Computer Networks 176 (2020) 107286

Contents lists available at ScienceDirect

Computer Networks

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

Revenue sharing in edge-cloud systems: A Game-theoretic perspective

Zhi Cao [a], Honggang Zhang [b,*], Benyuan Liu [c], Bo Sheng [a]

[a] Computer Science Department, Umass Boston, Boston, MA, 02125 United States
[b] Engineering Department, Umass Boston, Boston, MA 02125 United States
[c] Department of Computer Science, Umass Lowell, Lowell, MA, 01854 United States

ARTICLE INFO

Keywords:
Edge computing
Revenue sharing mechanisms
Shapley value
Nash equilibrium

ABSTRACT

We explore the design of revenue sharing mechanisms for an Edge-Cloud computing system from a game-theoretic perspective. Different from traditional cloud computing, the providers in an Edge-Cloud system are independent and self-interested. The system adopts a task distribution mechanism to maximize the total revenue received from clients and employs a revenue sharing mechanism to split the received revenue among service providers. Under system-level mechanisms, service providers game with the system in order to maximize their own utilities by strategically allocating their resources. We introduce a game-theoretic framework to model the competition among the service providers as a non-cooperative game. We show the existence of Nash equilibrium in the game theoretically and experimentally. We find that at Nash equilibria, the revenue sharing design based directly on actual contributions of service providers results in significantly worse system-level performance than revenue sharing mechanisms based on marginal contributions. We find that the reason for this seemingly counter-intuitive result is that revenue sharing mechanisms based on marginal contributions discourage providers with less powerful resources from contributing resources to the system at equilibrium state. Our framework offers an economics approach to the understanding of Edge-Cloud systems and provides fundamental insights into their revenue sharing design.

1. Introduction

Edge computing [1–4] is a promising computing paradigm that will meet the service requirements of the latency-sensitive and/or bandwidth-hungry applications brought by the rapid growing Internet of Things (IoT) and Artificial Intelligence (AI) systems. In this paper we investigate Edge-Cloud system, a type of edge computing system where computation service providers at the edge of the Internet (referred to as edge service providers or *edge providers*, which are close to IoT sensors, mobile devices, and end users) and the providers at the cloud (referred to as cloud service providers or *cloud providers*) collectively provide computing services to the clients at the edge. A service provider is also referred to as a resource provider in the paper as we focus on the computation servers offered by the provider.

Different from a traditional cloud computing environment in which all servers are organized in data centers and tightly controlled and managed by a provider, the various service providers in an Edge-Cloud system are independent and located at various distances away from clients, and they make independent decisions on the computation resources that they provide to the system. In order to achieve a high system-level efficiency, an Edge-Cloud system adopts a task distribution mechanism to maximize the total revenue received from clients, and it employs a revenue sharing mechanism to fairly split its received revenue among service providers.

Under system-level mechanisms, the providers of an Edge-Cloud system are likely to compete with each other and game with the system in order to maximize their own utilities by strategically adjusting the resources that they offer to the system. A provider may increase or decrease the amount of computation servers placed in the system, in order to increase its utility (e.g., a profit calculated as received value minus operation cost) under the current system-level mechanisms that allocate computation jobs (of clients or edge users) to servers and distribute the total values received (from edge users) to the providers that own the servers. For example, with complete information of the whole system (including all other providers' servers, job arrival information, and the details of system-level mechanisms), a provider P1 can find its optimal number of servers to be placed into the system in order to maximize its profit. Since obtaining complete information is nearly impossible in practice, P1 can find an estimated optimal number of servers that it should place in the system through learning from history data. Note that the optimal number of servers of P1 is a function of other providers' current numbers of servers placed in the system and other system state information. Once P1 deploys its current optimal number of servers in the system, another provider P2 may change its number

* Corresponding author.
E-mail addresses: Zhi.Cao001@umb.edu (Z. Cao), Honggang.Zhang@umb.edu (H. Zhang), bliu@cs.uml.edu (B. Liu), Bo.Sheng@umb.edu (B. Sheng).

https://doi.org/10.1016/j.comnet.2020.107286
Received 14 August 2019; Received in revised form 6 April 2020; Accepted 27 April 2020
Available online 4 May 2020

of servers in order to maximize its own profit as the numbers of P1's and other providers' servers may make P2's current number of servers no longer optimal. Once P2 changes its number of servers, P1 may need to find a new optimal solution based on the updated number of servers of P2. The competition between P1 and P2 is shown in the process of their repeatedly adjusting their own optimal numbers of servers respectively. A similar and more complicated competition process can happen in a system where there may be more than two providers that attempt to maximize their utilities by updating their number of servers, and the updating of different providers may happen asynchronously and in a distributed fashion. Furthermore, since providers keep changing their numbers of servers offered to the system as a result of their optimization processes aforementioned, the system needs to constantly adjust its job distribution and revenue sharing/splitting in order to maintain a high overall system-level performance. Therefore the pursuit of maximal profit or utility by the providers may lead to a competition among themselves. Furthermore, the self-interested behaviors of the providers in the system might result in an inefficient system state with low overall system performance, as their individual self-interested objectives (e.g., a provider tries to maximize its own utility) do not collectively align with system-wide objectives (e.g., to maximize the overall system utility). Therefore, it is important to choose a system-level mechanism that will minimize the loss of system-wide efficiency in the face of the self-interested behaviors of the providers.

Since edge computing is a new computing paradigm, both industry and academia are exploring various design options of edge computing systems. We believe that the system model proposed in this paper is well suited for edge computing applications, and it can be realized through the recent advances in container technologies (such as Kubernetes [5] and Docker [6]) and IoT data management platform (e.g., FogLAMP [7]). These state-of-art technologies make it feasible to deploy an edge system that relies on computing resources from various sources at edge or from cloud, and to meet the computation demands of the users distributed at the edge of the Internet. Since service or resource providers in a distributed edge system compete with each other (as described before), a non-cooperative game-theoretic model is a natural modeling choice that can be used to investigate the system.

In this paper, we take a game-theoretic approach to explore the design of revenue sharing mechanisms for an Edge-Cloud system, in order to improve its system-wide efficiency. Our major contributions are summarized below.

1. We introduce a game-theoretic framework to investigate an Edge-Cloud computing system that consists of edge service providers and cloud service providers. The service providers (including edge and cloud providers) play a non-cooperative game among themselves. They offer their computing servers to the system which then allocates the servers optimally to execute the computation jobs submitted by clients or end users. We first show theoretically and experimentally the existence of Nash equilibrium in the game between edge and cloud providers, under three revenue sharing mechanisms (Shapley value sharing [8], Ortmann proportional sharing [9], and Direct-contribution-based sharing), and across a wide range of system/networking settings. Our findings demonstrate that different revenue sharing mechanisms can result in drastically different system-level utility loss at system equilibrium states when compared with maximum system utility (which is achieved when providers do not game with the system). Thus, it is crucially important to design an appropriate revenue sharing mechanism in order to maintain high system-level efficiency in the face of service providers' self-interested behaviors.
2. We find that at the Nash equilibria of the game, revenue sharing mechanisms based on marginal contributions in general can lead to better system performance when performed at provider level than performed at server level. In addition, Direct-contribution-based sharing (i.e., revenue split based directly on actual contributions of servers) results in significantly worse system-level performance than revenue sharing mechanisms based on marginal contributions (i.e., Shapley sharing and Ortmann sharing). We further show that the reason for this seemingly counter-intuitive result is that revenue sharing based on marginal contributions discourage providers with less powerful resources from contributing resources to the system at equilibrium state, and therefore they can achieve significantly better system performance than Direct-contribution-based mechanism even when self-interested service providers attempt to game with the system for their own maximal benefits.
3. We demonstrate that our framework is an effective economics approach to the understanding and designing of efficient Edge-Cloud computing systems, based on our extensive simulations and experiments on an Edge-Cloud emulation system that we have developed. Furthermore, we have verified our findings through simulations based on Google cloud data trace [10].

