Journal of Network and Computer Applications 215 (2023) 103630

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Cost optimization of omnidirectional offloading in two-tier cloud–edge federated systems

Binayak Kar [a,*], Ying-Dar Lin [b], Yuan-Cheng Lai [c]

[a] Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan
[b] Department of Computer Science, National Yang Ming Chiao Tung University, Hsinchu, Taiwan
[c] Department of Information Management, National Taiwan University of Science and Technology, Taipei, Taiwan

ARTICLE INFO

Keywords:
Cloud–edge systems
Cost
Latency
Federation
Offloading
Reverse offloading
Optimization

ABSTRACT

The use of the federation can exploit the advantages of cloud and edge computing technologies as the federation provides the facilities by which both can complement each other. However, certain services are needed to offload from clouds to edges, termed reverse offloading, and between edges, termed horizontal offloading. By considering the scenarios discussed above, in this paper, we propose a generic omnidirectional (OMNI) architecture of cloud–edge computing systems intending to provide vertical (edge to cloud offloading, and vice versa), and horizontal (between edges) offloading. To investigate the effectiveness of the proposed architecture in different operational scenarios, we formulate the cost optimization problem with different latency (loose, low, ultra-low) constraints. We develop an offloading algorithm using simulated annealing named the two-tier simulated annealing (TTSA) algorithm. We set two criteria for offloading: (1) offloading based on the job's size and (2) offloading based on the job's priority, irrespective of its size. The experimental results show that our proposed OMNI architecture can reduce the total cost by 7%–10%, compared to other existing architectures, and our proposed TTSA algorithm can reduce the cost by 45%–55%, compared to other existing algorithms. The average latency in OMNI architecture is relatively very less compared to other architectures.

1. Introduction

A federation is a group of service providers agreeing upon standards of operation collectively. For example, "federated cloud" facilitates the interconnection of two or more geographically separate clouds for load-balancing traffic and accommodating spikes in demand from respective users (Liaqat et al., 2017). The federation scenario is defined by various researchers differently. The collection of service providers cooperates to provide resources requested by users (Truong-Huu and Tham, 2014). For example, in cloud-A, computation is cheaper, but storage cost is high, and in cloud-B, storage cost is low but has high computation cost. In such a scenario, federation plays a key role where both the clouds agree to provide a common platform to facilitate low computation and storage costs to their customers. One service provider wholesales or rents computing resources to another service provider (Wang et al., 2017); in such a scenario by this federation, a provider acts as a user and service provider simultaneously (Goudarzi et al., 2021). For example, when the customer submits its request to cloud-A, but cloud-A does not have enough resources to serve the customer; hence it offloads (Satyanarayanan, 2015) the request to cloud-B that has the resources to serve the request. However, after the edges are re-architectures as data centers (Peterson et al., 2016), the cloud–edge federation comes into existence (Kelaidonis et al., 2016), where both cloud and edge complement each other in various scenarios while providing services to their customers.

The following reasons significantly highlight the necessity of the cloud–edge and edge–edge federation. Cloud computing provides greater data storage and computing power but causes high communication latency as clouds are far away from the end users. In contrast, edge computing offers similar services with a lower communication latency as they are closer to the end user but with limited capacity and coverage (Shi et al., 2016; Shi and Dustdar, 2016). Since resources required to provide the service are limited on edge, an edge can federate with other edges and/or clouds to satisfy the users' requests. Through this federation, the efficiency in resource utilization and enlargement of capabilities of the federated entities can increase. Fig. 1 shows a two-layer cloud–edge federated system, where the upper layer and lower layer consist of multiple clouds (like google, amazon, etc.), and edges (such as AT&T, Chunghwa telecom, etc.), respectively. The federation manager (Kanwal et al., 2014) manages the federation between two entities in these systems. In the edge–edge federation, when an edge

* Corresponding author.
E-mail addresses: bkar@mail.ntust.edu.tw (B. Kar), ydlin@cs.nctu.edu.tw (Y.-D. Lin), laiyc@cs.ntust.edu.tw (Y.-C. Lai).

https://doi.org/10.1016/j.jnca.2023.103630
Received 26 June 2022; Received in revised form 4 January 2023; Accepted 26 March 2023
Available online 1 April 2023

Fig. 1. Cloud–edge federated model.

does not have the required resources to handle the requests, it can offload the tasks to other edges with enough resources (Cao et al., 2020). In the edge-cloud federation, the edge can offload the workloads to the clouds that require high computation resources or extra storage space (Chekired et al., 2018). Similarly, the cloud can reverse offload the latency-sensitive tasks (Villari et al., 2016).

Depending on different types of applications, input tasks/jobs can be grouped into three categories (Hsu et al., 2015) such as ultra-low, low, and loose latency jobs. Some applications whose end-to-end latencies are a few milliseconds, even in order of 100 microseconds, are ultra-low latency jobs (Nasrallah et al., 2018), for example, industrial applications (Wollschlaeger et al., 2017), tactile internet (Maier et al., 2016), virtual reality (Elbamby et al., 2018), automated guided vehicles (Osseiran et al., 2015), telesurgery (Tozal et al., 2013), control traffic in smart grid (Verma et al., 2016), etc. The tasks/jobs coming with loose deadlines, i.e., in seconds, are considered loose latency jobs and are reasonable in operational cost reduction in the cloud system (Goudarzi and Pedram, 2013). The jobs whose deadline is neither tight nor too loose are considered low or medium latency jobs. To the best of our knowledge, such categorization of jobs and using all categories of jobs together in the offloading model has not been discussed before.

In this paper, we considered two approaches to categorize the above-discussed job categorization. (1) Size of the jobs (Kim et al., 2018), where small jobs are coming under ultra-low or tight latency jobs, medium size jobs are part of low latency jobs, and larger jobs are part of loose latency jobs. (2) Priority of jobs (Choudhari et al., 2018; Gao and Moh, 2018), which is set irrespective of their size, i.e., a larger job can be in the category of tight latency, and a small job can have loose latency. Job categorization based on its size is a default approach; in such a scenario, we assume a larger job requires more resources and may cause the execution to get delayed, as we reserve the resources for the small jobs and offload the large job. However, job categorization based on priority is a special approach where job size will not affect resource allocation; if the job is on priority, the required resources are available, then they must be allocated.

When highly time-sensitive jobs are given as input to an edge, and the edge cannot handle such requests, it offloads the jobs to other edges horizontally (Cao et al., 2020). Similarly, when the edge receives loose latency jobs and/or jobs that consume very high storage space, it vertically offloads to a cloud (Chekired et al., 2018). However, when a cloud receives certain applications such as microservices (Thönes, 2015), it offloads to an edge to overcome the latency and data transfer cost (Villari et al., 2016). We define the key terms used in this paper with an example as follows. Let us consider two clouds, say C_1 and C_2, and two edges, say E_1 and E_2, where clouds are in the upper tier and edges are in the lower tier.

Definition 1. Federation Manager (FM) is the agent that is responsible for the federation agreement between two parties to whom both the service provider will expose their information such as available resources, based on which FM will take the decision (Francescon et al., 2017). The vertical federation between a cloud and an edge is managed by the cloud–edge federation manager, whereas the edge–edge federation manager manages the horizontal federation between two edges.

Definition 2. Horizontal Federation: Resource sharing agreement between two nodes in the same layer. For example, E_1 and E_2.

Definition 3. Vertical Federation: Resource sharing agreement between two nodes in different layers. For example, E_1 in the edge layer is federated with C_1 in the cloud layer.

Definition 4. Horizontal Offloading: When a request for E_1 in one layer is severed by E_2 in the same layer, i.e. when E_1 in the edge layer offloaded its request to E_2 in the same layer.

Definition 5. Vertical Offloading: When a request is for E_1 in one layer is severed by C_1 in another layer, i.e., E_1 offloaded its request to C_1.

Definition 6. Triangular Offloading: When a user of one service provider (say E_1) is served by another service provider (say C_1) and the users' inputs are offloaded from the user to C_1 via E_1, i.e., users give input to E_1 and E_1 offloads the tasks to C_1.

Definition 7. Non-triangular Offloading: When a user of one service provider (say E_1) is served by another service provider (say C_1) and based on the federation agreement, the FM will offload the users' inputs directly to C_1 without offloading via E_1.

Definition 8. Reverse Offloading: A vertical non-triangular offloading where a user of an upper layer node directly offloads its request to a node in the lower layer is called reverse offloading (Villari et al., 2016; Kar et al., 2022). For example, when a request for C_1 in the cloud layer is severed by E_1 in the edge layer i.e., the user of C_1 of the cloud layer offloaded its request directly to E_1 in the edge layer.

In this paper, we proposed cloud–edge federated architecture with omnidirectional offloading, where not only the edge can offload to the cloud, but also the cloud can offload back to the edge using reverse offloading, and the edges can offload to each other horizontally. By horizontal offloading, when an edge does not have enough resources to handle the requests that are time-sensitive and require less computing resources, such as ultra-low latency jobs, it can offload the tasks to nearby edges. Similarly, by vertical offloading, the edge can address the low computing resource issues by offloading the loose latency jobs to the clouds. By reverse offloading, as stated in Definition 8, the low and ultra-low latency jobs of the clouds are directly offloaded from the cloud users to the edges that are federated with the clouds, which will help to reduce both communication cost and communication latency. We set two criteria for offloading: (1) offloading based on the job's size and (2) offloading based on the job's priority, irrespective of its size. To the best of our knowledge, our work is the first to design the edge-cloud federation where offloading can be done in multiple directions (Kar et al., 2023), i.e., horizontal offloading, vertical offloading from edge-to-cloud and reverse offloading from cloud to edge. We also acknowledge the time sensitivity of different jobs and categorize them based on priority to minimize latency and communication costs. We develop an offloading algorithm using simulated annealing named the two-tier simulated annealing (TTSA) algorithm. Our detailed contributions are as follows.

1. We design a generic two-tier federated architecture enabling the clouds and edges to offload jobs omnidirectionally to satisfy the users' demand. In this architecture, not only can the edge offload to the cloud like many existing architectures, but also the cloud can reverse offload to the edge.

2. We propose an analytical model to minimize the total cost with latency as the constraint both from the cloud and edge layer perspective. We classify the latency as loose, low, and ultra-low. We considered all three types of latencies together as the constraints in this paper.
3. We use the modified simulated annealing algorithm to find the near-optimal solution globally of the proposed problem and compare the performance of our proposed architecture with three other existing architectures and compare the offloading algorithm with other existing algorithms.

