OMNI: Omni-directional Dual Cost Optimization of Two-Tier Federated Cloud-Edge Systems

Binayak Kar[1], Ying-Dar Lin[2], Yuan-Cheng Lai[3]

[1]Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan
[2]Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan
[3]Department of Information Management, National Taiwan University of Science and Technology, Taipei, Taiwan
Email: bkar@mail.ntust.edu.tw, ydlin@cs.nctu.edu.tw, laiyc@cs.ntust.edu.tw

***Abstract*—The federation between cloud and edge has been proposed to exploit the advantages of both technologies. However, the existing studies have only considered cloud-edge computing systems which merely support vertical offloading from edges to clouds in one direction. However, there are certain cases, where the offloading needs to be done from clouds to edges and between edges. Such a cloud to edge offloading is called reverse offloading. To this end, this paper proposes a generic Omni-directional architecture of cloud-edge computing systems intending to provide vertical and horizontal offloading. To investigate the effectiveness of the proposed architecture in different operational scenarios, we formulate the dual cost optimization problem with different latency (loose, low, ultra-low) constraints. We develop an offloading algorithm using simulated annealing (SA). The experimental results show by our proposed OMNI architecture we can reduce the total cost by 15–25% and 10–20% in non-uniform and uniform inputs, respectively, compared to other existing architectures. The average latency in OMNI architecture is relatively very less compared to other architectures. It also increases utilization in the edge nodes by 5–30% in comparison to other existing architectures.**

***Index Terms*—Cloud-edge systems, cost, latency, offloading, reverse offloading, optimization.**

I. Introduction

Cloud federation is the practice of interconnecting the cloud computing environments of two or more service providers for load balancing traffic and accommodating spikes in demand. Such a federation scenario can be described variously in different papers. 1) The collection of clouds cooperates to provide resources requested by users [1]. 2) One cloud to wholesale or rent computing resources to another cloud provider [2]. 3) Federation makes cloud a user and resource provider at the same time [3] where the customer request submitted to one cloud provider is fulfilled by another. However, after the edges are re-architecture as datacenters the cloud-edge federation comes into existence. Where the users' request to the edges is severed by the clouds and vice versa.

The significance of cloud-edge federation is high because of the following reasons. 1) The edges are closer to the users by which it can reduce the latency. 2) The edges have limited resources to provide the service. To satisfy the demand an edge can federate with other edges and/or clouds. By this federation, we can increase the efficiency in resource utilization and enlargement of capabilities of two federated entities.

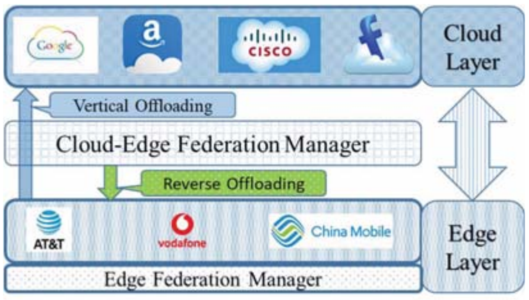


Figure 1. Federated Cloud-Edge Systems.

Figure 1 shows a federated cloud-edge system consists of two layers. The top layer is called the cloud layer consists of different clouds, such as google, amazon, etc. The bottom layer is called the edge layer having a different service provider like AT&T, Chunghwa telecom, etc. In these systems, the vertical federation between clouds and edges is managed by the cloud-edge federation manager (FM), whereas the edge-edge FM manages the horizontal federation between edges.

In this paper, we categorize the input jobs/traffics into three categories, such as ultra-low, low and loose latency jobs. 1) The jobs with latency less than 0.5ms are coming under ultra-low latency jobs and are highly time-sensitive jobs that must be handled by the edges. 2) The jobs with a latency greater than or equal to 1sec are loose latency jobs and required high storage space should be handled by the clouds. 3) The jobs whose latency is greater than 0.5ms and less than 1 sec are called low latency jobs and can be handled by either the edges or clouds.

When an edge receives any highly time-sensitive jobs and cannot handle such jobs it horizontally offloads to other edges. Similarly, when it receives loose latency jobs that consume very high storage space, it vertically offloads to a cloud node. However, when a cloud node receives any highly time-sensitive jobs, to overcome the latency and data transfer cost, it offloads to an edge node. Such downward offloading from cloud to edge is called reverse offloading [4].

Let us consider one example to explain some key terms of this paper. Consider two clouds say T1 and T2, two edges say B1 and B2 where clouds are in tier-2 and edges are in tier-1. *Federation Manager:* FM is the agent that is responsible for the federation agreement between two-party. *Horizontal*

Federation: Resource sharing agreement between two nodes in the same tier. For example, E1 and E2. *Vertical Federation:* Resource sharing agreement between two nodes in a different tier. For example, E1 and T1. *Horizontal Offloading:* When a request is for E1 is severed by E2 or when E1 offloaded its request to E2. *Vertical Offloading:* When a request is for E1 is severed by T1 or when E1 offloaded its request to T1. *Reverse Offloading:* When a vertical offloading is from an upper-tier node to a lower-tier node. For example, when a request is for T1 is severed by E1 or when T1 offloaded its request to E1. *Triangular Offloading:* When a user of one service provider (say E1) is served by another service provider (say T1) and the user's inputs are offloaded from the user to T1 via E1, *i.e.* users give input E1 and E1 offload the tasks to T1. *Non-triangular Offloading:* When a user of one service provider (say E1) is served by another service provider (say T1) and based on the federation agreement the FM will offload the user's inputs directly to T1 without offloading via E1.

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 i.e. horizontal offloading, vertical offloading from edge to cloud and vertical reverse offloading from cloud to edge based on loose, low, and ultra-low types of latencies. Our detailed contributions are as follows. (1) We design a generic two-tier architecture that enables the clouds and edges can federate with each other and offload jobs to satisfy the user's demand. (2) We propose an analytical model to minimize the cost both from the cloud and edge layer perspective with given latency constraints. (3) We use the modified simulated annealing algorithm to find the globally optimal solution of the proposed problem and compare the performance of our proposed architecture with three other existing architectures.

