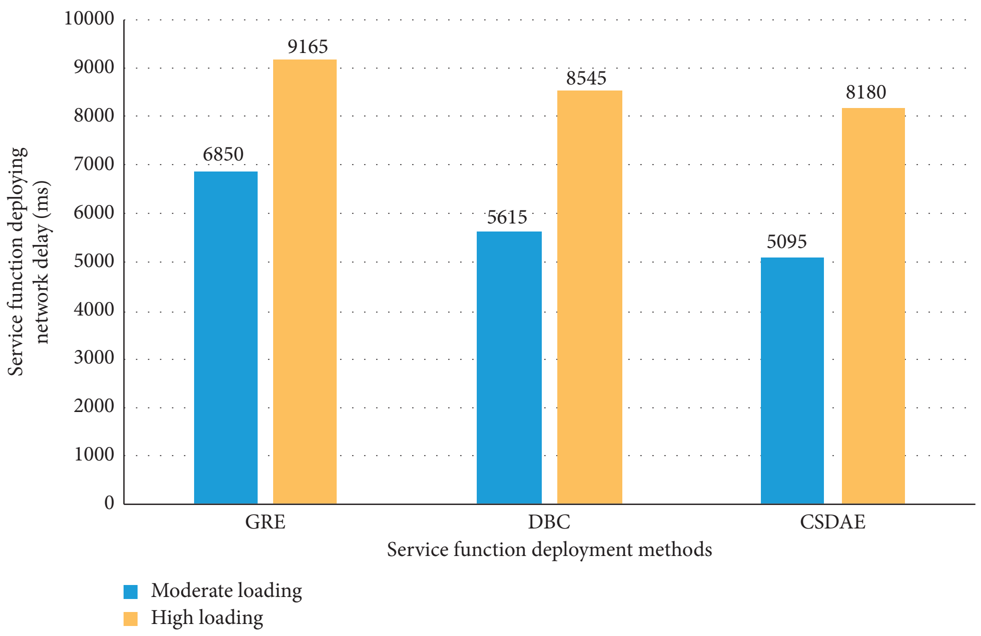


Figure 21: Comparison of SF deploying network delays in a large-scale edge computing network.

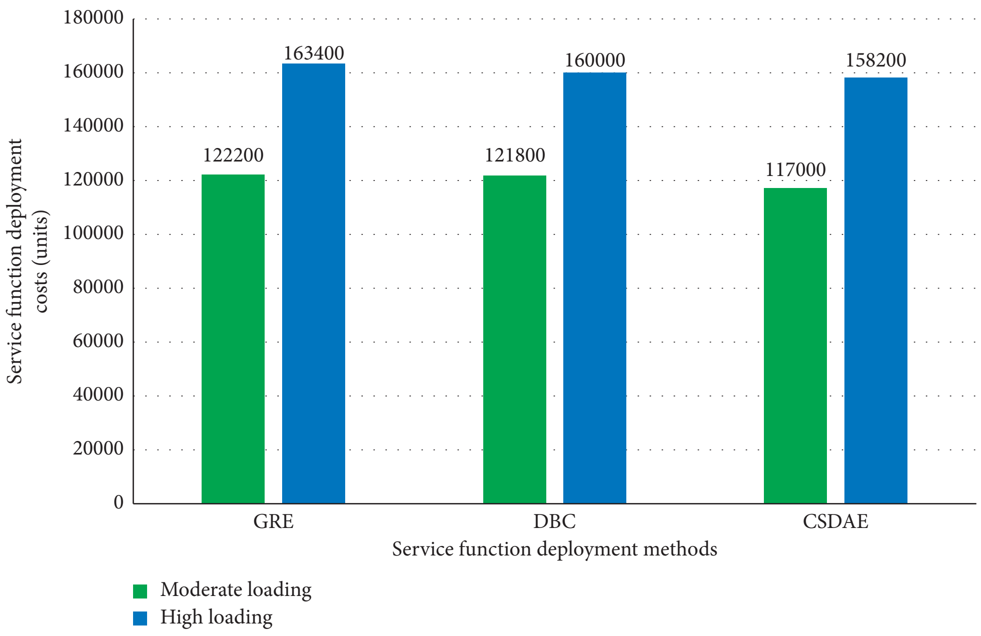


Figure 22: Comparison of SF deployment costs in a large-scale network.