The rest of the paper is organized as follows. In Section 2, we will discuss the related works, and in Section 3 our proposed architecture and optimization problem. We will present our solution in Section 4 and results in Section 5. Finally, we will discuss the conclusion in Section 6.

2. Related works

In this section, we will discuss some related work on the cloud–edge federation and how our work is different from the existing research.

Mashayekhy et al. (2014) proposed a game-theoretical model to reshape the business structure among cloud providers. These cloud providers can improve their dynamic resource scaling capabilities by establishing cooperation with the federation method. In this paper, a cloud federation mechanism is used to reduce resource utilization, as a result, the profit of the providers is maximized. A capacity-sharing mechanism in a federated cloud environment is proposed in Hassan et al. (2015), whose primary focus is global energy sustainability policy. By using game theory, the paper minimizes the overall energy by capacity sharing technique and promotes long-term individual profit of the service providers. Hammoud et al. (2020) proposed a genetic algorithm to study the problem of forming highly profitable federated clouds to maintain stability among federated entities when providers join or leave federations, which might affect the Quality of Service (QoS).

Tong et al. (2016) proposed a hierarchical edge cloud architecture to improve the performance of mobile computing by leveraging cloud computing and migrating mobile workloads for remote execution in the cloud. For the efficient utilization of resources, they proposed a workload placement algorithm to decide which programs are placed on which edge cloud servers and how much computational capacity is provisioned to execute that program. A new scheduling model in the two-tier cloud-fog architecture was proposed in Chekired et al. (2018) to process the industrial Internet of things (IIoT) data. They aimed to minimize communication and data processing delays in IIoT systems by deploying multiple servers for IIoT applications at the fog layer. Ren et al. (2019) investigate the cloud edge federation, where the tasks are partially processed at the edge and the cloud server to improve their efficiency with limited communication and computation capacities. They solve this non-convex capacity optimization problem to an equivalent convex optimization problem using the Karush–Kuhn–Tucker (KKT) conditions. Samanta and Chang (2019) proposed an adaptive service offloading (ASO) scheme for the MEC platform to maximize the profit for delay-sensitive and delay-tolerant services in the presence of multiple edge devices. Ascigil et al. (2021) proposed the optimal function provisioning and resource allocation problem by considering Function-as-a-Service in the edge-cloud systems. However, since the proposed fully-centralized, optimal models require accurate prediction of upcoming requests, they use heuristic algorithms from fully centralized to fully decentralized and use single cloudlet provisioning and coordinated provisioning to make predictions accurate.

The vertical federation and horizontal federation integration are discussed in Chen et al. (2017) to determine stable cooperation partners for the federation to improve efficiency where private clouds are federated with each other horizontally. These horizontal federated clouds are federated with the public clouds vertically. Cao et al. (2020) proposed an integrated service provisioning model by considering vertical and horizontal integration between multiple EIPs. In this edge federation, they formulated a linear programming problem of the provisioning process and took a variable dimension shrinking method to solve the optimization problem.

Table 1 lists the above-discussed papers based on the federation and presents a comparison of how the contribution of our work is different from theirs. While Mashayekhy et al. (2014), Hassan et al. (2015), and Hammoud et al. (2020) are based on horizontal federation, Tong et al. (2016), Ren et al. (2019), Samanta and Chang (2019), and Ascigil et al. (2021) are based on vertical federation and Chen et al. (2017), and Cao et al. (2020) discussed both horizontal and vertical federation together. Most papers in the table discussed the cost and capacity optimization issues, and their optimization is single tier only, whereas the article (Tong et al., 2016) is limited to only capacity optimization and considered the two-tier optimization between cloud and edge federation with one-way vertical offloading. The articles (Chekired et al., 2018; Ren et al., 2019) have addressed the latency issue and have considered two-tier optimization but based on an edge-cloud federation scenario where the workload is offloaded from the edge layer to the cloud layer. Similarly, Samanta and Chang (2019) also addressed both latency and cost issues in a two-tier cloud–edge architecture. However, non of these articles are considered different latency sensitivity. In this paper, we classify latency into three categories depending on the type of input from the users and put certain restrictions on which type of traffic should be handled by cloud or edge. We are also considering both horizontal and vertical federations together. All the above-discussed vertical federation papers are based on one-way offloading, i.e., edge-to-cloud offloading. However, in this paper, we are considering two-way offloading where not only the edge will offload to the cloud, but also the cloud can offload to edges.

3. System model

In this section, we will discuss our proposed cloud–edge federated architecture and cost optimization problem with latency as a constraint. In Table 2, we declared the notations and variables used to describe our model and formulate the optimization problem.

Nodes: C_i and E_j represent the ith and jth nodes in tier-2 and tier-1, respectively, where the tier-1 contains all the edges and tier-2 contains all the clouds.

Traffic: $\hat{\lambda}_i$ and λ_j are the traffic inputs to ith cloud and jth edge, respectively. The variables $\upsilon_{j,i}$, and $\hat{\upsilon}_{i,j}$, represent the vertical offloading from jth edge to ith cloud, and from ith cloud to jth edge, respectively, and horizontal offloading from jth edge to j'th edge is presented by $h_{j,j'}$.

Capacity: $\hat{\mu}_i$, and μ_j are the computing capacity of the servers in the ith cloud and jth edge, respectively. $\ddot{\mu}_{i,i}$ is the capacity of a server used by ith cloud for self-computation and $\breve{\mu}_{i,j}$ is the computing capacity of a server in ith cloud used by jth edge. $\bar{\mu}_{j,j'}$ is the capacity a server in jth edge is used by j'th edge, when $j = j'$, the capacity of jth edge server is used for self-computation. $\bar{\bar{\mu}}_{j,i}$ is the capacity of jth edge server used by ith cloud. The notations $\bar{b}_{C_i \leftrightarrow E_j}$, and $b_{E_j \leftrightarrow E_{j'}}$ are communication capacity between ith cloud and jth edge, and between jth edge and j'th edge. The communication capacity between user and ith cloud, and user and jth edge are represented by $\bar{\bar{b}}_{u,E_j}$, and $\hat{b}_{u,C_i}$, respectively.

Cost: The total cost of the cloud layer and edge layer are represented by $\hat{\tau}$, and τ, respectively. The unit computing cost of clouds is $\hat{c}$ and $\bar{\bar{c}}$ is the same for edges. m, $\bar{m}$, $\hat{m}$, and $\bar{\bar{m}}$ are the unit communication cost from user to edge, from user to cloud, between two edges, and between edge and cloud, respectively.

Table 1
Comparative analysis of related works.

References	Vertical federation	Horizontal federation	Reverse offloading	Cost	Latency	Latency categorization	Two-tier optimization	Methods
Mashayekhy et al. (2014)	×	✓	×	✓	×	×	×	GT
Hassan et al. (2015)	×	✓	×	✓	×	×	×	GT
Hammoud et al. (2020)	×	✓	×	✓	×	×	×	GA
Tong et al. (2016)	✓	×	×	×	✓	×	✓	SA
Chekired et al. (2018)	✓	×	×	×	✓	×	✓	SA
Ren et al. (2019)	✓	×	×	×	✓	×	✓	KKT
Samanta and Chang (2019)	✓	×	×	✓	✓	×	✓	ASO
Ascigil et al. (2021)	✓	×	×	×	✓	×	×	RAP
Chen et al. (2017)	✓	✓	×	✓	×	×	×	GT
Cao et al. (2020)	✓	✓	×	✓	✓	×	×	SEE
This paper	✓	✓	✓	✓	✓	✓	✓	TTSA

GT: Game Theory; GA: Genetic Algorithm; SA: Simulated Annealing; KKT: Karush–Kuhn–Tucker conditions; ASO: Adaptive Service Offloading; RAP: Resource Allocation and Provisioning algorithms; SEE: Service Provision for Edge Federation; TTSA: Two-tier Simulated Annealing.

Table 2
List of Commonly used variables and notations.

Notations	Descriptions
$C_i, 1 \le i \le q$	ith cloud in the upper tier (cloud layer) consists of q number of clouds
$E_j, 1 \le i \le p$	jth edge in the lower tier (edge layer) consists of p number of edges
	Traffic
$\hat{\lambda}_i, \lambda_j$	Traffic input to C_i node and E_j node
$\upsilon_{j,i}, \hat{\upsilon}_{i,j}, h_{j,j'}$	Vertical offloading from E_j node to C_i node, vertical offloading from C_i node to E_j node, and horizontal offloading from E_j node to $E_{j'}$ node
	Capacity
$\hat{\mu}_i, \ddot{\mu}_{i,i}, \ddot{\mu}_{i,j}$	Computing capacity of a server in C_i node, capacity of a server in C_i used for self-computation, and capacity of a server in C_i used by E_j node
$\mu_j, \bar{\mu}_{j,j'}, \bar{\bar{\mu}}_{j,i}$	Computing capacity of a server in E_j node, computing capacity of a server in C_j node used by $E_{j'}$ (when $j = j'$ capacity of E_j used for self task computation), capacity of a server in E_j used by C_i
$\bar{b}_{C_i \leftrightarrow E_j}, b_{E_j \leftrightarrow E_{j'}}$	Communication capacity between C_i and E_j, communication capacity between E_j and $E_{j'}$
$\bar{\bar{b}}_{u,E_j}$, $\hat{b}_{u,C_i}$	Communication capacity between user and E_j, communication capacity between user and C_i
	Cost
$\hat{\tau}, \bar{\bar{\tau}}$	Total cost of the cloud layer and edge layer
$\hat{c}, \bar{\bar{c}}$	Unit computing cost of the cloud and edge nodes
$m, \bar{m}, \hat{m}, \bar{\bar{m}}$	Unit communication cost from user to edge, from user to cloud, between two edges, and between edge & cloud
	Latency
$\hat{l}_i, \bar{\bar{l}}_j$	Computing latency at C_i and E_j
$l_{i,j}^{C\to E}, l_{j,i}^{E\to C}, l_{j,j'}^{E\to E}$	Communication latency from C_i to E_j, from E_j to C_i, from E_j to $E_{j'}$
$l_j^{u\to E}, l_i^{u\to C}$	Offloading latency from users to E_j, from users to C_i
L_{ul}, L_{lw}, L_{ls}	Maximum latency for ultra-low, low and loose latency traffic
	Distance
d^V, d^H	Distance between a cloud and an edge node, and between two edge nodes
$d^{u\to E}, d^{u\to C}$	Distance from user to edge, and from user to cloud

Latency: $\hat{l}_i$, and $\bar{\bar{l}}_j$ represent the computing latency at ith cloud, and jth edge, respectively. Whereas $l_{i,j}^{C\to E}$, $l_{j,i}^{E\to C}$, and $l_{j,j'}^{E\to E}$ are the communication latency from ith cloud to jth edge, from jth edge to ith cloud, and from jth edge to j'th edge, respectively. In this paper, we are considering three different types of latency, i.e., *ultra-low*, *low*, and *loose* latency. $l_j^{u\to E}$, $l_i^{u\to C}$ are the offloading latency from users to jth edge, and from users to ith cloud. The maximum latency limit for ultra-low, low, and loose latency traffics are represented by L_{ul}, L_{lw}, and L_{ls}, respectively.