Related Works: Mashayekhy *et al.* proposed a model based on game theory to reshape the business structure among cloud providers [5]. In this paper, they proposed a cloud federation mechanism to maximize the profit of the cloud providers by reducing the utilization of the resources. Hassan *et al.*, in [6] present a capacity-sharing mechanism using game theory in a federated cloud environment which can lead to a global energy sustainability policy for the federation and encourages them to cooperate. The main goal of the paper is minimizing the overall energy cost by capacity sharing technique to promote the long-term individual profit of the cloud providers. The integration of vertical and horizontal cloud federation is discussed in [7]. In this integration, private clouds are known as secondary clouds are federated with each other horizontally and these horizontally federated clouds federated with the public clouds known as primary clouds vertically. Chekired *et al.*, in [8] introduce a new scheduling model for the industrial Internet of things (IIoT) data processing and proposed a two-tier cloud-fog architecture for IIoT applications by deploying multiple servers at the fog layer. In [9], a two-tier federated Edge and Vehicular-Fog architecture was presented with the objective to minimize the total cost while meeting latency constraints. Tong *et al.*, [10] proposed a hierarchical edge cloud architecture for the efficient utilization of resources by leveraging cloud computing and migrating mobile workloads for remote execution at the cloud. In [11] Cao *et al.* proposed an integrated resource provisioning model between edge infrastructure providers to realize the resource cooperation and service provisioning between them.

The rest of the paper is organized as follows. In Section II we will discuss our proposed architecture and optimization problem. We will present our solution in Section III, results in Section IV and finally, the conclusion in Section V.

II. Modeling and Problem Formulation

In this section, we will discuss our proposed two-tier cloud-edge generic architecture and proposed our cost optimization problem with latency as a constraint. The variables used to discuss our model and formulate the optimization problem presented in Table I.

TABLE I: LIST OF COMMONLY USED VARIABLES AND NOTATIONS

Notation	**Description**
$T_i, 1 \leq i \leq n,$ $B_j, 1 \leq j \leq m$	T_i: i-th node in tier-2 B_j: j-th node in tier-1
Traffic	
$\hat{\lambda}_i, \lambda_j$	Traffic input to node T_i , Traffic input to node B_j
$v_{j,i}, \hat{v}_{i,j}, h_{j,k}$	Vertical offloading from B_j to T_i , Vertical offloading from T_i to B_j, Horizontal offloading from B_j to B_k
Capacity	
$\hat{\mu}_i, \hat{\mu}_{i,i}, \hat{\mu}_{i,j}$	Computing capacity of T_i, Capacity used by T_i for self computation, Capacity of T_i used by B_j
$\bar{\bar{\mu}}_j, \bar{\bar{\mu}}_{j,j}, \bar{\bar{\mu}}^V_{j,i}, \bar{\bar{\mu}}^H_{j,k}$	Computing capacity of B_j, Capacity used by B_j for self computation, Capacity of B_j used by T_i, Capacity of B_j used by B_k
$\mu^V_{i,j}, \mu^H_{j,k}$	Communication capacity between T_i and B_j, Communication capacity between B_j and B_k
$\mu^B_{u,j}, \mu^T_{u,i}$	Communication capacity between $user$ and B_j, Communication capacity between $user$ and T_i
Cost	
$\hat{\tau}, \tau$	Total cost of tier-1 and tier-2
$\hat{c}_T, c_B$	Unit computing cost of T_i, Unit computing cost of B_j
C^M	Unit Communication cost.
$\hat{r}\ , r^V\ , r^H$	Unit computing price of T_i for B_j, Unit computing price of B_j for T_i, Unit computing price of B_j for B_k
Latency	
l^T_i, l^B_j	Computing latency at T_i and B_j
$l^{T\to B}_{i,j}, l^{B\to T}_{j,i}, l^{B\to B}_{j,k}$	Communication latency from T_i to B_j , from B_j to T_i , from B_j to B_k
$l^{u\to B}_j, l^{u\to T}_i$	Offloading latency from users to B_j, from users to T_i
L_{ul}, L_{lw}, L_{ls}	Maximum latency for *ultra-low, low and loose latency* traffic
Distance and Speed	
$d^V_{i,j}, d^H_{j,k}$	Distance between T_i and B_j, and distance between B_j and B_k
$d^{\hat{u}\to B}_{i,j}, d^{\hat{u}\to T}_{i,i},$ $d^{u\to B}_{i,j}, d^{u\to T}_{i,j}$	Shortest distance from T_i's user to B_j, from T_i's user to T_i, from B_i's user to B_j, and from B_i's user to T_j
c	Speed of light

A. Generic Two-tier Architecture and Traffic Distribution

In this section, we will discuss our proposed generic cloud-edge federated architecture. As shown in Figure 2, the top tier (tier-2) consists of cloud nodes and the bottom tier (tier-1) consists of edge nodes. The subscribers and IoT devises can send their requests to the cloud or edge nodes to avail the services. In this architecture, the edge nodes and cloud nodes are connected vertically and all edge nodes are connected horizontally. Which implies there can be a vertical offloading

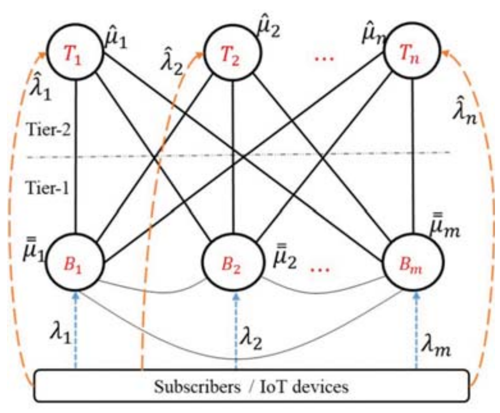


Figure 2. OMNI: A generic two-tier federated architecture.

from edges to clouds, reverse offloading from cloud to edges, and horizontal offloading from one edge node to other edge nodes. There must be a question arises, why no cloud to cloud horizontal offloading is not considered in this paper? The main reason is the clouds have no capacity limitation, no coverage limitation. Hence, based on the objective of this paper, it looks impractical to have a cloud to cloud federation.