Table 8: Moderate and high network loadings of various SF deployments in a large-scale network.

	Request amounts	
	Moderate loading	High loading
SF1	500	500
SF2	250	450
SF3	75	125

delay by approximately 48.97% and 17.90% compared with GRE and DBC, respectively. Meanwhile, the vertical axis of Figure 24(b) shows the additional cost required for SF deployment. GSDAE can save approximately 18.68% and 15.55% greater SF deployment costs compared with GRE and DBC, respectively. In sum, GSDAE demonstrates an outstanding performance in reducing SF deploying network delays and costs when each type of SF has the same number of requests.

Table 9: Configurations of mixed cloud and edge computing networks in experiment 7.

Attributes	Values
Number of data centres (DCs)	4
Number of MDCs	20
Number of links	40
Link delay (ms)	[10, 90]
Computing capacity of each DC	1000 units
Computing capacity of each MDC	32 units
Maximum request handled by an SF	5
Deployment cost per computing unit	[20, 50] $/unit

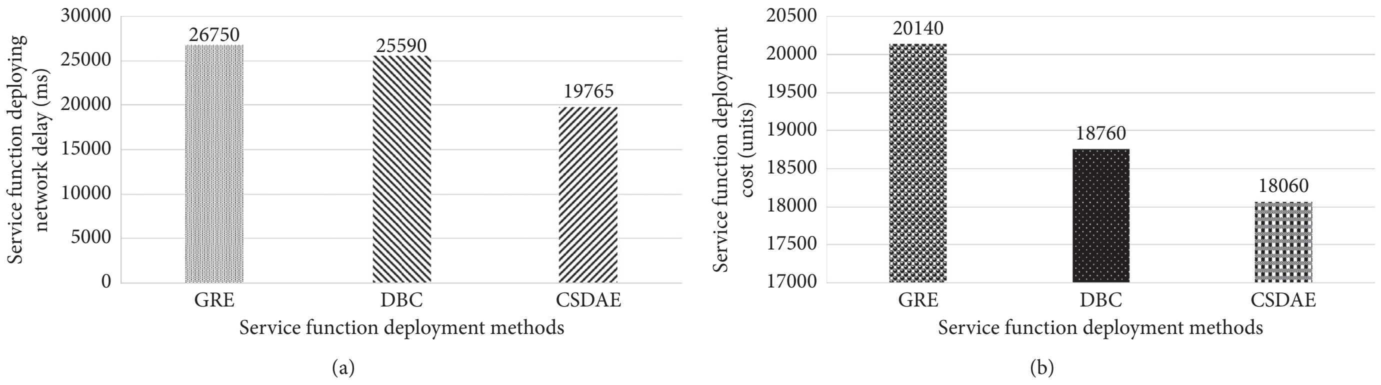


Figure 23: Comparison in a mixed cloud and edge computing environment under different numbers of requests for deployed SFs: (a) comparison of SF deploying network delay; (b) comparison of SF deploying costs.

Table 10: Configurations of the deployed SFs in experiment 7, with each computing capacity having a different number of required SFs.

	Computing capacity required	Number of service functions required
SF1	1	600
SF2	2	200
SF3	8	100

Table 11: Configurations of deployed SFs in experiment 7, with each SF having the same number of required SFs.