Distance: d^V is the vertical distance between cloud and edge, and d^H is the horizontal distance between two edges. The distance from users to edges and clouds are $d^{u\to E}$, and $d^{u\to C}$, respectively. Speed of the light is c.

3.1. Two-tier cloud–edge federated architecture

In this section, we will discuss our proposed generic two-tier cloud–edge federated architecture shown in Fig. 2. The upper tier consists of clouds, and the lower tier consists of edges. The subscribers of cloud or edge can submit their requests to their respective service providers to avail of the services. In this architecture, as in Fig. 2, the edges and clouds are connected vertically, and all edges are connected horizontally. This means requests can be vertically offloaded from edges to clouds, reverse offloaded from cloud to edges, and horizontally offloaded from one edge to another. Since the clouds have unlimited capacity and coverage, we have not considered cloud-to-cloud horizontal offloading. In this paper, we consider the following assumption while modeling. Input jobs are assumed to be CPU-intensive jobs, hence do not require any input/output interruption during execution. A job can start its execution only after getting all the required resources. A task cannot be partially interrupted or offloaded from one entity to another after it starts processing. Most importantly, these tasks are assumed to be independent of each other.

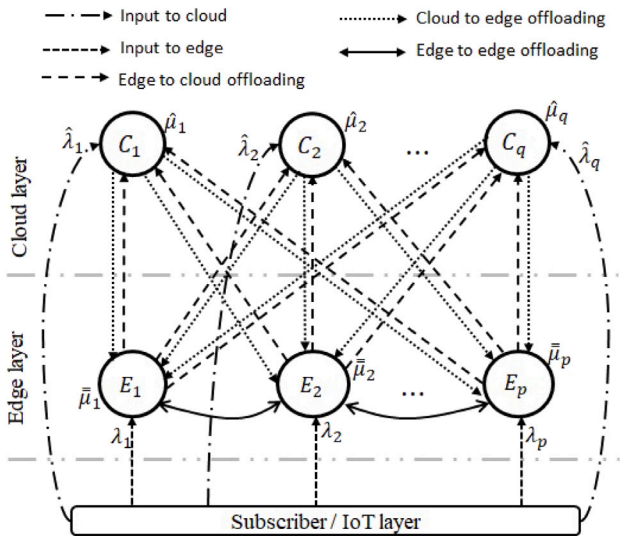


Fig. 2. Generic two-tier federated architecture.

3.2. Traffic distribution

3.2.1. Edge layer traffic distribution

Let us assume the probability of total traffic input to the jth edge is 1, which includes ultra-low, low, and loose latency traffic, *i.e.*, $\hat{P}_x(\lambda_j) + \hat{P}_y(\lambda_j) + \hat{P}_z(\lambda_j) = 1$, where $\hat{P}_x(\lambda_j)$, $\hat{P}_y(\lambda_j)$, and $\hat{P}_z(\lambda_j)$ are the probability of ultra-low, low, and loose latency traffics, respectively. The probability of offloaded ultra-low latency traffic from jth edge to all edges in the edge layer can be calculated as, $\hat{P}_x(\lambda_j) = [\hat{P}_{x(1)}(\lambda_j) + \hat{P}_{x(2)}(\lambda_j) + \cdots + \hat{P}_{x(p)}(\lambda_j)]$, where p is the number of edges. Similarly, $\hat{P}_z(\lambda_j)$ can be calculated as $\hat{P}_z(\lambda_j) = [\hat{P}_{z(1)}(\lambda_j) + \hat{P}_{z(2)}(\lambda_j) + \cdots + \hat{P}_{z(q)}(\lambda_j)]$ where q is the number of clouds in the cloud layer. Since the low latency traffic of jth edge can be offloaded both horizontally to edges and vertically to clouds, then $\hat{P}_y(\lambda_j)$ can be calculated as $\hat{P}_y^V(\lambda_j) + \hat{P}_y^H(\lambda_j)$, where $\hat{P}_y^V(\lambda_j)$ is the probability of vertically offloaded traffics and $\hat{P}_y^H(\lambda_j)$ is the horizontal offloaded traffics. $\hat{P}_y^V(\lambda_j)$ can be calculated as $[\hat{P}_{y(1)}^V(\lambda_j) + \hat{P}_{y(2)}^V(\lambda_j) + \cdots + \hat{P}_{y(q)}^V(\lambda_j)]$ and $\hat{P}_y^H(\lambda_j)$ as $[\hat{P}_{y(1)}^H(\lambda_j) + \hat{P}_{y(2)}^H(\lambda_j) + \cdots + \hat{P}_{y(p)}^H(\lambda_j)]$. Then the vertically offloaded total traffic from jth edge to ith cloud will be, $\upsilon_{j,i} = \hat{P}_{z(i)}(\lambda_j) * \lambda_j + \hat{P}_{z(i)}^V(\lambda_j) * \lambda_j$. Total horizontally offloaded traffic from jth edge to j'th edge will be, $h_{j,j'} = \hat{P}_{x(j')}(\lambda_j) * \lambda_j + \hat{P}_{x(j')}^H(\lambda_j) * \lambda_j$. In $h_{j,j'}$, if $j = j'$ then there will be no horizontal offloading.

3.2.2. Cloud layer traffic distribution

Let us assume the probability of total traffic input to the ith cloud is one that includes ultra-low, low, and loose latency traffics, *i.e.*, $P_x(\hat{\lambda}_i) + P_y(\hat{\lambda}_i) + P_z(\hat{\lambda}_i) = 1$, where $P_x(\hat{\lambda}_i)$, $P_y(\hat{\lambda}_i)$, and $P_z(\hat{\lambda}_i)$ are the probability of ultra-low, low latency, and loose latency traffic input to ith cloud, respectively. According to our architecture, the ultra-low latency traffics are processed at the edge layer due to their time sensitiveness; hence cloud always offloads such requests to the edges. $P_{x(j)}(\hat{\lambda}_i)$ represents probability of offloaded ultra-low latency traffic from ith cloud to jth edge. Then the total probability of offloaded ultra-low latency traffic from ith cloud C_i to the edge layer will be, $P_x(\hat{\lambda}_i) = [P_{x(1)}(\hat{\lambda}_i) + P_{x(2)}(\hat{\lambda}_i) + \cdots + P_{x(p)}(\hat{\lambda}_i)]$. $P_{y(j)}(\hat{\lambda}_i)$ represents probability of offloaded low latency traffic from ith cloud to jth edge and $P_{y(0)}(\hat{\lambda}_i)$ is the probability of same traffic type for ith cloud for self-computation. Then the probability of total low latency traffic input to a ith cloud will be, $P_y(\hat{\lambda}_i) = P_{y(0)}(\hat{\lambda}_i) + [P_{y(1)}(\hat{\lambda}_i) + P_{y(2)}(\hat{\lambda}_i) + \cdots + P_{y(q)}(\hat{\lambda}_i)]$. Then the total traffic offloaded from ith cloud to jth edge can be calculated as, $\hat{\upsilon}_{i,j} = P_{x(j)}(\hat{\lambda}_i) * \hat{\lambda}_i + P_{y(j)}(\hat{\lambda}_i) * \hat{\lambda}_i$.

3.3. Latency calculation

In our model, we assume both the clouds and edges consist of multiple servers, and each server has an equal capacity. However, the number of servers in a cloud is relatively high compared to an edge. Hence, a cloud has more computational capacity than an edge. To calculate the computation latency of the nodes, we apply the M/M/k queueing model, where k is the number of servers in the node. The value of k in the edges is kept fixed as the capacity of the edge is limited. However, since the capacity of the cloud is unlimited, we have considered the value of k is dynamic. The communication latency is calculated by the M/M/1 queueing model.

3.3.1. Computational latency calculation

Based on M/M/k model, the computational latency of ith cloud and jth edge are calculated in Eqs. (1) and (2), respectively, as follows.

$$\hat{l}_i = \frac{\mathbb{C}(k_i^C, J_i^C, \hat{\mu}_i)}{k_i^C \cdot \hat{\mu}_i - J_i^C} + \frac{1}{\hat{\mu}_i}, \text{where } J_i^C < k_i^C \cdot \hat{\mu}_i, \tag{1}$$

$$\bar{\bar{l}}_j = \frac{\mathbb{C}(k_j^E, J_j^E, \mu_j)}{k_j^E \cdot \mu_j - J_j^E} + \frac{1}{\mu_j}, \text{where } J_j^E < k_j^E \cdot \mu_j, \tag{2}$$

where k_j^E, and k_i^C are the number of servers present in an edge, and cloud node, respectively, $J_i^C = \hat{\lambda}_i - \sum_{j=1}^{m} \hat{\upsilon}_{i,j} + \sum_{j=1}^{m} \upsilon_{j,i}$ and $J_j^E = \lambda_j - \sum_{i=1}^{n} \upsilon_{j,i} + \sum_{i=1}^{n} \hat{\upsilon}_{i,j} - \sum_{j'=1}^{m} h_{j,j'} + \sum_{j'=1}^{m} h_{j',j}$. $\mathbb{C}(k, \lambda, \mu)$ is the Erlang $\mathbb{C}$ formula (Shortle et al., 2018), where $\mathbb{C}(k, \lambda, \mu) = \frac{1}{1+(1-\rho)\frac{k!}{(k\rho)^k}\sum_{n=0}^{k-1}\frac{k\rho^n}{n!}}$ and $\rho = \lambda/\mu$.