1) Tier-1 Traffic Distribution

Assuming the probability of total input traffic to an edge node B_j is 1, which includes ultra-low, low and loose latency traffics, i.e., $p_x(\lambda_j)+p_y(\lambda_j)+p_z(\lambda_j)=1$, where $p_x(\lambda_j)$, $p_y(\lambda_j)$and $p_z(\lambda_j)$ are the probability of ultra-low, low and loose latency traffic input to cloud node B_j. The probability of offloaded ultra-low latency traffic from edge B_j to all edge nodes in the tier-1 can estimate the $p_x(\lambda_j)$ as, $p_x(\lambda_j)=[p_{x(1)}(\lambda_j)+p_{x(2)}(\lambda_j)+\cdots+p_{x(m)}(\lambda_j)]$, where m is the number of edge nodes in the tier-1, and $p_z(\lambda_j)$ can be estimated as $p_z(\lambda_j)=[p_{z(1)}(\lambda_j)+p_{z(2)}(\lambda_j)+\cdots+p_{z(n)}(\lambda_j)]$, where n is the number of cloud nodes in the tier-2. Since the low latency traffics of B_j can be offloaded both horizontally to edge nodes and vertically to cloud nodes, then $p_y(\lambda_j)$ can be calculated as $p_y^V(\lambda_j)+p_y^H(\lambda_j)$, where $p_y^V(\lambda_j)$ is the probability of vertically offloaded traffic and $p_y^H(\lambda_j)$ is the probability of horizontally offloaded traffic. $p_y^V(\lambda_j)$ and $p_y^H(\lambda_j)$ can be estimated as, $p_y^V(\lambda_j)=[p_{y(1)}^V(\lambda_j)+p_{y(2)}^V(\lambda_j)+\cdots+p_{y(m)}^V(\lambda_j)]$ and $p_y^H(\lambda_j)=[p_{y(1)}^H(\lambda_j)+p_{y(2)}^H(\lambda_j)+\cdots+p_{y(n)}^H(\lambda_j)]$. Then the vertically offloaded total traffic from edge B_j to cloud node T_i will be, $v_{j,i}=p_{z(i)}(\lambda_j)*\lambda_j+p_{y(i)}^V(\lambda_j)*\lambda_j$. Total horizontally offloaded traffic from edge B_j to edge B_k will be, $h_{j,k}=p_{x(k)}(\lambda_j)*\lambda_j+p_{y(k)}^H(\lambda_j)*\lambda_j$. In $h_{j,k}$, if $j=k$ then there will be no horizontal offloading.

2) Tier-2 Traffic Distribution:

Assuming the probability of total input traffic to a cloud node T_i is 1, which include 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 $P_z(\hat{\lambda}_i), P_x(\hat{\lambda}_i)$ and $P_y(\hat{\lambda}_i)$ are the probability of loose latency, ultra-low and low latency traffics input to cloud node T_i, respectively. As these ultra-low latency traffics are highly time-sensitive, hence they will reverse offloaded to the edges in tier-1. $P_{x(j)}(\hat{\lambda}_i)$ represents probability of offloaded ultra-low latency traffic from cloud T_i to edge B_j. Then total probability of reverse offloaded ultra-low latency traffic from cloud node T_i to tier-1 will be, $P_x(\hat{\lambda}_i)=[P_{x(1)}(\hat{\lambda}_i)+P_{x(2)}(\hat{\lambda}_i)+\cdots+P_{x(m)}(\hat{\lambda}_i)]$. $P_{y(j)}(\hat{\lambda}_i)$ represents probability of offloaded low latency traffic from cloud T_i to edge B_j and $P_{y(0)}(\hat{\lambda}_i)$ is the probability of same traffic type for T_i for self-computation. Then the probability of total low latency traffic input to a cloud node T_i will be, $P_y(\hat{\lambda}_i)=P_{y(0)}(\hat{\lambda}_i)+[P_{y(1)}(\hat{\lambda}_i)+P_{y(2)}(\hat{\lambda}_i)+\cdots+P_{y(m)}(\hat{\lambda}_i)]$. Then the reverse offloaded traffic from the cloud T_i to the edge B_j can be estimated as, $\hat{v}_{i,j}=P_{x(j)}(\hat{\lambda}_i)*\hat{\lambda}_i+P_{y(j)}(\hat{\lambda}_i)*\hat{\lambda}_i$.

B. Latency Estimation

In our model, we assume both the cloud nodes and edge nodes are consisting of multiple servers having equal capacity. However, the number of servers in cloud nodes is relatively high compared to edge nodes so as the total computational capacity. To calculate the computation latency of the nodes, we can apply the $M/M/k$ queueing model where k is the number of servers in the node. As the capacities of the servers are equal, to simplify, our estimation, and apply the $M/M/1$ model instead multi-server to calculate the latency. Here, in this single server model, the capacity of the single server is the sum of the total capacities of the servers in that node. The communication latency is calculated by the $M/M/1$ queueing model.