	Computing capacity required	Number of service functions required
SF1	1	200
SF2	2	200
SF3	8	200

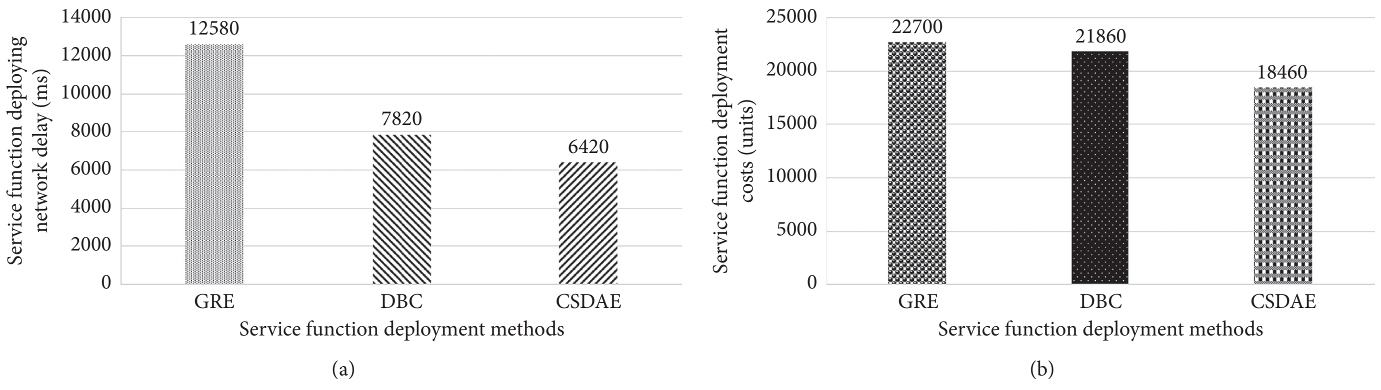


Figure 24: Comparison results in a mixed cloud and edge computing environment where each deployed SF has the same number of requests: (a) comparison of SF deploying network delays; (b) comparison of SF deploying costs.

6. Conclusion and Future Works

This paper designs and implements an SF deployment management platform based on a software-defined networking and network function virtualisation. This platform can simulate SF deployment under an edge computing network, and its kernel module can simulate the network delays amongst different OpenFlow switches and the service usage of users who access MDCs. The proposed platform can also establish various SF deployment scenarios in an edge computing environment.

This paper proposes a genetic algorithm-based GSDAE deployment strategy that expresses the SF deployment process via the evolution population of the genetic algorithm, implements a request-oriented SF, and continuously implements the evolutionary process of natural selection for the population of the genetic algorithm. GSDAE can improve the SF deployment efficiency along with an increasing number of evolution iterations. Extensive experiments are conducted to validate the performance of GSDAE, and the results show that whether in pure or mixed edge computing and cloud computing networks, GSDAE significantly reduces the network delay and service deployment costs and outperforms two other state-of-the-art SF deployment strategies. In sum, GSDAE can be applied in a real edge computing environment. The proposed genetic SF deployment management platform can also reduce the overall service deployment response time and enhance user experience. With the advent of the 5G era, new forms of network services will be designed and deployed in IoT, and service providers are bracing themselves for a complex and dynamic IoT era. Accordingly, these providers should design highly flexible and diversified service deployment charging methods. We hope that the edge network service deployment management strategy proposed in this work can offer these service providers with a reference tool for evaluating service deployment costs and guiding them in designing service deployment charging models for cloud and edge computing networks.

In a future study, the proposed SF deployment management platform will be implemented on other virtualisation architectures, such as OpenStack. Issues related to backup, high availability, dynamic automatic control mechanisms, and maintenance management will also be examined. Whilst the proposed platform provides services to tenants in consideration of the limited computing resources of MDCs, the bandwidth requirements of these MDCs are not taken into account. How to isolate and protect different SFs by using virtual network slicing techniques to provide customised 5G network services when providing service deployment presents another challenge.

Data Availability

Access to data is restricted due to commercial confidentiality.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the Ministry of Science and Technology of Taiwan, R.O.C., under Contract MOST 108-2221-E-008-033-MY3 and 105-2221-E-008-071-MY3.

References

[1] M. Satyanarayanan, “The emergence of edge computing,” *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[2] D. C. Li, L.-D. Chou, L.-M. Tseng, Y.-M. Chen, and K.-W. Kuo, “A bipolar traffic density awareness routing protocol for vehicular ad hoc networks,” *Mobile Information Systems*, vol. 2015, Article ID 401518, 12 pages, 2015.