In the rest of the paper, we first present the architecture of an Edge-Cloud system and give an overview of our game-theoretic framework in Section 2. Then in Sections 3 and 4, we describe task distribution mechanisms and revenue sharing mechanisms. Section 5 presents our findings via experiments and simulations, and we conclude the paper in Section 6.

2. Edge-Cloud system

2.1. Background and related work

This paper studies a computing system in the emerging edge computing paradigm [1–4,11–17], which broadly includes cloudlets, mobile edge computing, fog computing, fog networks, and mobile cloud computing [14–16]. Besides the low latency benefit of edge computing, recent research on Internet of Things (IoT) has shown that, by processing at the edge the large amount of raw data collected by IoT sensors (e.g., in a smart home or smart city system) or human users (e.g., videos, pictures taken by smartphones), edge computing can significantly reduce the consumption of network bandwidth in wide area and core networks when compared with transferring raw data to a cloud [18–25]. AT&T, a large Internet service provider, has recently announced plans to deploy edge computing servers in their mobile access towers on a large scale [26].

The economics and game-theoretic approach adopted in this paper is related to the existing rich literature of applying economics and game theory in networking research [17,27–32]. For example, Ma et al. [29] utilize the Shapley value solution to implement the profit distribution mechanism to maximize the profits of ISPs. [30] studies the incentive structure of content delivery networks by taking a cooperative game theory approach. It also utilizes the Shapley value to study the cooperative game. In addition, besides applying Shapley value in networking research, some existing work focuses on the optimal revenue maximization and/or revenue sharing in edge computing system. For example, Samanta and Chang [27] study the offloading scheme in edge computing system. Different from our paper that considers the non-cooperative game between different providers, Samanta and Chang [27] study the overall system revenue and total services latency. Yu and Tang [28] study the problem of competition and cooperation between an edge cloud and a remote cloud. And they formulate the interaction between the providers as a three-stage Stackelberg game. However, our work focuses on how different sharing mechanisms impact the overall system efficiency at Nash equilibrium points of a game where there is no leading player. In addition, recent work has shown that proving the existence and further finding Nash equilibria of a practical complex game are quite challenging. For example, existing work [33–37] demonstrate the difficulties on finding Nash equilibrium of a two-player or multiple-player game.

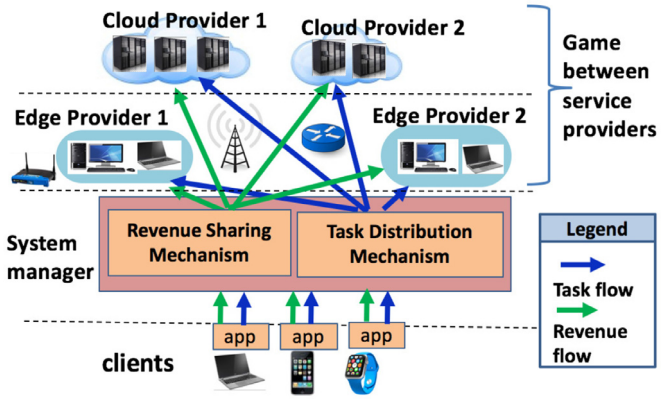


Fig. 1. System architecture. Computing service providers compete with each other under system-level mechanisms: task distribution and revenue sharing.

The novelties introduced in this article include: (1) A novel game theoretic model of the competition among edge and cloud service providers under system-level revenue sharing and task distribution mechanisms in an edge-cloud system; (2) The existence of Nash equilibrium or stable system state in the game of the competition among providers; (3) Revenue sharing mechanism based on marginal contributions is more preferable over sharing mechanism based on direct contributions, as the former can result in higher system-level efficiency at the equilibrium states of the game.

2.2. System architecture overview

An Edge-Cloud computing system is a hybrid system that integrates both edge computing and cloud computing to provide services to clients. The edge computing part of the system can execute tasks at the edge of the Internet to reduce network transmission cost of large amounts of data and to meet the low latency requirement of computing tasks. In addition, the cloud computing part of the system can provide necessary global analytics and execute tasks with flexible latency requirement.

There are three types of entities in an Edge-Cloud system, as shown in Fig. 1. (1) Clients, including applications and end users, that are at the edge of the Internet and submit computing tasks to the system. (2) Edge providers (i.e., computing service providers at the edge of the Internet and close to clients, and hence have high communication bandwidth and low propagation delay to clients), and cloud providers (providers in the cloud that offer servers to edge clients by joining an Edge-Cloud system). (3) A system manager, which is a collection of software components that implements mechanisms/algorithms for various management and scheduling issues such as facilitating task submissions, revenue collection from clients, revenue split among servers, accounting/auditing, etc. The main part of the manager resides on the edge and some of its components are distributed among providers throughout the Internet. The functionalities of the manager can be distributed among multiple edge servers to avoid single point of failure. In the system, clients communicate with and submit their tasks to the system manager through apps on their devices.

In an Edge-Cloud system, a monetary value is associated with each task. A system manager's objective is to maximize its total values or *revenue* received from clients via a task distribution mechanism that optimally assigns tasks to servers subject to the latency requirements of those tasks. Based on the revenue collected and the tasks completed by the servers, the manager utilizes a revenue sharing mechanism to split the received revenue among the servers (and hence between the service providers who own those servers)[1]. Therefore, an Edge-Cloud system has two basic types of mechanisms: a task distribution mechanism and a revenue sharing mechanism. In this paper, we investigate three types of revenue sharing mechanisms: Shapley value sharing [8], Ortmann proportional sharing [9], and Direct-contribution-based sharing. They will be discussed in detail in Sections 3 and 4.

In our design, the system does not have a leading provider to lead the system. Note that various edge systems/platforms are being proposed and explored in industry and academia. Certainly there may exist an edge system with a leading provider, which is similar to a traditional cloud computing system, but a platform without a leading provider may exist as well, and in fact, we believe that a platform without a leading provider will be a common type of design as it is well suitable for edge systems where computing resources are placed at the edge of the Internet and owned by various independent parties, which is quite different from traditional cloud computing.

[1] A system manager may keep a share of the total received revenue and split the rest among servers. We assume that a system manager's own revenue share is negligible compared with the rest of the revenue given to servers.

Table 1
Table of Model parameters.

Parameter	Description
p	player/provider p
$-p$	the set of all players/providers except p
n_p	the number of servers that player p places in the system
$U_p(n_p)$	the utility function of player p
$v_p(n_p)$	the revenue received by player p
$f_{cost}(n_p)$	the cost function of a player p
$\mathbb{N}_J$; N_J	the set of tasks; the number of tasks
$\mathbb{N}_S$; N_S	the set of servers; the number of servers
M_{opt}	task distribution mechanism
M_{share}	revenue sharing mechanism
i; j	task i; server j
d_{ij}	the completion time of task i when it is assigned to server j
x_{ij}	the assignment of task i to server j
v_i	the value of task i
L_i	latency requirement of task i
E; C	an edge provider; a cloud provider
$\phi_j(\mathbb{N}_S)$	the revenue share assigned to server j by Shapley value mechanism
$\upsilon(\mathbb{S})$	the value of set $\mathbb{S}$
ϕ_j^P	the revenue share assigned to server j by Ortmann sharing mechanism
T	system time duration
λ	task arrival rate
f_{delay}	the function to calculate the completion time of a task
$k_{latency}$; k_{bw}	a latency factor; a bandwidth factor

2.3. A Game-Theoretic framework

2.3.1. Assumptions

We assume that each service provider in an Edge-Cloud system is independent and self-interested. This assumption describes an important characteristic of an Edge-Cloud system: a service provider can choose to join an Edge-Cloud system and decides by itself the amount of computing resources it offers to the system. This characteristic differentiates an Edge-Cloud system from a traditional cloud computing system in which all computing resources are centrally managed and tightly controlled by an entity and they are typically placed in data centers. A traditional cloud computing provider can also join an Edge-Cloud system by offering service to the clients in the system. Hence an Edge-Cloud system consists of edge providers and cloud providers.