1) Computational Latency Estimation

Based on $M/M/1$ model, the computational latency of cloud node T_i will be, $l_i^T=\frac{1}{\hat{\mu}_i-[\hat{\lambda}_i-\sum_{j=1}^m\hat{v}_{i,j}+\sum_{j=1}^m v_{j,i}]}$, where $\hat{\lambda}_i-\sum_{j=1}^m\hat{v}_{i,j}+\sum_{j=1}^m v_{j,i}<\hat{\mu}_i$. Similarly, for an edge node the computation latency will be, $l_j^B=\frac{1}{\bar{\bar{\mu}}_j-[\lambda_j-\sum_{i=1}^n v_{j,i}+\sum_{i=1}^n\hat{v}_{i,j}-\sum_{k=1}^m h_{j,k}+\sum_{k=1}^m h_{k,j}]}$, where $\lambda_j-\sum_{i=1}^n v_{j,i}+\sum_{i=1}^n\hat{v}_{i,j}-\sum_{k=1}^m h_{j,k}+\sum_{k=1}^m h_{k,j}<\bar{\bar{\mu}}_j$.

2) Triangular Communication Latency Estimation:

From the users to cloud/edge initial communication latency can be estimated as follows. The communication latency from user to B_j can be estimated as, $l_j^{u\to B}=\frac{1}{\mu_{u,j}^B-\lambda_j}+\frac{d_{j,j}^{u\to B}}{c}$, where, $\lambda_j\le\mu_{u,j}^B$. The communication latency from user to T_i can be estimated as, $l_i^{u\to T}=\frac{1}{\mu_{u,i}^T-\hat{\lambda}_i}+\frac{d_{i,i}^{\hat{u}\to T}}{c}$, where, $\hat{\lambda}_i\le\mu_{u,i}^T$.

Triangular communication is the case, where users give their input to the node A_0. Node A_0 determines, whether, the input is appropriate based on its available capacity, capability and other constraints. If yes, then the input is handled by the node itself. Otherwise, node A_0 will determine another node A_1, who has a federation agreement with A_0 and can handle the input. Then A_0 will offload the traffic to A_1. This communication from user to A_0 and then $A_1$is called triangular communication. This federation takes place in the control plane level. The communication latencies are estimated as follows. In reverse offloading, the communication latency from node T_i to B_j is $l_{i,j}^{T\to B}=\frac{1}{\mu_{i,j}^V-\hat{v}_{i,j}}+\frac{D_{i,j}^V}{c}$, where $\hat{v}_{i,j}<\mu_{i,j}^V$. In vertical offloading, the communication latency from node B_j to T_i is $l_{i,j}^{B\to T}=\frac{1}{\mu_{ij}^V-v_{i,j}}+\frac{D_{i,j}^V}{c}$, where $v_{i,j}<\mu_{i,j}^V$, In horizontal offloading, the communication latency from node B_j to B_k is estimated as, $l_{j,k}^{B\to B} = \frac{1}{\mu_{j,k}^{H} - h_{j,k}} + \frac{D_{j,k}^{V}}{c}$, where $h_{j,k} < \mu_{j,k}^{H}$.

C. Problem Formulation with Triangular Offloading

The federation decision between two nodes is taken in the control plane. Whether to offload the traffic to another node is purely depend on the available resource capacity and capability of the nodes. In this section, for such triangular offloading, we present the optimization problem from the control plane perspective.

1) Objective of the Tier-1:

$$\begin{cases} x_1 = \sum_{j=1}^{m} \lambda_j * d^{u,B} * C^M \\ x_2 = \sum_{j=1}^{m} \bar{\bar{\mu}}_{j,j} * c_B \\ x_3 = \sum_{j=1}^{m} \sum_{k=1}^{m} h_{j,k} * d_{j,k}^{H} * C^M \\ x_4 = \sum_{k=1}^{m} \sum_{j=1}^{m} \bar{\bar{\mu}}_{k,j}^{H} * r^H \\ x_5 = \sum_{j=1}^{m} \sum_{i=1}^{n} v_{j,i} * d_{i,j}^{V} * C^M \\ x_6 = \sum_{i=1}^{n} \sum_{j=1}^{m} \hat{\mu}_{i,j} * \hat{r} \end{cases} \tag{1}$$

In equation (1), let x_1, x_2, x_3, x_4, x_5, and x_6 are the variables presenting different costs of tier-1 nodes. Where x_1 shows the communication cost from the users to the 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 price of edges while computation is done by other edge nodes. The total communication cost from edge nodes to the cloud nodes is presented by x_5. The x_6 presents the total computing price of edges while computation is done by cloud nodes. Then the objective function for the edge nodes is to *minimize* (τ), where,

$$\tau = x_1 + x_2 + x_3 + x_4 + x_5 + x_6, \tag{2}$$

subject to

$$\begin{cases} l_j^{u\to B} + l_j^{B} < L_{ul}, & (3a) \\ l_j^{u\to B} + l_{j,k}^{B\to B} + l_k^{B} < L_{ul}, & (3b) \end{cases}$$

$$\begin{cases} l_j^{u\to B} + l_j^{B} < L_{lw}, & (4a) \\ l_j^{u\to B} + l_{j,k}^{B\to B} + l_k^{B} < L_{lw}, & (4b) \\ l_j^{u\to B} + l_{j,i}^{B\to T} + l_i^{T} < L_{lw}, & (4c) \end{cases}$$

$$l_j^{u\to B} + l_{j,i}^{B\to T} + l_i^{T} < L_{ls}, \tag{5}$$

$$0 < \bar{\bar{\mu}}_{j,j} < \bar{\bar{\mu}}_j, \tag{6}$$

$$\bar{\bar{\mu}}_{j,j} + \sum_{j=1}^{m} \bar{\bar{\mu}}_{j,k}^{H} + \sum_{i=1}^{n} \bar{\bar{\mu}}_{ji}^{V} \le \lambda_j. \tag{7}$$