3.3.2. Communication latency calculation

From the users to the cloud/edge nodes, initial communication latency can be calculated as follows. The communication latency from the user to jth edge and from the user to ith cloud is calculated in Eqs. (3), and (4), respectively. Similarly, the communication latency from ith cloud to jth edge, from jth edge to ith cloud, and from jth edge to j'th edge are calculated in Eqs. (5), (6), and (7), respectively, as follows.

$$l_j^{u\to E} = \frac{1}{\bar{\bar{b}}_{u,E_j} - \lambda_j} + \frac{d^{u\to E}}{c}, where\ \lambda_j < \bar{\bar{b}}_{u,E_j}, \tag{3}$$

$$l_i^{u\to C} = \frac{1}{\hat{b}_{u,C_i} - \hat{\lambda}_i} + \frac{d^{u\to C}}{c}, where\ \hat{\lambda}_i < \hat{b}_{u,C_i}, \tag{4}$$

$$l_{i,j}^{C\to E} = \frac{1}{\bar{b}_{C_i\leftrightarrow E_j} - \hat{\upsilon}_{i,j}} + \frac{d^V}{c}, where\ \hat{\upsilon}_{i,j} < \bar{b}_{C_i\leftrightarrow E_j}, \tag{5}$$

$$l_{j,i}^{E\to C} = \frac{1}{\bar{b}_{C_i\leftrightarrow E_j} - \upsilon_{j,i}} + \frac{d^V}{c}, where\ \upsilon_{j,i} < \bar{b}_{C_i\leftrightarrow E_j}, \tag{6}$$

$$l_{j,j'}^{E\to E} = \frac{1}{b_{E_j\leftrightarrow E_{j'}} - h_{j,j'}} + \frac{d^H}{c}, where\ h_{j,j'} < b_{E_j\leftrightarrow E_{j'}}. \tag{7}$$

3.4. Triangular offloading problem

Triangular offloading is the case where users give their input to a node (service provider), say S_0. Node S_0 determines whether the input is appropriate to handle or not based on its available capacity and other constraints. If yes, then the input is handled by the node itself. Otherwise, S_0 will determine another service node say S_1, which has a federation agreement with S_0 and can handle the input. Then S_0 will offload the requests to S_1. This communication from the user to S_0 and then from S_0 to S_1 (i.e., $user \to S_0 \to S_1$) is called triangular offloading. In this section, for such triangular offloading, we present the optimization problem.

3.4.1. Objective of the edge layer

Let x_1, x_2, x_3, x_4, x_5, and x_6 are the variables in Eq. (8), presenting different costs of nodes in the edge layer.

$$\begin{cases} x_1 = \sum_{j=1}^{p} \lambda_j \cdot m, \\ x_2 = \sum_{j=1}^{p} k_j^E \cdot \bar{\bar{\mu}}_{j,j} \cdot \bar{\bar{c}}, \\ x_3 = \sum_{j=1}^{p} \sum_{j'=1}^{p} h_{j,j'} \cdot \hat{m}, \\ x_4 = \sum_{j=1}^{p} \sum_{j'=1}^{p} k_j^E \cdot \bar{\mu}_{j,j'} \cdot \bar{\bar{c}}, \\ x_5 = \sum_{j=1}^{p} \sum_{i=1}^{q} \upsilon_{j,i} \cdot \bar{\bar{m}}, \\ x_6 = \sum_{i=1}^{q} \sum_{j=1}^{p} k_i^C \cdot \bar{\bar{\mu}}_{j,i} \cdot \hat{c}. \end{cases} \tag{8}$$

Where x_1 shows the communication cost from the users to edges. The total self-computing cost of the edges is presented by x_2. The total communication cost between edges is shown by x_3. The x_4 shows the total computing cost of the edges while computation is done by other edges. The total communication cost from the edges to the clouds is presented by x_5. The x_6 presents the total computing cost of the edges while computation is done by the clouds. Then the objective function of the edges is to $minimize(\bar{\bar{\tau}})$, where,

$$\bar{\bar{\tau}} = x_1 + x_2 + x_3 + x_4 + x_5 + x_6, \tag{9}$$

subject to,

$$\begin{cases} l_j^{u \to E} + \bar{\bar{l}}_j < L_{ul}, & \text{(a)} \\ l_j^{u \to E} + l_{j,j'}^{E \to E} + \bar{\bar{l}}_{j'} < L_{ul}, & \text{(b)} \end{cases} \tag{10}$$

$$\begin{cases} l_j^{u \to E} + \bar{\bar{l}}_j < L_{lw}, & \text{(a)} \\ l_j^{u \to E} + l_{j,j'}^{E \to E} + \bar{\bar{l}}_{j'} < L_{lw}, & \text{(b)} \\ l_j^{u \to E} + l_{j,i}^{E \to C} + \hat{l}_i < L_{lw}, & \text{(c)} \end{cases} \tag{11}$$

$$l_j^{u \to E} + l_{j,i}^{E \to C} + \hat{l}_i < L_{ls}, \tag{12}$$

$$0 < \bar{\bar{\mu}}_{j,j} < \mu_j, \tag{13}$$

$$\sum_{j'=1}^{p} \bar{\mu}_{j,j'} + \sum_{i=1}^{q} \bar{\bar{\mu}}_{j,i} \le \mu_j, \forall j = 1, \ldots, p. \tag{14}$$

The Eqs. (10)–(12) present the latency constraints for the inputs given to all edges from users, where equations in (10) are for ultra-low latency, equations in (11) are for low latency, and Eq. (12) is for loose latency traffics. The inequalities in Eqs. (10)(a) and (11)(a) show the communication latency from the user to jth edge plus the computing latency at jth edge must be less than the ultra-low latency limit, and low latency limit, respectively. Eqs. (10)(b) and (11)(b) are the cases where the user submitted the jobs to jth edge, and jth edge offloaded the job to j'th edge. The inequalities in Eqs. (10)(b), and (11)(b) show the sum of the user to jth edge communication latency, jth edge to j'th edge communication latency and the computing latency at j'th edge must be less than the ultra-low latency limit, and low latency limit, respectively. Eqs. (11)(c), and (12) are the cases where the user submitted the jobs to jth edge, and jth edge offload the job to ith cloud. The inequalities in Eqs. (11)(c) and (12) show the sum of the user to jth edge communication latency, jth edge to ith cloud communication latency and the computing latency at ith cloud must be less than the low latency limit, and loose latency limit, respectively. The constraint in Eq. (13) presents an edge that can neither compute all its received requests by itself nor offload them entirely to others. The sum of self-computation and horizontal offloading capacity plus vertical offloading capacity must be less than the total input to the edges presented in Eq. (14).

3.4.2. Objective of the cloud layer

Let y_1, y_2, y_3, and y_4 be the variables in Eq. (15), presenting different costs of nodes in the cloud layer.

$$\begin{cases} y_1 = \sum_{i=1}^{q} \hat{\lambda}_i \cdot \hat{m}, \\ y_2 = \sum_{i=1}^{q} k_i^C \cdot \ddot{\mu}_{i,i} \cdot \hat{c}, \\ y_3 = \sum_{i=1}^{q} \sum_{j=1}^{p} \hat{\upsilon}_{i,j} \cdot \bar{\bar{m}}, \\ y_4 = \sum_{j=1}^{p} \sum_{i=1}^{q} k_j^E \cdot \breve{\mu}_{j,i} \cdot \bar{\bar{c}}. \end{cases} \tag{15}$$

Where y_1 shows the communication cost from the users to the clouds. The total self-computing cost of the clouds is presented by y_2. The total communication cost from the clouds to the edges is presented by y_3. The y_4 presents the total computing cost of the clouds while computation is done by the edges. Then the objective function of the clouds is to $minimize(\hat{\tau})$, where,

$$\hat{\tau} = y_1 + y_2 + y_3 + y_4, \tag{16}$$

subject to,

$$l_i^{u \to C} + l_{i,j}^{C \to E} + \bar{\bar{l}}_j < L_{ul}, \tag{17}$$

$$\begin{cases} l_i^{u \to C} + \hat{l}_i < L_{lw}, & \text{(a)} \\ l_i^{u \to C} + l_{i,j}^{C \to E} + \bar{\bar{l}}_j < L_{lw}, & \text{(b)} \end{cases} \tag{18}$$

$$l_i^{u \to C} + \hat{l}_i < L_{ls}, \tag{19}$$

$$\begin{cases} 0 < \ddot{\mu}_{i,i} < \hat{\mu}_i, \\ P_z(\hat{\lambda}_i) < 1, \end{cases} \tag{20}$$

$$\ddot{\mu}_{i,i} + \sum_{j=1}^{p} \breve{\mu}_{i,j} \le \hat{\mu}_i, \forall i = 1, \ldots, q. \tag{21}$$

The Eqs. (17)–(19) presents the latency constraints for the inputs given to all clouds from the users, where Eq. (17) is for ultra-low latency, equations in (18) are for low latency, and Eq. (19) is for loose latency traffics. Eqs. (17), and (18)(b) are the cases where the user submitted the jobs to ith cloud, and ith cloud offloaded the jobs to jth edge. The inequalities in Eqs. (17) and (18)(b) show the sum of the user to ith cloud communication latency, ith cloud to jth edge communication latency and the computing latency at jth edge must be less than the ultra-low latency limit, and low latency limit, respectively. The inequalities in Eqs. (18)(a) and (19) show the user to ith cloud communication latency plus the computing latency at ith cloud must be less than the low latency limit, and loose latency limit, respectively. The constraints in Eq. (20) present any cloud can neither compute all its request by itself nor offload them entirely to others, provided all the traffics to the cloud are not loose latency traffics. The sum of self-computation and vertical offloading capacity must be less than the total input to the clouds discussed in Eq. (21).

3.5. Non-triangular offloading problem

As discussed in Section 3.4, in triangular offloading, when two service nodes S_0 and S_1 have a federation agreement, S_0 receives the request from the user and offload to S_1. However, in non-triangular offloading, if S_0 and S_1 have a federation agreement, then the user of S_0 can directly submit its request to S_1 by the FM. This type of offloading is called non-triangular offloading. In this subsection, we modify our previously proposed optimization problem for such non-triangular offloading.