We assume that the two types of system-level mechanisms (i.e., task distribution and revenue sharing) are publicly known to all clients and service providers. Under those mechanisms, a service provider attempts to maximize its received benefit or utility (defined below) by strategically adjusting the computing resources it provides to the system. Note that the parameters of our system model are listed in Table 1.

2.3.2. The game

We model the competition among service providers in an Edge-Cloud system as a non-cooperative game [38], and the providers are *players* in

the game. We focus on the case where a provider's available strategy is to adjust the number of servers it offers to the system in order to maximize its utility. A utility function captures the tradeoff between the revenue and the cost of a provider. Providing more servers will incur more cost to a provider, even though more servers imply more revenue that the provider can potentially receive. Then the utility function of a provider p can be described as

$$U_p(n_p, n_{-p}) = v_p(n_p, n_{-p}) - f_{cost}(n_p, n_{-p}), \tag{1}$$

where $v_p(n_p, n_{-p})$ is the revenue received by provider p when placing n_p servers in the system, and the cost f_{cost} is an increasing function of the number of servers. Note that $-p$ denotes the set of all players except p.

Now we define a Nash equilibrium [38] for the game we study. Note that each player can be either an edge provider or a cloud provider in the system. Let v_p denotes the revenue received by an arbitrary player $p, \forall p \in \{1, 2, \dots, m\}$. Let M_{opt} and M_{share} denote a task distribution mechanism and a revenue sharing mechanism respectively. Then we know v_p is a function of M_{share} and M_{opt}. Note that M_{opt} is a function of (n_p, n_{-p}), i.e., the number of servers provided by the player p and the number of servers of all the other players.

Each player tries to solve its own optimization problem. Player p attempts to solve the following optimization problem

$$\max_{n_p} U_p(n_p, n_{-p}) = v_p(M_{share}(M_{opt}(n_p, n_{-p}))) - f_{cost}(n_p). \tag{2}$$

A Nash equilibrium of the game is a particular combination of players' strategies from which a player has no incentive to unilaterally deviate (i.e., it does not want to change its number of servers, given that the other players' numbers of servers remain unchanged), as any unilateral deviation will not increase its utility. The Nash equilibrium of a m-player game is denoted by

$$\{n_1^*, n_2^*, \dots, n_p^*, \dots, n_m^*\}, \tag{3}$$

where $n_p^* = \text{argmax}_{n_p} U_p(n_p, n_1^*, n_2^*, \dots, n_{p-1}^*, n_{p+1}^*, \dots, n_m^*)$. The recent advances in computational and algorithmic game theory have shown that finding a Nash equilibrium is hard. Usually problem-specific structures have to be exploited in order to find an equilibrium [33–37].

Note that the problem studied in the paper can be well understood through a non-cooperative game as providers compete with each other under the two system-level mechanisms. In addition, users do not purchase resources from providers through a market, therefore auction theory is not applicable to the problem studied in the paper.

In addition, in our game model the providers do not receive revenue or profit directly from end users. Instead, they receive revenue shares indirectly from the edge system. It is the revenue collected by the edge system that is then split or shared between the providers. The providers do not cooperate for revenue sharing, instead, they compete against each other in order to increase their revenue shares, given that they are aware of how the system decides the revenue shares for them through the revenue sharing mechanism.

In an edge/cloud computing market which is dominated by several giant resource providers competing with each other, our framework can be adopted by those providers and edge systems. The providers that join an edge system are players and they compete with each other under the two system-level mechanisms of the edge system, and their fair revenue sharing can be decided by an edge system through an optimal sharing mechanism.

3. Distribution of computing tasks

3.1. Objective of task distribution

Since an important application of edge computing is to serve tasks with low latency requirement, we focus on tasks with completion deadlines. Recall that a task has a value, which can be regarded as the payment that the task's owner (i.e., a client) is willing to pay for completing the task. *The objective of a task distribution mechanism is to maximize the total received value (as a revenue) for the tasks that are completed before their deadlines.* In this section, we present an optimization formulation for the case where tasks arrive in a batch (i.e., at the same time) to illustrate the characteristic of task distribution, and then we will present a greedy algorithm to address a practical dynamic task arrival setting.

3.2. Optimal task distribution formulation

We consider both batch task arrival and dynamic task arrival in designing optimal task distribution mechanisms. Recall the game is played under task distribution mechanisms.

3.2.1. Batch task arrival

A system manager distributes all tasks arriving in a batch to servers by solving an optimization problem to maximize the total received revenue from those completed tasks. We assume that the execution order of those tasks on a server should be the same as the order of their arrivals. We also assume that tasks are not splittable.

Let $\mathbb{N}_J$ denote the set of tasks with $N_J = |\mathbb{N}_J|$, and let $\mathbb{N}_S$ denote the set of servers with $N_S = |\mathbb{N}_S|$. Tasks are ordered increasingly according to their arrival times and indexed by $i = 1, \dots, N_J$, and servers are indexed by $j = 1, \dots, N_S$. Let x_{ij} denote the assignment of task i to server j. Then $x_{ij} = 1$ indicates that task i is assigned to server j; otherwise $x_{ij} = 0$. Let d_{ij} denote the completion time of task i when it is assigned to server j. Note that d_{ij} includes the computation time of task i on server j and the time to transfer task i to server j. In addition, a task i might experience a queuing delay if some other tasks are scheduled on the same sever (as task i) but should be executed before task i as they arrive earlier than task i. Queuing delay is discussed next.

Let v_i denote the value of task i or the payment that the owner (i.e., client) of task i will pay for completing task i. If task i is completed before its deadline, the system manager will receive v_i; otherwise, the manager receives nothing. The objective of the manager is to maximize its total received payment or value (as a revenue) by solving the following optimization problem.

$$\max_{x_{ij}} \quad \sum_{j=1}^{N_S} \sum_{i=1}^{N_J} v_i x_{ij} \tag{4}$$

$$s.t. \quad 0 \leqslant \sum_{j=1}^{N_S} x_{ij} \leqslant 1, \quad \forall i \tag{5}$$

$$x_{ij} \in \{0, 1\}, \forall i; \forall j \tag{6}$$

$$x_{ij} d_{ij} + \sum_{k=1}^{i-1} q_{ijk} d_{kj} \leqslant L_i, \quad \forall i, \forall j \tag{7}$$

$$x_{ij} = 0 \rightarrow q_{ijk} = 0, \forall i, \forall k \in \{1, \dots, i-1\}; \forall j \tag{8}$$

$$x_{ij} = 1 \rightarrow q_{ijk} = x_{kj}, \forall i, \forall k \in \{1, \dots, i-1\}; \forall j \tag{9}$$

$$q_{ijk} \in \{0, 1\}, \forall i, \forall k \in \{1, \dots, i-1\}; \forall j \tag{10}$$

where (5) and (6) require that a task can be assigned to at most one server. The three constraints (7), (8), and (9) collectively require that when assigned to server j, task i should be completed no later than its deadline (i.e., the maximum allowed latency L_i). Task i's total delay on server j is given by $x_{ij} d_{ij} + \sum_{k=1}^{i-1} q_{ijk} d_{kj}$, as shown in (7). The two constraints (8) and (9) indicate that q_{ijk} is equivalent to $x_{ij} x_{kj}$. Note that (8) and (9) are called indicator constraints in CPLEX solver [39]. The $\sum_{k=1}^{i-1} q_{ijk} d_{kj}$ represents the queuing delay of task i if it is assigned to server j. Recall that the tasks are served in a first-come first-serve order. If task k arriving before task i (with $k \in \{1, \dots, i-1\}$) is also assigned to server j, then task i has to wait till task k is finished. The queuing delay of task

i on server j only makes sense when task i is assigned to server j. Therefore, (8) says that when task i is not assigned to server j, its queuing delay constraint (7) on server j should be removed[2].

3.2.2. Dynamic task arrival

The above optimization formulation for batch arrival of tasks illustrates the nature of the optimization problem to be solved by an Edge-Cloud system's manager, but it is difficult to implement in practice. This is because usually tasks arrive in a dynamic fashion, and since they have deadlines, they need to be sent to available servers immediately in order to meet their latency requirements.