The equations (3) - (5) presents the latency constraints for the inputs given to all edge node from users, where equations in (3) are for ultra-low latency, equations in (4) are for low latency, and equation (5) is for loose latency traffics. The inequality in Equation (3a) shows the user to B_j communication latency plus the computing latency of B_j must be less than the ultra-low latency limit. Equation (3b) is the case where the user submitted the job to B_j and B_j offloaded the job to B_k. The inequality in Equation (3b) shows the sum of the user to B_j communication latency, B_j to B_k communication latency and the computing latency of B_k must be less than the ultra-low latency limit. The inequality in Equation (4a) shows the user to B_j communication latency plus the computing latency of B_j must be less than the low latency limit. Equation (4b) is the case where user submitted the job to B_j and B_j offloaded the job to B_k. The inequality in Equation (4b) shows the sum of the user to B_j communication latency, B_j to B_k communication latency and the computing latency of B_k must be less than the low latency limit. Equation (4c) is the case where the user submitted the job to B_j and B_j offloaded the job to T_i. The inequality in Equation (4c) shows the sum of the user to B_j communication latency, B_j to T_i communication latency and the computing latency of T_i must be less than the low latency limit. Equation (5) is the case where the user submitted the job to B_j and B_j offloaded the job to T_i. The inequality in Equation 5 shows the sum of the user to B_j communication latency, B_j to T_i communication latency and the computing latency of T_i must be less than the loose latency limit. The constraint in equation (6) presents no edge node can compute all its received requests by itself or 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 edge nodes is presented in equation (7).

2) Objective of the Tier-2:

$$\begin{cases} y_1 = \sum_{i=1}^{n} \hat{\lambda}_i * d^{u,T} * C^M \\ y_2 = \sum_{i=1}^{n} \hat{\mu}_{i,i} * \hat{c}_T \\ y_3 = \sum_{i=1}^{n} \sum_{j=1}^{m} \hat{v}_{i,j} * d_{i,j}^{V} * C^M \\ y_4 = \sum_{j=1}^{m} \sum_{i=1}^{n} \mu_{j,i} * r^V \end{cases} \tag{8}$$

In equation (8), let y_1, y_2, y_3, and y_4 are the variables presenting different costs of tier-2 cloud nodes. 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 clouds to the edges is presented by y_3. The y_4 presents the total computing price of clouds while computation is done by edge nodes. Then the objective function for the edge nodes is to *minimize* $(\hat{\tau})$, where,

$$\hat{\tau} = y_1 + y_2 + y_3 + y_4, \tag{9}$$

subject to

$$l_i^{u\to T} + l_{i,j}^{T\to B} + l_j^{B} < L_{ul}, \tag{10}$$

$$\begin{cases} l_i^{u\to T} + l_i^{T} < L_{lw}, & (11a) \\ l_i^{u\to T} + l_{i,j}^{T\to B} + l_j^{B} < L_{lw}, & (11b) \end{cases}$$

$$l_i^{u\to T} + l_i^{T} < L_{ls}, \tag{12}$$

$$\begin{cases} 0 < \hat{\mu}_{i,i} < \hat{\mu}_i \\ P_z(\hat{\lambda}_i) < 1 \end{cases}, \tag{13}$$

$$\hat{\mu}_{i,i} + \sum_{j=1}^{m} \hat{\mu}_{i,j} \le \hat{\lambda}_i. \tag{14}$$

The equations (10) – (12) presents the latency constraints for the inputs given to all cloud nodes from the users, where equation (10) is for ultra-low latency, equations in (11) are for low latency, and equation (12) is for loose latency traffics. Equation (10) is the case where the user submitted the job to T_i and T_i offloaded the job to B_j. The inequality in Equation (10) shows the sum of the user to T_i communication latency, T_i to B_j communication latency and the computing latency of B_j must be less than the ultra-low latency limit. The inequality in Equation (11a) shows the user to T_i communication latency plus the computing latency of T_i must be less than the low latency limit. Equation (11b) is the case where the user submitted the job to T_i and T_i offloaded the job to B_j. The inequality in Equation (11b) shows the sum of the user to T_i communication cost, T_i to B_j communication latency and the computing latency of B_j must be less than the low latency limit. The inequality in

Equation (12) shows the sum of the user to T_i communication latency and the computing latency of T_i must be less than the loose latency limit. The constraint in equation (13) presents no cloud node can compute all its request by itself or offload them entirely to others provided all the traffics to the cloud node are not loose latency traffics. The sum of self-computation and vertical offloading capacity must be less than the total input to the clouds is discussed in equation (14).

D. Problem Formulation without Triangular Offloading

As discussed in Section I, in practice the offloading decision between two party in the federated systems is taken by the FM in the data plane based on the federation agreement between the two party. In this subsection, for such non-triangular offloading, we modify our previously proposed optimization problem from control plane perspective to data plane perspective.

1) Non-Triangular Latency Estimation:

When offloading decision takes place in data plane the communication from the user to service provider become non-triangular. The new offloading latency can be estimated as follows. The vertical offloading latency from the user to cloud node T_i, where the job input is for B_j, is estimated as, $l_{j,i}^{u\to BT} = \frac{1}{\mu_{u,i}^{T}-\lambda_j} + \frac{d_{j,i}^{u\to T}}{c}$, where, $\lambda_j \le \mu_{u,i}^{T}$. The horizontal offloading latency from the user to edge B_k, where the job input is for B_j, is calculated as, $l_{j,k}^{u\to BB} = \frac{1}{\mu_{u,k}^{B}-\lambda_j} + \frac{d_{j,k}^{u\to B}}{c}$, where, $\lambda_j \le \mu_{u,k}^{B}$. The reverse offloading latency from user to edge B_j, where job input is for T_i will be, $l_{ij}^{u\to TB} = \frac{1}{\mu_{u,j}^{B}-\hat{v}_{i,j}} + \frac{d_{i,j}^{\hat{u}\to B}}{c}$, where, $\hat{v}_{i,j} \le \mu_{u,j}^{B}$.