3.5.1. Non-triangular latency calculation

In a non-triangular offloading scenario, the communication latency is estimated as follows. The vertical offloading latency from the user to ith cloud, where the job input is for jth edge, is estimated as, $l_{j,i}^{u\to EC} = \frac{1}{\hat{b}_{u,C_i} - \upsilon_{j,i}} + \frac{d^{u\to C}}{c}$, where $\upsilon_{j,i} \le \hat{b}_{u,C_i}$. The horizontal offloading latency from the user to j'th edge, where the job input is for jth edge, is calculated as, $l_{j,j'}^{u\to EE} = \frac{1}{\bar{\bar{b}}_{u,E_{j'}} - h_{j,j'}} + \frac{d^{u\to E}}{c}$, where $h_{j,j'} \le \bar{\bar{b}}_{u,E_{j'}}$. The reverse offloading latency from the user to jth edge, where job input is for ith cloud, will be, $l_{i,j}^{u\to CE} = \frac{1}{\bar{\bar{b}}_{u,E_j} - \hat{\upsilon}_{i,j}} + \frac{d^{u\to E}}{c}$, where $\hat{\upsilon}_{i,j} \le \bar{\bar{b}}_{u,E_j}$.

3.5.2. Modified objective of the edge layer

The new cost in the edge layer required the following cost variables, including some variables from Eq. (8).

$$\begin{cases} x_7 = \sum_{j=1}^{p} \sum_{j'=1}^{p} h_{j,j'} \cdot m, \\ x_8 = \sum_{j=1}^{p} \sum_{i=1}^{q} \upsilon_{j,i} \cdot \bar{m}. \end{cases} \tag{22}$$

In Eq. (22), x_7 is the case where the input traffic from the user is for jth edge. However, due to the decision of edge–edge FM, the traffic is offloaded directly to the j'th edge. The x_7 represents the total horizontal offloading cost between the edges. The x_8 is the case where the input traffic from the user is for jth edge; however, due to the decision of cloud–edge FM, the traffic is offloaded directly to the ith cloud. Then the modified objective function of the edge layer will be to $minimize(\bar{\bar{\tau}}_1)$, where,

$$\bar{\bar{\tau}}_1 = x_4 + x_6 + x_7 + x_8, \tag{23}$$

subject to,

$$l_{j,j'}^{u\to EE} + \bar{\bar{l}}_{j'} < L_{ul}, \tag{24}$$

$$\begin{cases} l_{j,j'}^{u\to EE} + \bar{\bar{l}}_{j'} < L_{lw}, & \text{(a)} \\ l_{j,i}^{u\to EC} + \hat{l}_i < L_{lw}, & \text{(b)} \end{cases} \tag{25}$$

$$l_{j,i}^{u\to EC} + \hat{l}_i < L_{ls}, \tag{26}$$

(10)(a), (11)(a), (13), and (14).

The Eqs. (24) and (25)(a) are the cases where jobs from the user of jth edge are directly offloaded to j'th edge. The inequalities in Eqs. (24) and (25)(a) present the sum of the user to j'th edge communication latency, and the computing latency at j'th edge must be less than the ultra-low latency limit, and the low latency limit, respectively. Eqs. (25)(b) and (26) are the cases where jobs from the user of jth edge are directly offloaded to ith cloud. The inequalities in Eqs. (25)(b) and (26) present the sum of the user to ith cloud communication latency, and the computing latency at ith cloud must be less than the low latency limit, and loose latency limit, respectively.

3.5.3. Modified objective of the cloud layer

The new cost in the cloud layer required the following cost variables, including some variables from Eq. (15).

$$\begin{cases} y_5 = \sum_{i=1}^{q} \sum_{j=1}^{p} \hat{\upsilon}_{i,j} \cdot m, \\ y_6 = \sum_{i=1}^{q} (\hat{\lambda}_i - \sum_{j=1}^{p} \hat{\upsilon}_{i,j}) \cdot \bar{m}. \end{cases} \tag{27}$$

In Eq. (27), y_5 is the case where the input traffic from the user is for ith cloud. However, based on the decision of the cloud–edge federation manager, the traffic is offloaded directly to jth edge. The y_5 represents the total reverse offloading cost from the cloud to the edges. The y_6 shows the communication cost of the remaining traffic that from the user to the clouds. Then the new objective function of the clouds will be $minimize(\hat{\tau}_1)$, where

$$\hat{\tau}_1 = y_2 + y_4 + y_5 + y_6, \tag{28}$$

subject to,

$$l_{i,j}^{u\to CE} + \bar{\bar{l}}_j < L_{ul}, \tag{29}$$

$$l_{i,j}^{u\to CE} + \bar{\bar{l}}_j < L_{lw}, \tag{30}$$

(18)(a), (19), (20), and (21).

The Eqs. (29) and (30) are the cases where jobs from the user of ith cloud are reverse offloaded to jth edge. The inequalities in Eqs. (29) and (30) present the sum of the user to jth edge communication latency, and the computing latency at jth edge must be less than the low latency limit, and loose latency limit, respectively.

3.6. Objective of OMNI architecture

After analyzing both the triangular offloading problem in Section 3.4 and the non-triangular offloading problem in Section 3.5, we found both the offloading scenarios have their advantages as well as disadvantages. For example, triangular offloading increases the communication latency, whereas non-triangular offloading reduces the communication latency but increases the burden on the federation manager to make an optimal decision. Hence, we adopted both offloading scenarios in our OMNI architecture. For the edge layer, we consider the triangular offloading as in Eq. (9), where the edge will offload the task to other edges horizontally and the cloud vertically. Because in this triangular offloading, the communication latency will not make more difference than the non-triangular offloading due to the short distance from the user to the edges and between edges. However, we consider the non-triangular offloading for the cloud layer as in Eq. (28), where the cloud will reverse offload the time-sensitive tasks to the edges. By this reverse offloading, the communication latency of non-triangular offloading will be relatively less as compared to triangular offloading since the distance "from user to cloud and cloud to edge (triangular) is much higher than simply "from user to edge (non-triangular). Since the main goal of our model is to reduce the total cost of the system, with given latency as the constraint, the modified problem is as follows. The modified objective function is to minimize the total system cost, i.e., *minimize* (τ_{total}), where

$$\tau_{total} = a \cdot \bar{\bar{\tau}} + (1 - a) \cdot \hat{\tau}_1, \tag{31}$$

subject to,

(10)(a), (10)(b), (11)(a), (11)(b), (11)(c), (12), (13), (14), (18)(a), (19), (20), (21), (29), and (30).

4. Problem formulation

In the solution, we describe the simulated annealing (SA) algorithm (Johnson et al., 1989), which performs a probabilistic technique to find a globally optimal solution. Specifically, it is a meta-heuristic approach to approximate global optimization in an ample search space for an optimization problem and is used when the search space is discrete.

4.1. Two-tier simulated annealing

The two-tier simulated annealing (TTSA) process is a modified version of SA described in Algorithm 1. In each iteration, we generate a new state x_{new} from the previous state x_{old} and compute the cost for both the tiers using $Cost_1()$, *i.e.*, in Eq. (9) and $Cost_2()$, *i.e.*, in Eq. (28). The differences between the old and new costs of the edge tier and cloud tier denoted as Δ_1 and Δ_2, respectively, are computed. The x_{new} will be immediately accepted if $\Delta_1 \le 0$ and $\Delta_2 \le 0$. If $\Delta_1 \le 0$ and $\Delta_2 > 0$, then we have a probability $e^{-\Delta_2/T}$ to accept x_{new}, where T is the simulated temperature and decreases in each step of the iteration by a cooling parameter α $(0 < \alpha < 1)$. However, if $\Delta_1 > 0$ and $\Delta_2 \le 0$, we

Algorithm 1 TTSA Algorithm

Initially: Randomly generate initial solution x_{old}
Set: $T \leftarrow T_{max}$
while $T > T_{min}$ **do**
  Generate new solution x_{new}
  $\Delta_1 = Cost_1(x_{new}) - Cost_1(x_{old})$
  $\Delta_2 = Cost_2(x_{new}) - Cost_2(x_{old})$
  if $(\Delta_1 \leq 0)$ & $(\Delta_2 \leq 0)$ **then**
    accept $x_{old} \leftarrow x_{new}$
  else
    if $(\Delta_1 \leq 0)$ & $(\Delta_2 > 0)$ **then**
      accept $x_{old} \leftarrow x_{new}$ with probability $e^{-\Delta_2/T}$
    else
      if $(\Delta_1 > 0)$ & $(\Delta_2 \leq 0)$ **then**
        accept $x_{old} \leftarrow x_{new}$ with probability $e^{-\Delta_1/T}$
      else
        if $(\Delta_1 > 0)$ & $(\Delta_2 > 0)$ **then**
          accept $x_{old} \leftarrow x_{new}$ with probability $e^{-(\Delta_1+\Delta_2)/T}$
        end
      end
    end
  end
  $T \leftarrow \alpha \cdot T$
end

can accept x_{new} with probability $e^{-\Delta_1/T}$. The x_{new} will acceptable with probability $e^{-(\Delta_1+\Delta_2)/T}$ if $\Delta_1 > 0$ and $\Delta_2 > 0$. The length of the iterations in SA is determined by the initial temperature T_{max}, the terminating temperature T_{min}, and the cooling parameter α. At the beginning, T equals to T_{max}. T then decreases in each iteration, when $T = T_{min}$ the SA iterations terminate.

This algorithm applies randomness with some restrictions for the initial solution, which is initialized as follows. As discussed in our proposed architecture, our input jobs are three different types. Only edges handle the ultra-low latency jobs, only clouds handle loose latency jobs, and both edges and clouds can handle low latency jobs. For a cloud, when the inputs are loose latency jobs, they will have kept by the cloud, whereas if the job is ultra-low latency, it will offload to any edge randomly. For low latency jobs, the cloud will have kept 70 percent by itself and offload the rest to the edges randomly. For an edge, if the inputs are loose latency jobs, it will offload randomly to the clouds. If the inputs are low latency jobs, then 50 percent of jobs are kept on the edge, and the other 50 percent will be offloaded to other edges and clouds. If the inputs are ultra-low latency jobs, then 70 percent will be kept by the edge, and the other 30 percent will be randomly offloaded to other edges.

We use three processes (swapping, reversion, and insertion) in this simulated annealing to make the offloading decision. In the swapping process, two federated nodes can swap their jobs with each other. And in the insertion process, a job from one node is taken and inserted in another node. While invoking these processes, we impose some restrictions not to violate offloading criteria set in the systems.

1. In the swapping process, "two loose latency jobs/two low latency jobs/one low and one loose latency jobs" between two clouds is applicable if and only if those two jobs are offloaded from edges. Non-offloaded jobs cannot be swapped between two clouds as there is no cloud-to-cloud offloading. The swapping of two low latency jobs between an edge and a cloud is applicable, provided the selected job in the edge is not offloaded from any other clouds. The swapping can be performed between two edges if the following condition holds. (a) if both the jobs are low latency jobs, (b) if both the jobs are ultra-low latency jobs, and (c) if one is a low latency job and the other is an ultra-low latency job.