[3] D. C. Li, L.-D. Chou, L.-M. Tseng, and Y.-M. Chen, “Energy efficient min delay-based geocast routing protocol for the internet of vehicles,” *Journal of Information Science and Engineering*, vol. 31, no. 6, pp. 1903–1918, 2015.

[4] K. Zhu, Z. Chen, Y. Peng, and L. Zhang, “Mobile edge assisted literal multi-dimensional anomaly detection of in-vehicle network using LSTM,” *IEEE Transactions on Vehicular Technology*, vol. 68, no. 5, pp. 4275–4284, 2019.

[5] D. Wang, X. Pei, L. Li, and D. Yao, “Risky driver recognition based on vehicle speed time series,” *IEEE Transactions on Human-Machine Systems*, vol. 48, no. 1, pp. 63–71, 2017.

[6] K. Zhu, Z. Yao, N. He, D. Li, and L. Zhang, “Toward full virtualization of the network topology,” *IEEE Systems Journal*, vol. 13, no. 2, pp. 1640–1649, 2018.

[7] Y. Guo, H. Xu, Y. Zhang, and D. Yao, “Integrated variable speed limits and lane-changing control for freeway lane-drop bottlenecks,” *IEEE Access*, vol. 8, pp. 54710–54721, 2020.

[8] H.-C. Hsieh, J.-L. Chen, and A. Benslimane, “5G virtualized multi-access edge computing platform for IoT applications,” *Journal of Network and Computer Applications*, vol. 115, pp. 94–102, 2018.

[9] L.-D. Chou, T. C. Li, D. C. Li, C.-M. Lin, and Y.-C. Lin, “Development of a lilliput multimedia system to enhance students’ learning motivation,” *Journal of Information Science and Engineering*, vol. 31, no. 4, pp. 1357–1372, 2015.

[10] L.-D. Chou, D. C. Li, W.-S. Chen, and Y.-J. Chang, “Design and implementation of a novel location-aware wearable mobile advertising system,” *Journal of Internet Technology*, vol. 18, no. 5, 2016.

[11] S. Singh and I. Chana, “A survey on resource scheduling in cloud computing: issues and challenges,” *Journal of Grid Computing*, vol. 14, no. 2, pp. 217–264, 2016.

[12] Y. Ai, M. Peng, and K. Zhang, “Edge computing technologies for Internet of Things: a primer,” *Digital Communications and Networks*, vol. 4, no. 2, pp. 77–86, 2018.

[13] S. Chen, L. Jiao, L. Wang, and F. Liu, “An online market mechanism for edge emergency demand response via cloudlet control,” in *Proceedings of the IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pp. 2566–2574, IEEE, Paris, France, May 2019.

[14] W. Shi and S. Dustdar, “The promise of edge computing,” *Computer*, vol. 49, no. 5, pp. 78–81, 2016.

[15] P. Porambage, J. Okwuibe, M. Liyanage, M. Ylianttila, and T. Taleb, “Survey on multi-access edge computing for internet of things realization,” *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 2961–2991, 2018.

[16] OpenStack, https://www.openstack.org/.

[17] Kubernetes (K8s), https://kubernetes.io/.

[18] G. Premsankar, M. Di Francesco, and T. Taleb, “Edge computing for the Internet of Things: a case study,” *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 1275–1284, 2018.

[19] F.-H. Tseng, Y.-M. Jheng, L.-D. Chou, H.-C. Chao, and V. C. M. Leung, “Link-aware virtual machine placement for cloud services based on service-oriented architecture,” *IEEE Transactions on Cloud Computing*, p. 1, 2017.

[20] S. Han, J. Li, Q. Dong, Y. Ma, and L. Song, “Service-aware based virtual network functions deployment scheme in edge computing,” in *Proceedings of the 2020 22nd International Conference on Advanced Communication Technology (ICACT)*, pp. 562–565, IEEE, PyeongChang, Republic of Korea, February 2020.

[21] A. Aral, I. Brandic, R. B. Uriarte, R. De Nicola, and V. Scoca, “Addressing application latency requirements through edge scheduling,” *Journal of Grid Computing*, vol. 17, no. 4, pp. 677–698, 2019.