To address the case of dynamic task arrival, we introduce an online greedy algorithm (shown as Algorithm 1) to be used by a system manager to maximize its total received revenue. The idea of the algorithm is: whenever a server is available, it should be given to the task with the highest value among all tasks that are present in the system and can be completed before their deadlines by the server. Note that servers become available at different times, which depends on the tasks they are currently working on and the task arrival process.

Algorithm 1 Online Greedy Task Distribution Algorithm.

Require: $\langle \mathbb{N}_J(T), \mathbb{N}_S, T\rangle$,where T is the time period during which the algorithm executes,and $\mathbb{N}_J(T)$ is a set of tasks and their arrival times during T, and $\mathbb{N}_S$ is a server set.

1: $t \leftarrow 0$, $Q = \emptyset$ (Q is a priority queue where the task with the highest value is at the front of Q).
2: **while** $t \leq T$ **do**
3: If a task arrives at time t, insert it into Q:
4: If multiple tasks have the same value, order them according to their arrival time order.
5: If a set of servers are available at time t (denoted by $\mathbb{S}_t \subseteq \mathbb{N}_S$),use a loop to select all servers one at a time and in random order from $\mathbb{S}_t$, and for each selected server svr_j:
6: Start from the front of Q, search for the task with the highest valueamong all tasks that can be finishedbefore their deadlines if processed by svr_j. Let $task^*$ denote such a task.
7: If $task^*$ is found, stop search and start a new thread for svr_j to work on $task^*$.
8: **end while**

If all tasks arrive at the same time at the beginning, then Algorithm 1 is essentially a heuristic to solve the optimization problem (4) formulated for the batch task arrival case. In addition, if a task's value is inversely proportional to its deadline, then Algorithm 1 is a variant of earliest-deadline-first scheduling algorithm, but without preemptive scheduling. In an Edge-Cloud system, a server cannot suspend the execution of a task in order to execute some other task with higher priority, due to the non-negligible communication cost/delay in edge computing environment. Both batch arrival and dynamic arrival of tasks will be investigated in Section 5.

4. Mechanisms for revenue sharing

We investigate the following three revenue sharing mechanisms: (1) Shapley value [8]; (2) A proportional sharing mechanism proposed by Ortmann [9], referred to as *Ortmann proportional sharing*; (3) and a sharing mechanism based directly on each server's actual contribution, referred to as *Direct-contribution-based* sharing mechanism.

[2] We do not consider task drops due to buffer overflow by assuming that the system's task queue is sufficiently large and the system's task processing capacity is well provisioned.

4.1. Shapley-value revenue sharing mechanism

Shapley value [8] is a marginal contribution based revenue sharing mechanism. For an Edge-Cloud system, we define Shapley value as a function that distributes among a set of servers the total revenue received by the system in organizing the servers to work on a set of tasks. It specifies that the revenue a server receives equals the server's expected marginal contribution.

Formally, consider a set of tasks $\mathbb{N}_J$, and a set of servers $\mathbb{N}_S$ (with $N_S = |\mathbb{N}_S|$). Note that a server can be owned by a cloud provider or an edge provider. Define *the value of set* $\mathbb{S}$, denoted by $v(\mathbb{S})$, as the total received value by only using servers in set $\mathbb{S}$ (with $\mathbb{S} \subseteq \mathbb{N}_S$) to work on the tasks in $\mathbb{N}_J$. Note that v is a function of task distribution mechanism. Let ϕ_i denote the revenue share given to server i. The Shapley value mechanism assigns the following revenue share to server i:

$$\phi_i(\mathbb{N}_S) = \frac{1}{N_S!} \sum_{\mathbb{S}\subseteq\mathbb{N}_S\setminus\{i\}} |\mathbb{S}|!(N_S - |\mathbb{S}| - 1)!\big(v(\mathbb{S}\cup\{i\}) - v(\mathbb{S})\big). \tag{11}$$

This revenue distribution mechanism satisfies the following desired property [8,30,40,41]: *fairness or balanced contribution*, which says that for any pair of servers $i, j \in \mathbb{N}_S$, j's contribution to i equals i's contribution to j, i.e., $\phi_i(\mathbb{N}_S) - \phi_i(\mathbb{N}_S \setminus \{j\}) = \phi_j(\mathbb{N}_S) - \phi_j(\mathbb{N}_S \setminus \{i\})$. Shapley value sharing mechanism also has some other important properties such as symmetry, efficiency (the sum of revenue shares distributed to all servers equals the total received revenue), and dummy (i.e., a player will not receive any revenue share if it does not make any contribution in any coalition). In the system we study, a server will always receive non-zero revenue share because there always exists some coalition in which the server can make some contribution.

4.1.1. Computing shapley values

The amount of time to compute Shapley values for all servers in a game is exponential, if the computation is done according to the definition in Eq. (11). However, we are able to derive a polynomial time algorithm, shown as Algorithm 2, based on the following assumptions. The servers in a system can be divided into groups, which belong to different providers. For ease of exposition, assume that a provider has one and exactly one group. Further we assume that the servers in a group or provider have similar capacity (in terms of CPU, path bandwidth, and propagation delay) as they are offered by the same provider. Then the Shapley values for all servers in a provider should be the same. Therefore for provider k, we just need to calculate a Shapley value ϕ_i, where i is an arbitrary server in the set of servers of provider k (denoted by $\mathbb{N}_S^k$), i.e., $i \in \mathbb{N}_S^k$. Then, provider k's revenue is $v_k(N_k) = N_k\phi_i$, with $N_k = |\mathbb{N}_S^k|$.

An example of computing Shapley values. We use a simple example of two providers to illustrate Algorithm 2. Consider a cloud provider that offers two cloud servers c_1, c_2 and an edge provider that offers three edge servers e_1, $e2$, and $e3$ in an Edge-Cloud system. According to Eq. (11), in order to derive the Shapley value of c_1, we need to find the values of sets $\mathbb{S}$ and $\mathbb{S}\bigcup\{c_1\}$. There are 32 such sets (including the empty set).

However, all three edge servers can be treated as equivalent, and both cloud servers are also equivalent. For example, the value of set $\{c_1, e1, e2\}$ equals the values of sets $\{c_1, e1, e3\}\{c_1, e2, e3\}$, $\{c_2, e1, e2\}$, $\{c_2, e1, e3\}$ and $\{c_2, e2, e3\}$. Therefore, to calculate the Shapley value of cloud server c_1, we only need to calculate the values of the following *eleven* sets $\{c_1\}$, $\{e_1\}$, $\{e_1, c_1\}$, $\{e_1, e_2\}$, $\{c_1, c_2\}$, $\{e_1, e_2, c_1\}$, $\{e_1, e_2, e_3\}$, $\{e_1, c_2, c_1\}$, $\{e_1, e_2, e_3, c_1\}$, $\{e_1, e_2, c_2, c_1\}$, $\{e_1, e_2, e_3, c_2, c_1\}$, based on Algorithm 2. Essentially, we reduce the computation time from exponential to polynomial.

4.1.2. Time complexity of Algorithm 2

Recall that m is the number of providers and it does not depend on N (the number of servers). In practice, m is always upper bounded and small because a resource provider that serves a particular region either belongs to a local/regional network service provider or a national

Algorithm 2 Compute the Shapley value of each server of m providers in an Edge-Cloud system..

Require: Task set $\mathbb{N}_J$; Server set $\mathbb{N}_S = \bigcup_k \mathbb{N}_S^k$, where $\mathbb{N}_S^k$ isthe set of servers of provider k, $N_k = |\mathbb{N}_S^k|$, $N = |\mathbb{N}_S|$, $k = 1, 2, \ldots, m$.