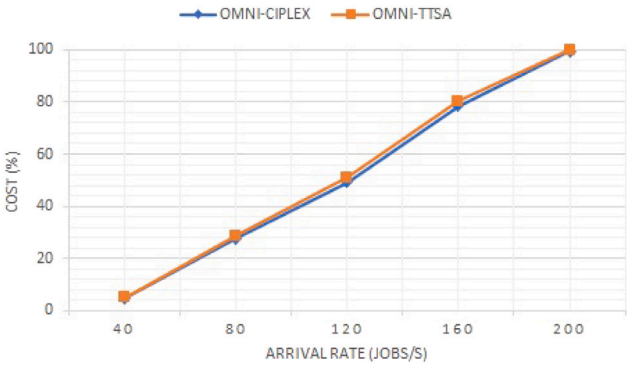


Fig. 3. Performance comparison between heuristic and optimal solutions.

2. In the reversion process, after two jobs are selected for swapping, the necessary swapping condition (as discussed above) must be satisfied between intermediate jobs in the intermediate nodes. If there is any violation of the swapping rule between two intermediate nodes, then swapping of those jobs will not be performed.
3. In the insertion process, a loose latency job cannot be inserted in an edge, and an ultra-low latency job cannot be inserted in a cloud. A loose/low latency job, initially assigned to one cloud, can be inserted into a new cloud if and only if the job was offloaded from an edge.

Consequently, like the SA process, the TTSA also has no bounded time complexity. Instead, its time complexity is determined by the cooling parameter α.

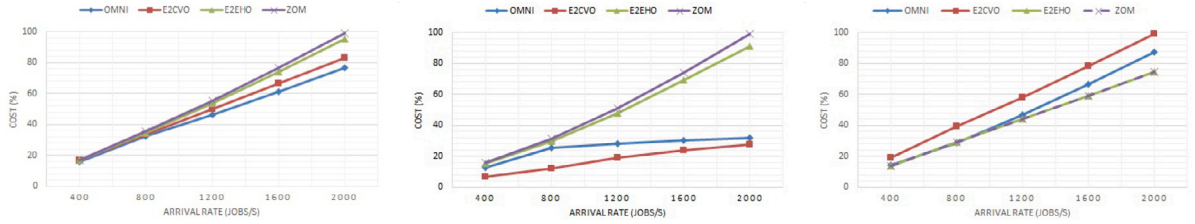
5. Numerical results

5.1. Experiment setup

For the experiment, we considered a cloud–edge federated network consisting of five clouds, with five edges geographically distributed. The capacity of a server on edge is 2 GB, and each edge consists of 10 servers. The capacity of a server in the cloud is 4 GB, and the number of servers in the cloud node is unlimited. The maximum distance from the user to an edge, the user to a cloud, between two edges, and between an edge and a cloud are 10 KM, 1000 KM, 100 KM, and 1000 KMs, respectively. The size of the input jobs is taken randomly between 1 KB to 250 MB. Based on the size, the jobs are categorized into three groups. The unit cost of communication and computation is considered in terms of money. The detailed parameters for our experiment are discussed in Table 3. We conducted simulations to evaluate the performance of OMNI architecture and the proposed TTSA algorithm. To calculate the total cost in Eq. (31), we set the value of a to 0.5 since the job inputs given to both the edge and cloud layers are equal, and the cooling parameter α value is set to 0.99. Our simulation runs on a Windows 11 operating system running on an i7 processor with 16 GB RAM. We carried out our simulation by using Python and compared it with CPLEX (Cplex, 2009) for the optimal result. Fig. 3 shows the cost analysis of TTSA and CPLEX results for omnidirectional architecture. This result is of cost analysis based on job sizes, and the input jobs were taken randomly from 490 to 510 KB. Fig. 3 shows the total cost result from the TTSA algorithm and CPLEX result are nearly similar.

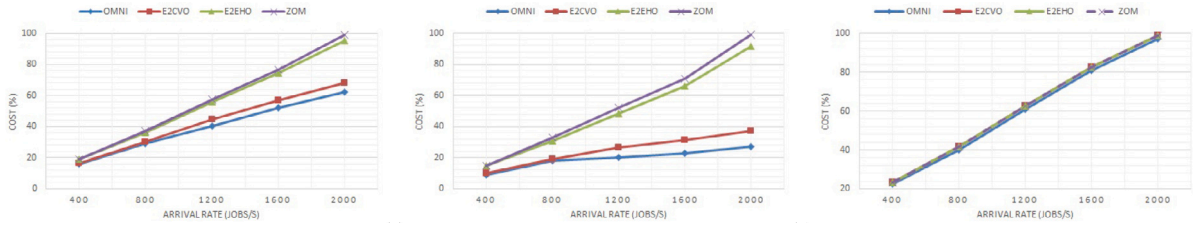
5.2. Performance analysis

In this section, we compared the performance with three other architectures: (1) zero offloading model (ZOM), (2) edge-to-cloud vertical



(a) total cost (b) edge cost (c) cloud cost

Fig. 4. The total cost of the edge-cloud systems per number of jobs per second in terms of job priority.



(a) total cost (b) edge cost (c) cloud cost

Fig. 5. The total cost of the edge-cloud systems per number of jobs per second in terms of job size.

Table 3
Parameter settings.

Items	Values
No. of clouds	5
No. of edges	5
Capacity and bandwidth	
Capacity of an edge server	2 GB
Capacity of a cloud server	4 GB
No. of servers in each edge	10
No. of servers in each cloud	unlimited
Job size	1KB–250 MB
small job	job size ≤100 KB
medium job	100 KB<job size<25 MB
Large job	25 MB≤job size≤250 MB
Distance	
User to edge distance	1 KM
User to cloud distance	1000 KM
Edge to edge distance	10 KM
Edge to cloud distance	1000 KM
Computing cost	
cloud	20 money units/Mcycles
edge	15 money units/Mcycles
Communication cost	
Cloud–edge	3 money units/MBytes
Edge–edge	2 money units/MBytes
User to edge	1 money units/MBytes
User to cloud	4 money units/MBytes

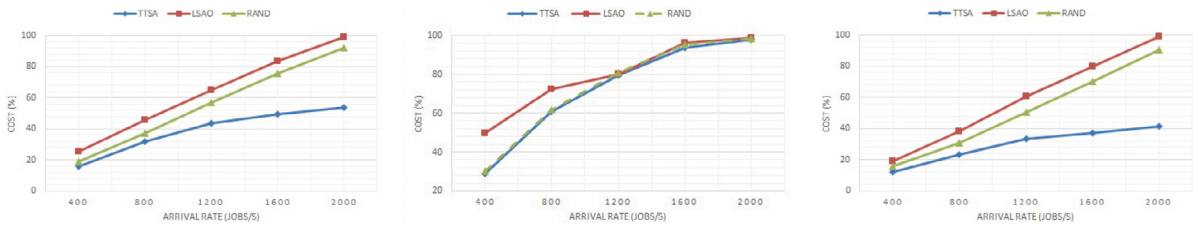
offloading (E2CVO) as in Tong et al. (2016), and (3) edge-to-edge horizontal offloading (E2EHO) as in Cao et al. (2020). In this experiment, we considered two types of input scenarios. First, the offloading is based on the job's size, where a small size job is treated as an ultra-low latency job that an edge must process, and a large job is treated as a loose latency job that a cloud must process. (2) Second, offloading is based on the job's priority, irrespective of its size. A priority value (low, mid, high) was randomly assigned to each job, irrespective of their size. In this scenario, a job of any size can be an ultra-low, low, or loose latency job.

5.2.1. Total cost analysis – OMNI vs. Others

Figs. 4 and 5 show the cost analysis of different architectures with the increase in the number of input jobs. Fig. 4 shows the system performance where the offloading decision of the jobs is considered based on their priority irrespective of their size. In contrast, Fig. 5 shows the offloading decision of the jobs is taken based on the size of the jobs as given in Table 3. In both figures, we presented the total cost of the systems in Figs. 4(a) and 5(a), and the intermediate results are the edge layer cost in Figs. 4(b) and 5(b), and the cloud layer cost in Figs. 4(c) and 5(c). If we consider the total cost as given presented in Figs. 4(a) and 5(a), in both cases, our OMNI architecture saves the total cost by 7%–10%, 20%–30%, and 25%–40% compared to the E2CVO, E2EHO, and ZOM architectures, respectively, because of its horizontal, vertical, and reverse offloading mechanisms. Due to reverse offloading, the highly time-sensitive jobs of clouds are redirected to edges, which reduces the communication cost. Due to horizontal federation, the edge extends its computing capacity in the edge layer. The E2CVO saves more cost than ZOM and E2EHO as the edges offload the loose latency and some low latency jobs to the clouds, reducing the storage burden on the edges and making the computation faster as the cloud has unlimited computing resources. Let us compare the ZOM and E2EHO performance. The E2EHO saves more cost than ZOM due to its edge-to-edge horizontal federation as it extends the computing capacity and provides the service faster than ZOM. For the edge cost, in the case where offloading takes place based on priority irrespective of job size, the cost of OMNI and E2CVO is lesser than others. The edge cost of E2CVO is better than OMNI, as shown in Fig. 4(b), because of the reverse offloading of some large jobs that have high priority offloaded to the edges. However, in Fig. 5(b), the edge cost of OMNI is lower than E2CVO as the jobs that are offloaded to the edges are smaller, where the offloading decision is taken based on the job size. For the cloud cost, OMNI architecture's cost is better than E2CVO and similar to others in low input, as shown in Fig. 4(c), where the offloading decision is based on the priority of the jobs. However, the cloud cost of all architectures is nearly similar in the case where the offloading decision is taken based on the job size, as shown in Fig. 5(c).

5.2.2. Total cost analysis – TTSA vs. Others

In Fig. 6, we compared the performance of our proposed TTSA algorithm with two other algorithms: (1) local servers are overloaded (LSAO) used in Mehmeti and Spyropoulos (2016), (2) random (RAND) offloading scheme used in Chen and Hao (2018). In this LSAO scenario, all jobs coming to the edge are directly served by the edges locally, until local servers become overloaded. When the edge becomes overloaded, then tasks are offloaded to the other edges and clouds (Tong et al.,



(a) total cost (b) edge cost (c) cloud cost

Fig. 6. Performance comparison of our proposed TTSA algorithm with other algorithms in terms of cost analysis.