2) Modified objective of the tier-1:

$$\begin{cases} x_7 = \sum_{j=1}^{m}\sum_{k=1}^{m} h_{j,k} * d_{j,k}^{u\to B} * C^M \\ x_8 = \sum_{j=1}^{m}\sum_{i=1}^{n} v_{j,i} * d_{j,i}^{u\to B} * C^M \end{cases} \tag{15}$$

In Equation (15), x_7 is the case where the input traffic from the user is for edge B_j, however, due to the decision of edge-edge FM the traffic is offloaded directly to the edge node B_k. The x_7 represents the total horizontal offloading cost between the edges. And in x_4 (from equation (1)), when $j = k$ in the horizontal offloading capacity $\bar{\bar{\mu}}_{k,j}^{H}$ and the unit computing price $r_{j,j}^{H} = c_j$, then there will be no extra self-computing cost. The x_8 is the case where the input traffic from the user is for edge B_j, however, due to the decision of cloud-edge FM the traffic is offloaded directly to the cloud T_i. Then the modified objective function for the edge nodes will be to *minimize* (τ_1), where,

$$\tau_1 = x_4 + x_6 + x_7 + x_8, \tag{16}$$

subject to

$$l_{j,k}^{u\to BB} + l_k^B < L_{ul}, \tag{17}$$

$$\begin{cases} l_{j,k}^{u\to BB} + l_k^B < L_{lw}, & (18a) \\ l_{j,i}^{u\to BT} + l_i^T < L_{lw}, & (18b) \end{cases}$$

$$l_{j,i}^{u\to BT} + l_i^T < L_{ls}, \tag{19}$$

(3a), (4a), (6), and (7).

Equation (17) and (18a) are the cases where jobs from user for edge node B_j are directly offloaded to B_k based on edge-edge FM's decision. The inequality in Equation (17) presents the sum of the user to B_k communication latency, and the computing latency of B_k must be less than the ultra-low latency limit, and for equation (18a) the same inequality must be less than the low latency limit. Equation (18b) and (19) are the cases where jobs from the user for edge B_j are directly offloaded to cloud T_i based on cloud-edge FM's decision. The inequality in Equation (18b) presents the sum of the user to T_i communication latency, and the computing latency of T_i must be less than the low latency limit, and for equation (19) the same inequality must be less than the loose latency limit.

3) Modified objective of the tier-2:

$$\begin{cases} y_5 = \sum_{i=1}^{n}\sum_{j=1}^{m} \hat{v}_{i,j} * d_{i,j}^{\hat{u}\to B} * C^M \\ y_6 = \sum_{i=1}^{n}(\hat{\lambda}_i - \sum_{j=1}^{m} \hat{v}_{i,j}) * d_{i,i}^{\hat{u}\to T} * C^M \end{cases} \tag{20}$$

In Equation (20), y_5 is the case where the input traffic from the user is for cloud node T_i, however, based on the decision of cloud-edge federation manager the traffic is offloaded directly to the edge node B_j. The y_5 represents the total reverse offloading cost from cloud to edge nodes. The communication cost from user to cloud nodes where the input traffic is for the cloud nodes is y_6. Then the new objective function of the cloud nodes will be *minimize* $(\hat{\tau}_1)$, where

$$\hat{\tau}_1 = y_2 + y_4 + y_5 + y_6, \tag{21}$$

subject to

$$l_{i,j}^{u\to TB} + l_j^B < L_{ul}, \tag{22}$$

$$l_{i,j}^{u\to TB} + l_j^B < L_{lw}, \tag{23}$$

(11a), (12), (13), and (14).

The Equation (22) and (23) are the cases where jobs from user for cloud T_i are reverse offloaded to edge B_j based on cloud-edge FM's decision. The inequality in Equation (22) presents the sum of the user to B_j communication latency, and the computing latency of B_j must be less than the low latency limit, and for equation (23) the same inequality must be less than the loose latency limit.

III. Solutions

In the solution, we describe the Simulated Annealing (SA) algorithm [12], which performs a probabilistic technique to find a globally optimal solution. Specifically, it is a metaheuristic to approximate global optimization in a large search space for an optimization problem and is used when the search space is discrete.

A. Two-tier Simulated Annealing

The two-tier simulated annealing process is a modified version of SA is 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 Equation (16) and $Cost_2()$, *i.e.* in Equation (21). The differences between the old and new costs of tier-1 and tier-2 denoted as Δ_1 and Δ_2, respectively, are computed. 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 $\exp\left(-\frac{\Delta_2}{T}\right)$ 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 can accept x_{new} with probability $\exp\left(-\frac{\Delta_1}{T}\right)$. x_{new} will acceptable with probability

$\exp\left(-\frac{(\Delta_1+\Delta_2)}{T}\right)$ 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. Consequently, the SA process has no bounded time complexity. Instead, its time complexity is determined by the cooling parameter α.

Algorithm 1: Two-tier Simulated Annealing Algorithm

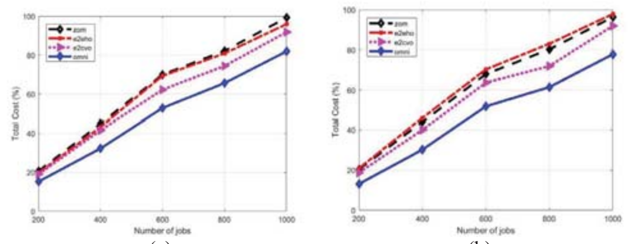
```
1: Randomly generate initial solutions $x_{old}$
2: $T \leftarrow T_{ini}$
3: **While** $T > T_{ter}$ **do**
4:     Generate new solutions $x_{new}$
5:     $\Delta_1 = Cost_1(x_{new}) - Cost_1(x_{old})$
       $\Delta_2 = Cost_2(x_{new}) - Cost_2(x_{old})$
6:     **if** ($\Delta_1 \leq 0$) & ($\Delta_2 \leq 0$) **then**
           accept $x_{old} \leftarrow x_{new}$
7:     **else if** ($\Delta_1 \leq 0$) & ($\Delta_2 > 0$) **then**
           accept $x_{old} \leftarrow x_{new}$ with probability $e^{-\frac{\Delta_2}{T}}$
8:         **else if** ($\Delta_1 > 0$) & ($\Delta_2 \leq 0$) **then**
               accept $x_{old} \leftarrow x_{new}$ with probability $e^{-\frac{\Delta_1}{T}}$
9:             **else if** ($\Delta_1 > 0$) & ($\Delta_2 > 0$) **then**
                   accept $x_{old} \leftarrow x_{new}$ with probability $e^{-\frac{(\Delta_1+\Delta_2)}{T}}$
               **end**
10:    $T \leftarrow \alpha \cdot T$
11: **end**
```