1: **for** $j = 1, 2, \ldots, m$ **do**
2: Initialize $V_j^s = 0$.
3: **for** $n_j = 0$; $n_j \leq N_j - 1$ **do**
4: Do a $m-1$ level nested loop to find all combinations$(n_1, n_2, \ldots, n_{j-1}, n_{j+1}, \ldots, n_m)$,where each number n_k (with $k \in \{1, 2, \ldots, j-1, j+1, \ldots, m\}$) varies from 0 to N_k.
5: For each $(n_1, n_2, \ldots, n_{j-1}, n_j, n_{j+1}, \ldots, n_m)$, do:
6: Invoke a task distribution algorithm (e.g., **Algorithm 1**)for task set $\mathbb{N}_J$ to calculate two values V_1 and V_2 :
7: $V_1 = V[n_1][n_2]\ldots[n_{j-1}][n_j][n_{j+1}]\ldots[n_m]$, i.e., the value of the setthat contains n_k servers from provider k, with $k \in \{1, 2, \ldots, m\}$.
8: $V_2 = V[n_1][n_2]\ldots[n_{j-1}][n_j + 1][n_{j+1}]\ldots[n_m]$. (Similar to V_1, except that $n_j + 1$ is used for provider j).
9: Calculate $C_{coeff} = C_{N_1}^{n_1} \cdot C_{N_2}^{n_2} \cdots C_{N_{j-1}}^{n_{j-1}} \cdot C_{N_j - 1}^{n_j} \cdot C_{N_{j+1}}^{n_{j+1}} \cdots C_{N_m}^{n_m}$.(Note that $N_j - 1$ is used for provider j).
10: Let $S = \sum_{k=1}^{m} n_k$.
11: Calculate $V^{inc}_{n_1, n_2, \ldots n_{j-1}, n_j, n_{j+1} \ldots n_m} = \frac{S!}{N!}(N - S - 1)! \cdot C_{coeff} \cdot (V_2 - V_1)$.
12: Increase V_j^s by $V^{inc}_{n_1, n_2, \ldots n_{j-1}, n_j, n_{j+1} \ldots n_m}$.
13: **end for**
14: Record V_j^s (Shapley value of a server in provider j).
15: **end for**
16: Return Shapley value $\phi_{i \in \mathbb{N}_S^j}(\mathbb{N}_S) = V_j^s$,with $j = 1, 2, \ldots, m$. (All servers in a provider havethe same Shapley value.)

provider (e.g., AT&T [26])[3]. Consider a system of two providers, in which an edge provider has a set of servers $\mathbb{N}_1 = \{e_1, e_2, \cdots, e_{N_1}\}$, $|\mathbb{N}_1| = N_1$, and a cloud provider has a set of servers $\mathbb{N}_2 = \{c_1, c_2, \cdots, c_{N_2}\}$, $|\mathbb{N}_2| = N_2$. All cloud servers are equivalent, and all edge servers are equivalent. And the set of all servers is $\mathbb{N} = \{e_1, e_2, \cdots, e_{N_1}, c_1, c_2, \cdots, c_{N_2}\}$, $|\mathbb{N}| = N_1 + N_2$. Suppose we would like to apply Algorithm 2 to calculate the Shapley value of a particular edge server e_i that belongs to the edge provider with server set $\mathbb{N}_1$. For any two subsets S_1 and S_2 of $\mathbb{N}$ that do not contain e_i, we have $v(S_1) = v(S_2)$ and $v(S_1 \bigcup e_i) = v(S_2 \bigcup e_i)$, if the number of edge servers in S_1 equals the number of edge servers in S_2, and the number of cloud servers in S_1 equals the number of cloud servers in S_2. Then, we only need to calculate the values of $(N_1 + 1)(N_2 + 1) - 1$ sets. This is because, for a set listed in the Shapley value formula (11), the set might contain a number of edge servers and the number can be 0, 1, $\cdots$, N_1; similarly, the set might contain a number of cloud servers and the number can be 0, 1, $\cdots$, N_2. Thus, the total number of the unique sets is $(N_1 + 1)(N_2 + 1) - 1$, where the -1 is needed for removing the value calculation for the empty set $\emptyset$ which is always zero. Therefore, the time complexity of Algorithm 2 is $O(N_1 N_2)$. In general, if there are m providers which have $N_1, N_2, \ldots, N_m$ servers, the time complexity of Algorithm 2 is $O(N_1 N_2 \cdots N_m)$.

[3] It will not make any sense for a local resource provider to provide low-latency and low-network-cost edge computing service to another region that is geographically far away. In addition, cloud service providers are sufficient to meet the global analytics requirements of the clients in a particular region, and there are only very few large cloud providers in practice.

4.2. Direct-contribution-based and ortmann proportional sharing mechanisms

Direct-contribution-based sharing mechanism is explained as follows. A server is rewarded with a share of revenue that is proportional to the actual contribution that it has made when working together with other servers to complete a set of tasks. In the case where a system manager distributes all of its received revenue among participating servers (i.e., with no revenue share left for itself), the amount of revenue that a server receives is exactly the same amount of payment given by the clients whose tasks are completed by the server. Direct-contribution-based sharing can be regarded as a baseline sharing mechanism, against which other sharing mechanisms can be compared.

Ortmann proportional sharing [9] is a sharing mechanism that is similar to Shapley value, in the sense that it also relies on some calculation of the marginal contribution of a server, instead of relying directly on the server's actual contribution. For example, for a system with only two servers i and j, according to Ortmann proportional sharing, the revenue received by server i should be $\phi_i^P = \frac{v(\{i\})}{v(\{i\})+v(\{j\})} v(\{i, j\})$, where $v(\{i\})$ is the revenue or value generated by the system when it only contains server i, and $v(\{i, j\})$ is the revenue generated by the system when it contains servers i and j. Formally, Ortmann value mechanism assigns the following revenue share to server i:

$$\phi_i^P(\mathbb{N}_S) = \frac{v(\mathbb{N}_S)}{1 + \sum_{j \in \mathbb{N}_S \setminus \{i\}} \frac{\phi_j^P(\mathbb{N}_S \setminus \{i\})}{\phi_i^P(\mathbb{N}_S \setminus \{j\})}}. \tag{12}$$

Note that Ortmann proportional sharing also has a balanced contribution property, but it differs from Shapley value's balanced contribution in the following sense. Under Ortmann sharing, for any pair of servers $i, j \in \mathbb{N}_S$, j's contribution to i equals i's contribution to j in terms of quotient, i.e., $\phi_i^P(\mathbb{N}_S)/\phi_i^P(\mathbb{N}_S \setminus \{j\}) = \phi_j^P(\mathbb{N}_S)/\phi_j^P(\mathbb{N}_S \setminus \{i\})$. That is, Ortmann's balanced contribution property is in the form of ratio equality instead of the difference equality of Shapley's. For example, consider a system of two players 1 and 2. Let $v(\{1\}) = 2$, $v(\{2\}) = 6$, and $v(\{1, 2\}) = 40$. According to Shapley value sharing, they will get $\phi_1^S = 18$ and $\phi_2^S = 22$; according to Ortmann proportional sharing, they will get $\phi_1^P = 10$ and $\phi_2^P = 30$. The Shapley value sharing satisfies $18 - 2 = 22 - 6$, but Ortmann proportional sharing satisfies $10/2 = 30/6$.

In this paper, we consider both Shapley and Ortmann sharing mechanisms at two different levels, provider level and server level. At provider level, a provider with all of its servers is regarded as a single player in a revenue sharing mechanism. At server level, we consider each server as an independent player, and sum the total values of all servers of a provider as the value of the provider. For example, consider a system of two providers, edge provider E and cloud provider C. Edge provider E has two servers, 1 and 2. Cloud provider C only has one server, 3. Let $v(\{1\}) = 2$, $v(\{2\}) = 2$, $v(\{3\}) = 6$, $v(\{1, 2\}) = 10$, $v(\{1, 3\}) = 30$, $v(\{2, 3\}) = 30$, and $v(\{1, 2, 3\}) = 40$. According to Shapley value sharing at provider level, we have $\phi_E^S = 22$ and $\phi_C^S = 18$; according to Ortmann proportional sharing at provider level, we have $\phi_E^P = 25$ and $\phi_C^P = 15$. The Shapley value sharing satisfies $22 - 10 = 18 - 6$, but Ortmann proportional sharing satisfies $25/10 = 15/6$. According to Shapley value sharing at server level, we have $\phi_E^S = \frac{56}{3}$ and $\phi_C^S = \frac{64}{3}$; according to Ortmann proportional sharing at server level, we have $\phi_E^P = \frac{160}{13}$ and $\phi_C^P = \frac{360}{13}$.