Fig. 7. Average latency analysis in different architectures based on job's priority and job's size.

2016). The jobs that are coming to the cloud will get served by the cloud as the cloud will never become overloaded due to its unlimited capacity. In the RAND offloading scenario, wherever the jobs arrive, they will be processed in the host node or offloaded to other nodes; that decision was taken randomly. Fig. 6(a) shows that our TTSA algorithm performs nearly 45%–55% better than both RAND and LSAO algorithms in terms of the total cost since the offloading decision of TTSA is taken based on the job's priority. As shown in Fig. 6(b), all algorithms have similar costs for the edge cost; only LSAO has a high cost during low inputs. However, for the cloud cost, in TTSA, the cloud reverse offloads the high-priority jobs to the edges, whereas in LSAO, jobs from the cloud are never offloaded to the edges as the cloud has unlimited capacity, and in the RAND offloading decision is taken randomly. Hence, TTSA's cloud cost is lower than others, as shown in Fig. 6(c).

5.2.3. Latency analysis – OMNI vs. others

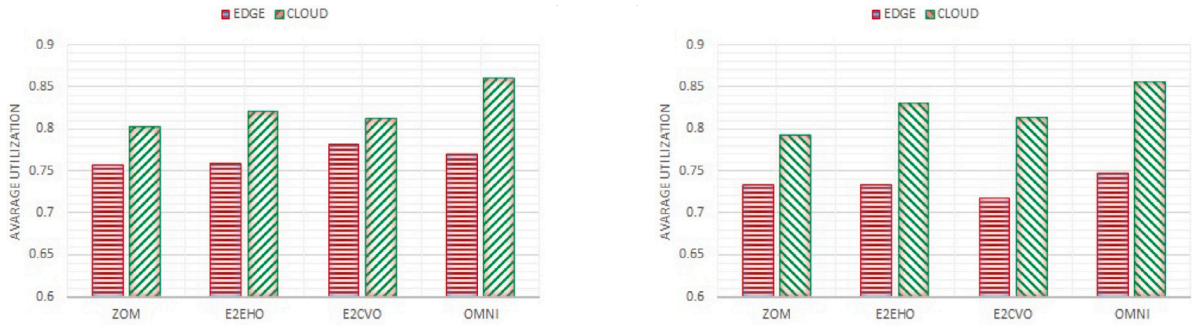
Fig. 7 shows the average latency of the jobs in OMNI architecture with others in both offloading scenarios. In both cases, the latency of the ZOM model has the worst performance due to the long waiting time as they do not have the offloading facility. E2EHO architecture is better than ZOM because they can horizontally offload the nearby edges when they require extra resources to handle the requests. However, they still have a long waiting time due to the limited capacity at the edges. The reason for the high waiting time in ZOM and E2EHO architectures is limited resources on the edges. When requests are made, edges do not have the resources to handle those requests in the edges layer and also do not have the facility to offload them to the clouds. However, the performance of E2CVO and OMNI is much better than the other two architectures as they have very negligible waiting time due to the option to offload to the clouds that have unlimited resources. Since the clouds have unlimited computing resources and storage capacity, as soon as the inputs are given to the clouds, the required resources get allocated to the jobs for computation. The latency of OMNI is less than E2CVO due to its reverse offloading architecture. It reverse offloads the highly time-sensitive jobs to the edges, reducing communication time. Also, due to OMNI's horizontal federation, it extends its computation capacity in the edge layer. As a result, it is able to handle more jobs in a limited time and reduce the latency compared to E2CVO.

5.2.4. Average utilization – OMNI vs. others

Fig. 8 shows the average utilization of the active servers in the edge and cloud layers in both cases, i.e., offloading based on the job's priority in Fig. 8(a) and offloading based on the job's size in Fig. 8(b). The utilization of the cloud servers in ZOM and E2EHO are relatively similar in both Fig. 8(a) and (b), as there is no offloading to the cloud in both cases. It only processes the tasks that are received by the cloud from its users. However, the utilization of the edge servers in E2EHO is better than ZOM as an edge can offload the tasks to nearby edges in case of excessive load. The utilization of the cloud servers in E2CVO architecture is better than others, as shown in Fig. 8(a), where offloading takes place based on the job's priority. Whereas in Fig. 8(b), where offloading is based on the job's size, cloud servers' utilization is less due to large fragmentation as only large-sized jobs are offloaded from the edges to the clouds. In both cases (priority and size), the edge servers' utilization in E2EHO is even better than in E2CVO architecture. In OMNI architecture, the heavily loaded edges offload their excessive jobs either to clouds vertically or to other edges horizontally. Similarly, the cloud reverse-offload the highly time-sensitive jobs to the edges. The utilization of OMNI architecture is relatively better compared to other architectures both in edge and cloud layer nodes as it tries to accommodate more jobs within the specified amount of resources to maximize resource utilization both in edges and clouds.

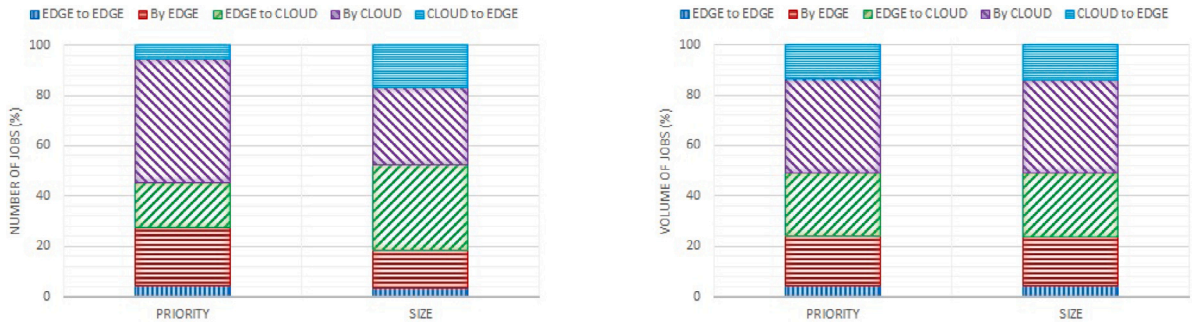
5.2.5. Offloading ratio – number of jobs vs. volume of jobs

Fig. 9 shows the offloading ratio of OMNI architecture in both cases, i.e., based on the job's priority and size. Fig. 9(a), and (b) present the offloading ratios in terms of the number of jobs, and volume of jobs, respectively. Let us compare the offloading ratios in terms of the number of jobs in Fig. 9(a), in case of priority. The number of jobs offloaded from cloud to edge is only about 5%–6% as the offloading takes place based on the job's priority, irrespective of the size of the jobs. Whereas in the case of size, the number of jobs offloaded from the cloud to the edge is significantly more, i.e., 17%–18%, as there are smaller size jobs. Similarly, the number of jobs offloaded from the edge to the cloud is only 17%–18% in the case of offloading based on the job's priority, whereas it is 33%–34% when offloading is based on the job's size. As Fig. 9(a) shows, when we calculate the offloading ratio in terms of the number of jobs, we found a significant difference in the offloading ratios between the offloading based on the job's priority and offloading based on the job's size. However, that is not the case when we calculate the offloading ratios in terms of the volume of the jobs. Fig. 9(b) shows the offloading ratios in OMNI architecture in terms of the volume of the jobs for both cases, i.e., based on the job's priority and size. As indicated, the offloading ratios in both offloading scenarios are nearly similar.



(a) priority (b) size

Fig. 8. Average utilization of active servers in edge and cloud nodes based on job's priority and job's size.



(a) based on number of jobs (b) based on volume of jobs

Fig. 9. Offloading ratio in the OMNI architecture.

6. Conclusions

In this paper, we proposed a two-tier federated cloud–edge architecture with omnidirectional offloading. In this federated architecture, the edges can horizontally offload their requests to each other, and not only can edges vertically offload their request to the clouds, but also clouds can reverse offload their highly time-sensitive requests to the edges for faster services. We proposed offloading optimization problem to minimize the total cost of both the cloud and edge layer with given latency constraints, where offloading takes place based on the job's size and priority. We used a modified SA algorithm to find the global optimum results. The results show our proposed OMNI architecture reduces the total cost by 7%–10%, 20%–30%, and 25%–40% compared to E2CVO, E2EHO, and ZOM architectures, respectively. Our proposed TTSA algorithm also reduces the cost by 45%–55% compared to LSAO and RAND offloading algorithms. It also increases utilization in the edges, and the average latency in our architecture is relatively less compared to other existing architectures.

CRediT authorship contribution statement

Binayak Kar: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Writing – original draft, review & editing, Supervision, Project administration, Funding. **Ying-Dar Lin:** Conceptualization, Investigation, Writing – review & editing, Supervision, Project administration, Funding. **Yuan-Cheng Lai:** Conceptualization, Investigation, Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article

Acknowledgments

This work was supported by the Ministry of Science and Technology (MOST), Taiwan, under Grant 109-2221-E-011-104-MY3. A preliminary version of this paper appears in the proceedings of IEEE ICC 2020 (Kar et al., 2020). We thank Yoel Pater Siswojo, Brandon Wymer Pramana, and Primatar Kuswiradyo for their help with our simulations.

References

Ascigil, O., Tasiopoulos, A., Phan, T.K., Sourlas, V., Psaras, I., Pavlou, G., 2021. Resource provisioning and allocation in function-as-a-service edge-clouds. IEEE Trans. Serv. Comput. 15 (4), 2410–2424.

Cao, X., Tang, G., Guo, D., Li, Y., Zhang, W., 2020. Edge federation: Towards an integrated service provisioning model. IEEE/ACM Trans. Netw. 28 (3), 1116–1129.

Chekired, D.A., Khoukhi, L., Mouftah, H.T., 2018. Industrial IoT data scheduling based on hierarchical fog computing: A key for enabling smart factory. IEEE Trans. Ind. Inform. 14 (10), 4590–4602.

Chen, H., An, B., Niyato, D., Soh, Y.C., Miao, C., 2017. Workload factoring and resource sharing via joint vertical and horizontal cloud federation networks. IEEE J. Sel. Areas Commun. 35 (3), 557–570.

Chen, M., Hao, Y., 2018. Task offloading for mobile edge computing in software defined ultra-dense network. IEEE J. Sel. Areas Commun. 36 (3), 587–597.

Choudhari, T., Moh, M., Moh, T.-S., 2018. Prioritized task scheduling in fog computing. In: Proceedings of the ACMSE 2018 Conference. pp. 1–8.