In this algorithm, for the initial solution, we apply randomness with some restrictions. As discussed in our proposed architecture our input jobs are three different types and ultra-low latency jobs are handled by only edges, loose latency jobs are handled by only clouds and low latency jobs can be handled by both edges and clouds. For a cloud node, when the inputs are loose latency jobs it will have kept by the cloud, whereas if the job is ultra-low latency job it will offload to any edge randomly. For low latency jobs, the cloud will have kept 70 percent by itself and offload rest to the edges randomly. For an edge node, if the input is a loose latency job, then it will offload to the clouds randomly. If the inputs are low latency jobs than 50 percent of jobs are kept the edge and the other 50 percent will be offloaded to other edges and clouds. If the inputs are ultra-low latency jobs than 70 percent will be kept by the edge and the other 30 percent will be randomly offloaded to other edges.

IV. Results

A. Experiment Setup

For the experiment, we have considered a cloud-edge federated network consist of five clouds and five edges and are geographically distributed. The capacity of an edge is 2GB consist of 4 VMs each having capacity 500MB. However, the clouds have unlimited capacity but each VM is of 500MB. The distance between the user to edge, user to cloud, edge to edge and edge to cloud are 1~100, 500~1000, 100~200 and 500~1000 KMs, respectively. The size of the input jobs is between 1KB to 500MB.



(a) (b)

Figure 3. Percentage of total cost of the edge-cloud systems per number of jobs where job inputs are uniform and non-uniform.

B. Performance Analysis

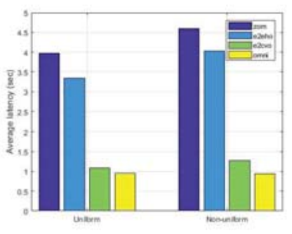
In this section, we compared the performance with three other architectures: 1) zero offloading model (*zom*), 2) edge-to-cloud vertical offloading (*e2cvo*) [10], and 3) edge-to-edge horizontal offloading (*e2eho*) [11]. In the experiment, we have considered both uniform and non-uniform inputs. In the case of uniform inputs, we gave the equal number of job inputs to each node irrespective of the size of the jobs. While in the non-uniform case, the number of inputs to the nodes are random irrespective of the job size.

1) Total Cost Analysis

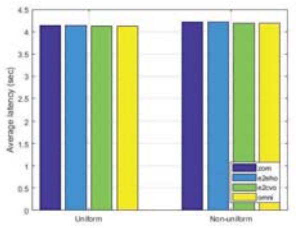
Fig. 3 shows with the increase in the number of input jobs the total cost of all models gradually increases. Fig. 3a shows the system performance with uniform inputs whereas Fig. 3b shows for non-uniform inputs. In the non-uniform case, our *Omni* architecture saves the total cost by 15-25% compared to all other architectures because of their horizontal, vertical and reverse offloading mechanisms and 10-20% in case of uniform inputs. Due to the reverse offloading, the highly time-sensitive jobs of the clouds are redirected to the edges which reduce the communication cost and due to horizontal federation, the edge extends their computing capacity in the edge layer. The *e2cvo* saves more cost compared to *zom* and *e2eho* as the edge nodes offload the loose latency and some low latency jobs to the clouds which reduces the storage burden on the edge nodes and makes the computation faster as the cloud has unlimited computing resources. If we compare the *zom* and *e2eho* performance, with uniform input the both save a nearly equal amount of costs, however, with non-uniform inputs, the *e2eho* saves more cost compared to *zom* due to its edge to edge horizontal federation as it extends the computing capacity and provides the service faster than *zom*.

2) Tier-1 and Tier-2 Latency Analysis

Fig. 4 shows the average latency for uniform and non-uniform job inputs of tier-1 and tier-2 nodes respectively. Fig. 4a shows the average latency of non-uniform input is relatively higher than uniform input both in federated (*e2eho, e2cvo, Omni*) and non-federated (*zom*) architectures. Because, in case of non-uniform inputs, in non-federated architecture, high input nodes will take more time to finish the computation due to limited computing resources whereas federated architectures take less time by offloading from high input nodes to low input nodes. Also in both cases, the latency of federated architectures has less compared to *zom*, because in federated architecture the nodes can virtually extend their capacity by the high input nodes offloading jobs to low input nodes. However, in *zom* all the input jobs of each node are handled by themselves. And due

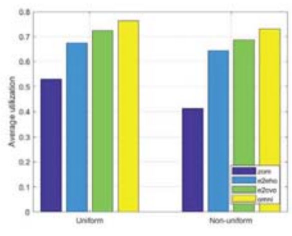


(a)

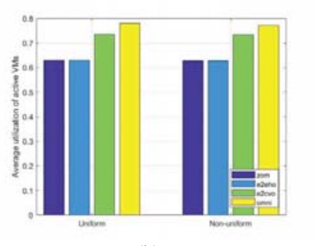


(b)

Figure 4. Average latency of tier-1 and tier-2 nodes both uniform and non-uniform inputs.