In comparison, as for Direct-contribution-based mechanism, we do not distinguish between provider level and server level. This is because the revenue share that a provider receives is the same at both provider level and server level.

4.3. Equilibrium state of an edge-Cloud system

Recall that we model the competition among multiple service providers as a non-cooperative game, which is under the two system-level mechanisms: task distribution mechanism and revenue sharing

mechanism. In this section, we prove the existence of Nash equilibrium of the game.

Theorem 1. *The game of multiple resource providers in an Edge-Cloud system has a mixed-strategy Nash equilibrium point, which is an equilibrium system state.*

Proof. Recall that in an Edge-Cloud system there is a set of computing tasks denoted by $\mathbb{N}_J$, a set of servers denoted by $\mathbb{N}_S$ with $N_S = |\mathbb{N}_S|$, and m providers. Without loss of generality, we consider provider i, and its set of servers is denoted by $\mathbb{N}_S^i$, and its number of servers is denoted by $N_i = |\mathbb{N}_S^i|$, $i = 1, 2, \ldots, m$. Let $-i$ represent the set of all providers except provider i, $V_{opt}(N_i)$ denote the maximum value of the objective function in Eq. (4). Let N_{-i} be fixed, $V_{opt}(N_i)$ is a non-decreasing function of N_i.

Note that there always exists a minimum number of servers that provider i can place in the system so that all tasks can be completed in the required time duration. Let $\bar{N}_i$ denote this threshold number. When $N_i \geq \bar{N}_i$ and N_i increases, $V_{opt}(N_i)$ remains the same value. That is, once there is a sufficient number of servers from provider i that can satisfy the server allocation to tasks to get the optimal value (in solving the task distribution optimization problem), then further increasing N_i will not get any additional value increase of the objective function.

Let V_{opt}^* denote the stable maximum objective value of $V_{opt}(N_i)$, that is,

$$V_{opt}^* = V_{opt}(\bar{N}_i). \tag{13}$$

Let $V_i(N_i)$ denote the revenue received by provider i as the result of a revenue sharing mechanism (e.g., Shapley value) that splits the total value/revenue by solving the task distribution optimization problem. Then,

$$V_{opt}^* = \sup_{N_i \in \mathbb{Z}^+ \cup \{0\}} V_i(N_i). \tag{14}$$

Here we focus on a linear cost function $f_{cost}(N_i) = \alpha_i N_i$ with $\alpha_i > 0$.

Eq. (1) shows that the utility of provider i is:

$$U_i(N_i) = V_i(N_i) - \alpha_i N_i. \tag{15}$$

If we take Eq. (14) into consideration, then we have

$$U_i(N_i) = V_i(N_i) - \alpha_i N_i \leqslant V_{opt}^* - \alpha_i N_i. \tag{16}$$

Note that $\alpha_i > 0$, thus when $N_i > \frac{V_{opt}^*}{\alpha_i}$, $U_i(N_i) < 0$. In addition, this also indicates that once N_i reaches a certain value, provider i's utility will be a decreasing function of N_i. Therefore, N_i is upper bounded[4]. Meanwhile, the number of servers of any provider must be a non-negative integer, which means $N_i \in \mathbb{Z}^+ \cup \{0\}$. The above discussion shows that the game is finite [42], and hence the theorem is proved. □

Remarks: We observe in our experiments that pure-strategy Nash equilibrium also exists in some cases, which are shown in Fig. 5(a) and (b) in Section 5. When there is no pure-strategy Nash equilibrium state, we use the system performance at mixed-strategy Nash equilibrium to represent system stable state. When we observe multiple Nash equilibria in some of our experiments, we examine the average system performance at those equilibrium states.

5. Impact of revenue sharing mechanisms on system performance

We discuss in this section the impact of revenue sharing mechanisms on the performance of an Edge-Cloud system. Our investigation is conducted through a combination of emulations and simulations[5].

[4] In practice, N_i is also upper bounded as the number of servers of a provider is always limited.

[5] Due to resource constraints, it is impossible for us to conduct Internet-scale experiments. Therefore we mainly rely on simulations with system parameters derived from our experiments and empirical trace.

Fig. 2. Edge-Cloud emulation system.

5.1. Edge-Cloud emulation system

We have built an experimental system to emulate an Edge-Cloud system, and based on which we have conducted experiments to derive system parameters to drive our simulations. The system consists of a pool of edge clients (on a number of Raspberry Pi's [43] and Ubuntu laptops), a system manager (a distributed software component), and a pool of servers (on a number of Ubuntu workstations), as shown in Fig. 2. A client at the edge submits its computation tasks to the manager at the edge who schedules and dispatches received tasks to servers. There are two types of servers in the system: edge servers and cloud servers, which respectively belong to edge service providers and cloud service providers. The edge servers have higher bandwidth and shorter propagation delays (on the paths between them and clients) than the cloud servers. Once a server receives a task from a client, a Docker container [6] will be launched on the server to process the task. Once the task is completed, the server will notify the manager and sends back the result of the task to the edge client. The communication between the clients, the manager, and the servers is through Web Application Messaging Protocol (WAMP) [44], a real-time messaging protocol.

5.2. Determining simulation parameters

We utilize image processing (a typical edge computing application) in our simulations to investigate the impact of revenue sharing mechanisms. In this subsection, we discuss how to derive system parameters used in our simulations.

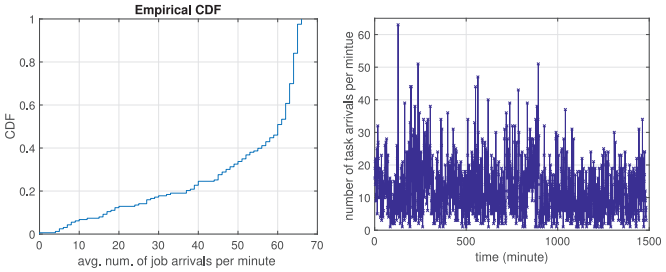
We focus on an object detection application, i.e., a client's task is to detect whether a specific object appears in a collection of images. The client submits a collection of images including the image of the target object and a number of candidate images in a batch to the system, and then the system assigns the task to a server. The server launches a Docker container [6] to process the images using OpenCV [45].

A simulation run is characterized by a group of system parameters:

$$\{T, \mathbb{N}_J(T), \mathbb{N}_S, \lambda, f_{delay}, k_{latency}, k_{bw}\}, \tag{17}$$

where T is the system time duration we simulate; $\mathbb{N}_J(T)$ denotes the set of tasks and their arrival times during T; $\mathbb{N}_S$ denotes the set of servers; λ denotes task arrival rate; f_{delay} denotes the function to calculate the completion time of a task; $k_{latency}$ denotes a latency factor; and k_{bw} denotes a bandwidth factor. These parameters are discussed below.

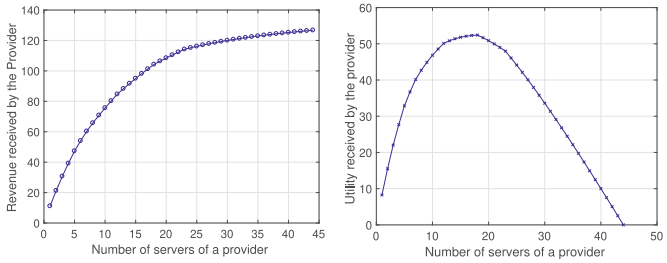
In our simulations, we let the size of a task be a uniform random number drawn from the range [1,20] MB. We can think of task size in the context of an object detection application as follows. The average task size 10 MB (in our simulations) roughly corresponds to a batch of 6 images with a typical image size (about 1.6 MB on a typical smartphone). The average task size also roughly corresponds to a collection of 63 images from the Microsoft COCO image dataset [46] with an average image size of 159 KB. The value of a task is a number chosen uniformly at random from range [1,5].