Cplex, I.I., 2009. V12. 1: User's manual for CPLEX. Int. Bus. Mach. Corp. 46 (53), 157.

Elbamby, M.S., Perfecto, C., Bennis, M., Doppler, K., 2018. Toward low-latency and ultra-reliable virtual reality. IEEE Netw. 32 (2), 78–84.

Francescon, A., Baggio, G., Fedrizzi, R., Orsini, E., Riggio, R., 2017. X-MANO: An open-source platform for cross–domain management and orchestration. In: 2017 IEEE Conference on Network Softwarization. NetSoft, IEEE, pp. 1–6.

Gao, L., Moh, M., 2018. Joint computation offloading and prioritized scheduling in mobile edge computing. In: 2018 International Conference on High Performance Computing & Simulation. HPCS, IEEE, pp. 1000–1007.

Goudarzi, P., Hosseinpour, M., Ahmadi, M.R., 2021. Joint customer/provider evolutionary multi-objective utility maximization in cloud data center networks. Iran. J. Sci. Technol. Trans. Electr. Eng. 45 (2), 479–492.
Goudarzi, H., Pedram, M., 2013. Force-directed geographical load balancing and scheduling for batch jobs in distributed datacenters. In: 2013 IEEE International Conference on Cluster Computing. CLUSTER, IEEE, pp. 1–8.
Hammoud, A., Mourad, A., Otrok, H., Wahab, O.A., Harmanani, H., 2020. Cloud federation formation using genetic and evolutionary game theoretical models. Future Gener. Comput. Syst. 104, 92–104.
Hassan, M.M., Abdullah-Al-Wadud, M., Almogren, A., Song, B., Alamri, A., 2015. Energy-aware resource and revenue management in federated cloud: a game-theoretic approach. IEEE Syst. J. 11 (2), 951–961.
Hsu, C.-H., Zhang, Y., Laurenzano, M.A., Meisner, D., Wenisch, T., Mars, J., Tang, L., Dreslinski, R.G., 2015. Adrenaline: Pinpointing and reining in tail queries with quick voltage boosting. In: 2015 IEEE 21st International Symposium on High Performance Computer Architecture. HPCA, IEEE, pp. 271–282.
Johnson, D.S., Aragon, C.R., McGeoch, L.A., Schevon, C., 1989. Optimization by simulated annealing: An experimental evaluation; part i, graph partitioning. Oper. Res. 37 (6), 865–892.
Kanwal, A., Masood, R., Shibli, M.A., 2014. Evaluation and establishment of trust in cloud federation. In: Proceedings of the 8th International Conference on Ubiquitous Information Management and Communication. pp. 1–8.
Kar, B., Lin, Y.-D., Lai, Y.-C., 2020. OMNI: Omni-directional dual cost optimization of two-tier federated cloud-edge systems. In: ICC 2020-2020 IEEE International Conference on Communications. ICC, IEEE, pp. 1–7.
Kar, B., Yahya, W., Lin, Y.-D., Ali, A., 2022. A survey on offloading in federated cloud-edge-fog systems with traditional optimization and machine learning. arXiv preprint arXiv:2202.10628.
Kar, B., Yahya, W., Lin, Y.-D., Ali, A., 2023. Offloading using traditional optimization and machine learning in federated cloud-edge-fog systems: A survey. IEEE Commun. Surv. Tutor..
Kelaidonis, D., Rouskas, A., Stavroulaki, V., Demestichas, P., Vlacheas, P., 2016. A federated edge cloud-IoT architecture. In: 2016 European Conference on Networks and Communications. EuCNC, IEEE, pp. 230–234.
Kim, K.S., Kim, D.K., Chae, C.-B., Choi, S., Ko, Y.-C., Kim, J., Lim, Y.-G., Yang, M., Kim, S., Lim, B., et al., 2018. Ultrareliable and low-latency communication techniques for tactile internet services. Proc. IEEE 107 (2), 376–393.
Liaqat, M., Chang, V., Gani, A., Ab Hamid, S.H., Toseef, M., Shoaib, U., Ali, R.L., 2017. Federated cloud resource management: Review and discussion. J. Netw. Comput. Appl. 77, 87–105.
Maier, M., Chowdhury, M., Rimal, B.P., Van, D.P., 2016. The tactile internet: vision, recent progress, and open challenges. IEEE Commun. Mag. 54 (5), 138–145.
Mashayekhy, L., Nejad, M.M., Grosu, D., 2014. Cloud federations in the sky: Formation game and mechanism. IEEE Trans. Cloud Comput. 3 (1), 14–27.
Mehmeti, F., Spyropoulos, T., 2016. Performance modeling, analysis, and optimization of delayed mobile data offloading for mobile users. IEEE/ACM Trans. Netw. 25 (1), 550–564.
Nasrallah, A., Thyagaturu, A., Alharbi, Z., Wang, C., Shao, X., Reisslein, M., ElBakoury, H., 2018. Ultra-low latency (ULL) networks: A comprehensive survey covering the IEEE TSN standard and related ULL research. arXiv preprint arXiv:1803.07673.
Osseiran, A., Sachs, J., Puleri, M., et al., 2015. Manufacturing reengineered: robots, 5G and the industrial IoT. Ericsson Bus Rev. 4.
Peterson, L., Al-Shabibi, A., Anshutz, T., Baker, S., Bavier, A., Das, S., Hart, J., Palukar, G., Snow, W., 2016. Central office re-architected as a data center. IEEE Commun. Mag. 54 (10), 96–101.
Ren, J., Yu, G., He, Y., Li, G.Y., 2019. Collaborative cloud and edge computing for latency minimization. IEEE Trans. Veh. Technol. 68 (5), 5031–5044.
Samanta, A., Chang, Z., 2019. Adaptive service offloading for revenue maximization in mobile edge computing with delay-constraint. IEEE Internet Things J. 6 (2), 3864–3872.
Satyanarayanan, M., 2015. A brief history of cloud offload: A personal journey from odyssey through cyber foraging to cloudlets. GetMobile: Mob. Comput. Commun. 18 (4), 19–23.
Shi, W., Cao, J., Zhang, Q., Li, Y., Xu, L., 2016. Edge computing: Vision and challenges. IEEE Internet Things J. 3 (5), 637–646.
Shi, W., Dustdar, S., 2016. The promise of edge computing. Computer 49 (5), 78–81.
Shortle, J.F., Thompson, J.M., Gross, D., Harris, C.M., 2018. Fundamentals of queueing theory, Vol. 399. John Wiley & Sons.
Thönes, J., 2015. Microservices. IEEE Softw. 32 (1), 116.
Tong, L., Li, Y., Gao, W., 2016. A hierarchical edge cloud architecture for mobile computing. In: IEEE INFOCOM 2016-the 35th Annual IEEE International Conference on Computer Communications. IEEE, pp. 1–9.
Tozal, M.E., Wang, Y., Al-Shaer, E., Sarac, K., Thuraisingham, B., Chu, B.-T., 2013. Adaptive information coding for secure and reliable wireless telesurgery communications. Mob. Netw. Appl. 18 (5), 697–711.
Truong-Huu, T., Tham, C.-K., 2014. A novel model for competition and cooperation among cloud providers. IEEE Trans. Cloud Comput. 2 (3), 251–265.
Verma, P.K., Verma, R., Prakash, A., Agrawal, A., Naik, K., Tripathi, R., Alsabaan, M., Khalifa, T., Abdelkader, T., Abogharaf, A., 2016. Machine-to-machine (M2M) communications: A survey. J. Netw. Comput. Appl. 66, 83–105.
Villari, M., Fazio, M., Dustdar, S., Rana, O., Ranjan, R., 2016. Osmotic computing: A new paradigm for edge/cloud integration. IEEE Cloud Comput. 3 (6), 76–83.
Wang, H., Shi, P., Zhang, Y., 2017. Jointcloud: A cross-cloud cooperation architecture for integrated internet service customization. In: 2017 IEEE 37th International Conference on Distributed Computing Systems. ICDCS, IEEE, pp. 1846–1855.
Wollschlaeger, M., Sauter, T., Jasperneite, J., 2017. The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0. IEEE Ind. Electr. Mag. 11 (1), 17–27.

Binayak Kar is an Assistant Professor of computer science and information engineering at National Taiwan University of Science and Technology (NTUST), Taiwan. He received his Ph.D. degree in computer science and information engineering from the National Central University (NCU), Taiwan in 2018. He was a post-doctoral research fellow in computer science with National Chiao Tung University (NCTU), Taiwan from 2018 to 2019. His research interests include network softwarization, cloud/edge/fog computing, queueing theory, optimization, machine learning, cyber security, and quantum computing.

Ying-Dar Lin is a Chair Professor of computer science at National Yang Ming Chiao Tung University (NYCU), Taiwan. He received his Ph.D. in computer science from the University of California at Los Angeles (UCLA) in 1993. He was a visiting scholar at Cisco Systems in San Jose during 2007–2008, CEO at Telecom Technology Center, Taiwan, during 2010–2011, and Vice President of National Applied Research Labs (NARLabs), Taiwan, during 2017–2018. He cofounded L7 Networks Inc. in 2002, later acquired by D-Link Corp. He also founded and directed Network Benchmarking Lab (NBL) from 2002, which reviewed network products with real traffic and automated tools, also an approved test lab of the Open Networking Foundation (ONF), and spun off O'Prueba Inc. in 2018. His research interests include network security, wireless communications, and network softwarization. His work on multi-hop cellular was the first along this line, and has been cited over 1000 times and standardized into IEEE 802.11s, IEEE 802.15.5, IEEE 802.16j, and 3GPP LTE-Advanced. He is an IEEE Fellow (class of 2013), IEEE Distinguished Lecturer (2014–2017), ONF Research Associate (2014-2017), and received K. T. Li Breakthrough Award in 2017 and Research Excellence Award in 2017 and 2020. He has served or is serving on the editorial boards of several IEEE journals and magazines, and was the Editor-in-Chief of IEEE Communications Surveys and Tutorials (COMST). He published a textbook, Computer Networks: An Open Source Approach (www.mhhe.com/lin), with Ren-Hung Hwang and Fred Baker (McGraw-Hill, 2011).

Yuan-Cheng Lai received the Ph.D. degree from the Department of Computer and Information Science, National Chiao Tung University in 1997. He joined the faculty of the Department of Information Management, National Taiwan University of Science and Technology in August 2001 and has been a Distinguished Professor since June 2012. His research interests include performance analysis, software-defined networking, wireless networks, and IoT security.