(a)



(b)

Figure 5. Utilization of active VMs in edge and cloud layers' nodes.

to the edge nodes' capacity limitation, the latency goes up. Fig. 4a shows the latency of *Omni* is less than *e2cvo* due to its reverse offloading architecture. It offloads the highly time-sensitive jobs to the edges which reduce the communication time. Also due to *Omni*'s horizontal federation, it extends its computation capacity in the edge layer as a result, it able to handle more jobs in time and reduce the latency compared to *e2cvo*. Again Fig. 4a also shows both *Omni* and *e2cvo* have relatively less latency compared to *e2eho* and *zom*. This is because of the vertical offloading in *Omni* and e2cho, where loose latency or high storage jobs are offloaded to the cloud where no unlimited computing capacities are available. The average latency of all the models is relatively same for both uniform and non-uniform inputs as shown in Fig. 4b. This is because the clouds have unlimited computing resources and storage capacity, as soon as the input jobs are given to the clouds required resources get allocated to the jobs for computation.

3) Utilization of Active VMs in Edge and Cloud Layer

The utilization of the VM depends on its *active/inactive* state. If it is *inactive*, we will not consider the VM to estimate the utilization. If it is in *active* state, then depending on the size of the allocated jobs the utilization is estimated. For example, if there is no job in the VM and still it is active, the utilization is zero. And if the allocated jobs size is 100MB, then the utilization of the VM is 20%. Fig. 5 shows the average utilization of the active VMs in edge and cloud layers, respectively, both for uniform and non-uniform job inputs. As discussed before the heavy loaded edge nodes offload their excessive jobs to either to cloud nodes vertically or edge nodes horizontally. Similarly, the cloud node reverse offloads the highly time-sensitive jobs to the edges. By the simulated annealing algorithm, the *Omni* architecture tries to accommodate more jobs within the specified amount of resources to maximize resource utilization both in edge and cloud nodes. As a result, the utilization of *Omni* architecture is relatively better compared to other architectures both in edge and cloud layer nodes for uniform and non-uniform job inputs. The result in Fig. 5a also shows due to the federation, utilization of the resources in federated architectures are better than *zom*. In *zom*, the utilization for uniform input is relatively better compared to non-uniform input due to the load balance among the nodes. Fig. 5b shows the utilization of cloud nodes. Here the utilization of *zom* and *e2eho* is similar as they cannot offload to other nodes. And the e2cvo gets some excessive jobs from the edges which helps to reduce the internal fragmentation in the VMs and increases the utilization.

V. Conclusions

In this paper, we have proposed an Omni-directional two-tier federated cloud-edge architecture where 1) an edge can horizontally federate with other edges, 2) cloud and edge can vertically federate with each other. In the vertical federation not only edge can offload the request to the cloud nodes due to limited capacity but also the cloud can reverse offload its highly time-sensitive request to edges for faster service. We proposed a dual cost optimization problem to minimize the cost of both cloud and edge layer with given latency constraint, and apply a modified SA algorithm to find the global optimum results. The results show, our proposed *Omni* architecture reduces the total cost by 15–25% in non-uniform inputs and 10–20% in uniform inputs, compared to other existing architectures. It also increases utilization in the edge nodes and the average latency in our architecture is relatively less compared to other existing architectures.

Acknowledgements

This work was supported in part by H2020 EU-TW 5G-Coral project, and also in part by Ministry of Science and Technology of Taiwan.

References

[1] B. Rochwerger, *et al*., "Reservoir-when one cloud is not enough," *Computer*, vol. 44, no. 3, pp. 44–51, Mar. 2011

[2] L. Vaquero, *et al*., "A break in the clouds: Towards a cloud definition," *ACM SIGCOMM Comput. Commun*. Rev., vol. 39, no. 1, pp. 50–55, 2008.

[3] Tusa, Francesco, *et al*., "How CLEVER-based clouds conceive horizontal and vertical federations," *Symposium on Computers and Communications (ISCC), IEEE*, pp. 167–172, 2011.

[4] M. Villari, *et al*., "Osmotic computing: A new paradigm for edge/cloud integration." *Cloud Computing, IEEE,* vol. 3, no. 6, pp. 76–83, 2016.

[5] Lena Mashayekhy, *et al*., "Cloud federations in the sky: Formation game and mechanism,*" IEEE Transactions on Cloud Computing,* vol. 3, no. 1, pp. 14–27, 2015.

[6] M. M. Hassan, *et al*., "Energy-aware resource and revenue management in federated cloud: a game-theoretic approach," *IEEE Systems Journal,* vol. 11, no. 2, pp. 951–961, 2017.

[7] H. Chen, *et al*., "Workload factoring and resource sharing via joint vertical and horizontal cloud federation networks," *IEEE Journal on Selected Areas in Communications,* vol. 35, no. 3, pp. 557–570, 2017.

[8] D. A. Chekired, *et al*., "Industrial IoT data scheduling based on hierarchical fog computing: a key for enabling smart factory," *IEEE Trans. on Industrial Informatics,* vol. 14, no. 10, pp. 4590–4602, 2018.

[9] Y. D. Lin, *et al*., "Cost Minimization with Offloading to Vehicles in Two-tier Federated Edge and Vehicular-Fog Systems," *IEEE 90th Vehicular Technology Conference (VTC2019-Fall),* pp. 1–6, Honolulu, Hawaii, Sep., 2019.

[10] L. Tong, *et al*., "A hierarchical edge cloud architecture for mobile computing," *International Conference on Computer Communications*, *IEEE*, pp. 1–9, 2016.

[11] Xiaofeng Cao, *et al*. "Edge Federation: Towards an Integrated Service Provisioning Model," *arXiv preprint arXiv: 1902.09055,* 2019.

[12] D. S. Johnson, *et al*. "Optimization by simulated annealing: An experimental evaluation; part I, graph partitioning," *Operations research,* vol. 37, no. 6, pp. 865–892, 1989.