(a) Average number of task arrivals per minute when sampled every 60 minute (60-min intervals that have zero arrivals are removed).

(b) The average number of arriving tasks every minute.

Fig. 3. Google trace [10].



(a) Revenue of a provider.

(b) Utility of a provider.

Fig. 4. Revenue and utility of one provider.

We assume that tasks arrive at the system in a Poisson process with rate λ (number of tasks per minute). Poisson process is a typical stochastic process used in modeling task arrival process. We choose $\lambda = 40$ per minute in our simulations, which is the average task arrival rate in Google cloud trace [10]. Besides the Poisson task arrival process, we also run simulations of the empirical task arrival process from the Google dataset, as shown in Section 5.5. With Google cloud data, a task may consist of multiple jobs with different job arrival times. We take the arrival time of the earliest job of a task as the task's arrival time. Specifically, we divide the time duration of the Google task arrival trace [10] into a sequence of 60-minute time intervals, which is shown in Fig. 3, and within each interval, we calculate the average number of arriving tasks per minute. We remove those 60-minute time intervals during which there is no task arrival, and we observe that over 75 percent of those 60-minute intervals have more than 40 task arrivals per minute. Then in the simulations, we adopt a task arrival rate of 40 per minute.

The completion time f_{delay} of a task on a server depends on the server's CPU and bandwidth (of the path between itself and clients). Through our experiments on our experimental system, we find that for the object detection application, the computation time of processing a batch of images is linearly proportional to the size of the batch. For example, we tested a server in our experiments, which was a Dell mobile workstation with Intel Core *i*7 2.60 GHz, 4 cores CPU, and 16 GB memory. We randomly selected n images from Microsoft COCO image database [46] and then ran the object detection application with OpenCV. Each experimental setting was repeated 10 times. We choose $n = 10, 50, 100, 200, 300,$ and for each n value and each experiment, we recorded the total size of the batch of images. Let t_i (sec) denote the computation time of processing a batch of images of s_i MB. We found that there was a strong linear relationship between t_i and s_i, and a linear regression analysis shows that $t_i = 2.6s_i$. We will utilize this function to calculate the computation time of a task in our simulations reported in the rest of this section.

A task i has a latency requirement, denoted by L_i seconds (i.e., the maximum allowed delay). It is determined as follows. Let $L_{i,\ avg}$ denote the amount of time to complete task i on an average server (i.e., a server with an average CPU power and average bandwidth to clients in the system) without considering any queuing delay. Note that $L_{i,\ avg}$ includes computation time and task transmission time. Assume that the average upload bandwidth of the paths from clients to servers is B Mbps, and assume that an average server has a CPU similar to the one used in our experiments described above. Then, based on our experiments, we found the following empirical functional relationship, $L_{i,avg} = 2.6s_i + (8s_i/B) + d_{prop}$ sec, where s_i is task i's size (MB), and the average propagation delay d_{prop} is negligible compared with computation and transmission delays. In our experiments, we let the bandwidth from a client to an edge server be 24 Mbps (i.e., a WiFi environment), and then we let the bandwidth from a client to a cloud server be $24/k_{bw}$ Mbps, where bandwidth factor k_{bw}(= 1, 2, 3, and 4) model the practice where a cloud server usually has lower bandwidth to clients than an edge server. In calculating $L_{i,\ avg}$, we do not consider the time to transfer results back to clients as the sizes of the results are very small and negligible[6]. Then, we let L_i be a number chosen uniformly at random from range $[L_{i,\ avg},\ k_{latency}L_{i,\ avg}]$, where latency factor $k_{latency} \geq 1$. The rationale of choosing such a latency requirement is: a client should not expect that its task to be completed earlier than what an average server can offer; and it is reasonable for a client to expect its task to be completed not $k_{latency}$ times longer than what an average server can offer. The actual deadline of task i is given by $a_i + L_i$, where a_i is the arrival time of task i.

Recall that the utility of a provider p is given by $U_p(n_p) = \upsilon_p(n_p) - f_{cost}(n_p)$. We focus on a linear cost function $f_{cost}(n_p) = \alpha_p n_p$ with $\alpha_p > 0$. Thus, the utility of a provider p is given by $U_p = \upsilon_p(n_p) - \alpha_p n_p$. We choose a value for α_p to make the cost of providing a certain number of servers be comparable to the revenue received due to making those servers available[7]. Specifically, we simulated an Edge-Cloud system with an edge player and a cloud player. We repeated the simulation for various combinations of the numbers of edge and cloud servers, with a total number of servers varying from 2 to 30. Based on these simulation runs, we choose $\alpha_{edge} = 4$ and $\alpha_{cloud} = 3$ for edge and cloud player respectively, when simulation lasts $T = 60$ minutes. We let $\alpha_{edge} > \alpha_{cloud}$, as typically a cloud provider can deploy servers with a lower cost due to its economy of scale compared with edge providers.

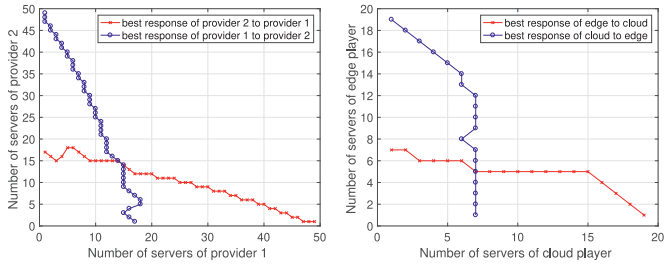
5.3. Nash equilibrium and efficiency loss

We next illustrate the existence of Nash equilibrium through a system that consists of two providers, with a set of tasks arriving in a batch, and Shapley value mechanism is adopted by the system.

First suppose that the servers of both providers are all identical in terms of CPU and bandwidth. This game is referred to as a symmetric game. Fig. 4(a) shows the revenue received by a provider by varying its number of servers in the system, when the other provider places 6 servers in the system. The revenue curve is increasing and concave, which shows that the provider's marginal revenue is decreasing even though its received revenue is increasing due to its increasing number of servers. Fig. 4(b) shows the utility received by the provider. We see that due to the decreasing marginal revenue and the increasing cost, the utility first increases and then drops.

[6] If the size of a task's result is comparable with the task's size, an edge server's advantage over a cloud server is even larger in terms of latency. The delay due to result transmission will not qualitatively affect our results.

[7] As a future work, we will decide empirically the cost coefficient α_p by considering actual operating cost, e.g., power consumption, electricity usage. We will also study other cost functions that may include investment cost, energy cost, storage cost, etc.



(a) Best response curves of two providers in a symmetric game.

(b) Best responses curves of two providers in an asymmetric game.

Fig. 5. Best response curves of two providers in two games.

Based on the utilities of the two players, we derive their best response curves [31,38] and draw them in Fig. 5(a). Each point of a player's best response curve represents the player's best strategy in response to the other player's strategy. For example, a point on the best response curve of player 2 to player 1 represents the number of servers (the y-axis value of the point) that gives player 2 the maximum utility given that player 1 chooses a certain number of servers (the x-axis value of the point). Therefore, any intersection point of the two best response curves represents a Nash equilibrium. Fig. 5(a) shows that the game has two Nash equilibria $(n_1^*, n_2^*) = (14, 15)$ or $(n_1^*, n_2^*) = (15, 14)$, where n_1^* is the optimal strategy of player 1 with respect to player 2's strategy n_2^*, and vice versa. Our simulation of 3-player games also show the existence of equilibrium states.