[22] Z. Zhou, Q. Wu, and X. Chen, “Online orchestration of cross-edge service function chaining for cost-efficient edge computing,” *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 8, pp. 1866–1880, 2019.

[23] J. Martín-Pérez, L. Cominardi, C. J. Bernardos, A. De la Oliva, and A. Azcorra, “Modeling mobile edge computing deployments for low latency multimedia services,” *IEEE Transactions on Broadcasting*, vol. 65, no. 2, pp. 464–474, 2019.

[24] J. Xu, B. Palanisamy, H. Ludwig, and Q. Wang, “Zenith: utility-aware resource allocation for edge computing,” in *Proceedings of the 2017 IEEE International Conference on Edge Computing (EDGE)*, pp. 47–54, IEEE, Honolulu, HI, USA, June 2017.

[25] Q. Fan and N. Ansari, “Cost aware cloudlet placement for big data processing at the edge,” in *Proceedings of the 2017 IEEE International Conference on Communications (ICC)*, pp. 1–6, IEEE, Paris, France, May 2017.

[26] X. Sun and N. Ansari, “Latency aware workload offloading in the cloudlet network,” *IEEE Communications Letters*, vol. 21, no. 7, pp. 1481–1484, 2017.

[27] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, “A survey on mobile edge computing: the communication perspective,” *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[28] L. Wang, L. Jiao, T. He, J. Li, and M. Mühlhäuser, “Service entity placement for social virtual reality applications in edge computing,” in *Proceedings of the IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pp. 468–476, IEEE, Honolulu, HI, USA, April 2018.

[29] M. Marjanovic, A. Antonic, and I. P. Žarko, “Edge computing architecture for mobile crowdsensing,” *IEEE Access*, vol. 6, pp. 10662–10674, 2018.

[30] C. Wöbker, A. Seitz, H. Mueller, and B. Bruegge, “Fogernetes: deployment and management of fog computing applications,” in *Proceedings of the NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–7, IEEE, Taipei, Taiwan, April 2018.

[31] GitHub, https://github.com/.

[32] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, “Joint service placement and request routing in multi-cell mobile edge computing networks,” in *Proceedings of the IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pp. 10–18, IEEE, Paris, France, May 2019.

[33] M. Chen, W. Li, G. Fortino, Y. Hao, L. Hu, and I. Humar, “A dynamic service migration mechanism in edge cognitive computing,” *ACM Transactions on Internet Technology*, vol. 19, no. 2, pp. 1–15, 2019.

[34] F. A. Salaht, F. Desprez, and A. Lebre, “An overview of service placement problem in fog and edge computing,” *ACM Computing Surveys (CSUR)*, vol. 53, no. 3, 2020.

[35] Y. Zhai, T. Bao, L. Zhu, M. Shen, X. Du, and M. Guizani, “Toward reinforcement-learning-based service deployment of 5G mobile edge computing with request-aware scheduling,” *IEEE Wireless Communications*, vol. 27, no. 1, pp. 84–91, 2020.

[36] A. Zhou, S. Wang, S. Wan, and L. Qi, “LMM: latency-aware micro-service mashup in mobile edge computing environment,” *Neural Computing and Applications*, vol. 32, pp. 15411–15425, 2020.

[37] Open vSwitch, http://www.openvswitch.org/.

[38] XenServer, https://xenserver.org/.

[39] Japronto, https://github.com/squeaky-pl/japronto.

[40] Ryu SDN Framework, https://osrg.github.io/ryu/.

[41] Generic Routing Encapsulation (GRE), “RFC1701 Internet Engineering Task Force (IETF),” March 2000, https://tools.ietf.org/html/rfc1701.

[42] Virtual eXtensible Local Area Network (VXLAN): A framework for overlaying virtualized layer 2 networks over layer 3 networks, RFC7348 internet engineering task force (IETF), https://tools.ietf.org/html/rfc7348.

[43] Flazr, https://sourceforge.net/projects/flazr/.

[44] Sniper, https://github.com/btfak/sniper.

[45] Amazon EC2 instance types, https://www.amazonaws.cn/en/ec2/instance-types/.