Note that a typical metric to measure a system's performance at a Nash equilibrium is efficiency loss [31,47], which is a comparison between the overall system utility at the equilibrium with the maximum overall system utility. We use the relative *utility loss* of an equilibrium to capture the efficiency loss of the equilibrium, which is defined as

$$U_{loss} = \frac{\left(\max_{\{n_p\}} \sum_p U_p(n_p)\right) - \left(\sum_p U_p^{NE}(n_p^{NE})\right)}{\max_{\{n_p\}} \sum_p U_p(n_p)}, \tag{18}$$

where p denotes a provider or a player, U_p^{NE} is the utility of player p at Nash equilibrium *NE*, and $\max_{\{n_p\}} \sum_p U_p(n_p)$ is the maximum overall system utility[8]. For example, in the above game with 50 tasks arriving in a batch, we observe that the system's utility loss is around 19.5% at the unique equilibrium[9].

As another example, we change the previous game by setting $k_{bw} = 4$ (then the game is asymmetric as the two players have different bandwidth). Fig. 5(b) shows the existence of Nash equilibrium in this game. Note the difference of the settings in Fig. 5(a) and (b). The servers in Fig. 5(a) are all of the same type (i.e., edge servers), and the servers in Fig. 5(b) are of two different types (i.e., edge servers and cloud servers). To execute the same number of tasks, more servers are needed in the case where only edge servers are available. Thus, the total number of servers in the system in Fig. 5(a) is greater than the total number of servers in Fig. 5(b).

Note that our experiments involve a process of finding optimal task distributions, solving revenue sharing problems, and then finding Nash equilibria. We find that this is a computationally highly expensive process. For example, for the experiments of a system of 50 computational tasks with results shown in Fig. 5(b), it took us more than 20 hours to obtain the Nash equilibrium points under only one type of task distribution and one type of revenue sharing mechanism on a Dell Precision Tower work station with 8 cores, 3.70 GHz CPU, 64 GB RAM, and a 2 TB HD. In addition, it took us more than 60 hours on the same Dell work station to generate experimental results shown in Fig. 6.

[8] It is the solution of a system-wide utility maximization problem, and the numbers of servers specified by the solution for the providers may not maximize their individual utilities.

[9] In some games, there are two Nash equilibria and they are very close to each other, similar to the case shown in Fig. 5(a). The existence of multiple equilibria is due to the discrete nature of strategies (i.e., number of servers). In those cases, utility loss is calculated as the average of those equilibria.

5.4. Impact of revenue sharing mechanism

Our extensive simulations have demonstrated the existence of Nash equilibrium in the game between edge and cloud providers in a wide range of system/network settings, when tasks arrive in a dynamic process. We find that *different revenue sharing mechanisms have quite different impacts on the performance of an Edge-Cloud system*, and in general *Direct-contribution-based sharing mechanism results in worse system-level efficiency than Shapley and Ortmann mechanisms, and both of Shapley and Ortmann mechanisms have better performance at provider level than at server level*. Due to space limitations, we only present in this section some results of the game between a cloud player and an edge player, shown in Figs. 6–8. For ease of exposition, we let the cloud player's servers differ from the edge player's servers only in the bandwidth of the paths between themselves and clients, and we let all servers have the same CPU capacity. Each simulation presented here lasts $T = 60$ minutes, and tasks arrive at the system in a Poisson process with $\lambda = 40$ tasks per minute. The results of our simulations based on the empirical task arrival process in Google trace [10] are similar to the results presented in this subsection.

5.4.1. Utility loss of shapley and ortmann mechanisms compared at provider level and serve level

In Fig. 6, we see that Shapley and Ortmann mechanisms both result in higher system efficiency at provider level than at server level in 79% cases. This happens due to the fact that the two sharing mechanisms place more edge servers at Nash equilibria at provider level to improve the system performance than at server level.

When the cloud player's transmission bandwidth is significantly lower than that of the edge player ($k_{bw} = 4$), and when tasks have very stringent latency requirement (i.e., $k_{latency} = 1.4$), Fig. 6(a) shows that Shapley mechanism gives the lowest utility loss, near 0% at provider level, and 1.98% at server level. Recall that Direct-contribution-based sharing mechanism results in the same utility loss at provider level and at server level.

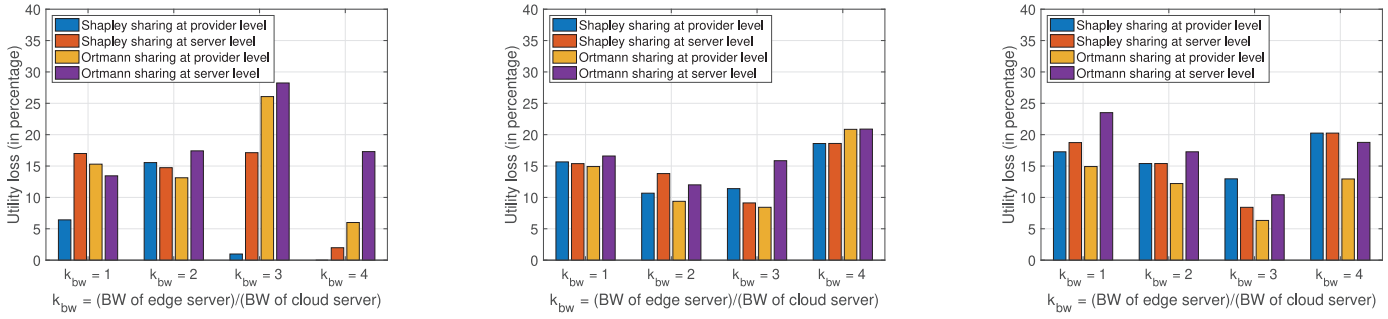
5.4.2. Utility loss of direct-contribution-based, shapley, and ortmann mechanisms at provider level

From above, we know that Shapley and Ortmann sharing mechanisms have better system performance at provider level in general than at server level. Thus, in this subsection, we compare their performance at provider level with the performance of Direct-contribution-based mechanism.

From Fig. 7, we observe that when the transmission bandwidth difference between cloud servers and edge servers is small (i.e., $k_{bw} = 1$ or 2), and the latency requirement is not stringent ($k_{latency} = 2$, or 4)[10], the losses of system utility are roughly the same for all three different revenue sharing mechanisms. Note that when the transmission bandwidth difference between cloud servers and edge servers is 1 (i.e., $k_{bw} = 1$), the only difference between edge servers and cloud servers is the cost coefficients α_{edge} and α_{cloud}.

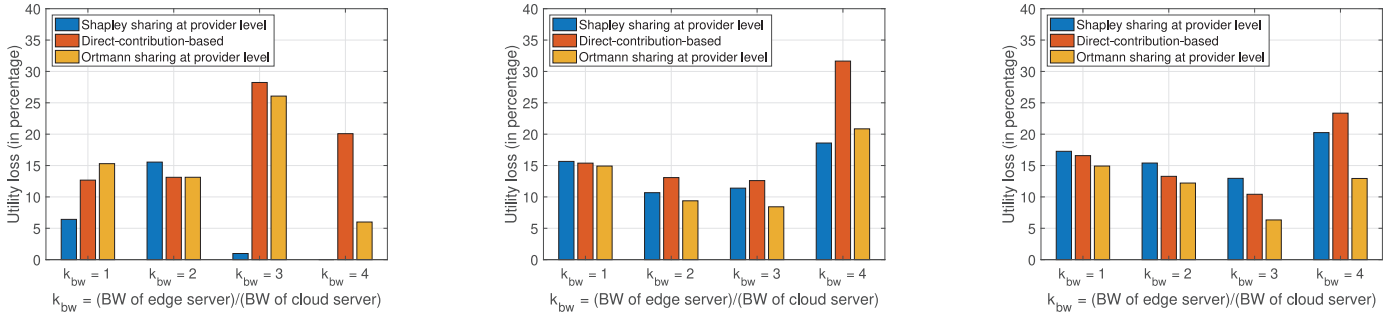
However, when the cloud player's transmission bandwidth is significantly lower than that of the edge player ($k_{bw} = 4$), Fig. 7 shows that Direct-contribution-based sharing gives the worst utility loss among all three mechanisms, across different latency requirement levels ($k_{latency}$) of tasks.

[10] If $k_{latency} = 2$, then $2L_{i,\,avg}$ is the amount of time to complete task i (without queuing delay) on the server with the lowest bandwidth.



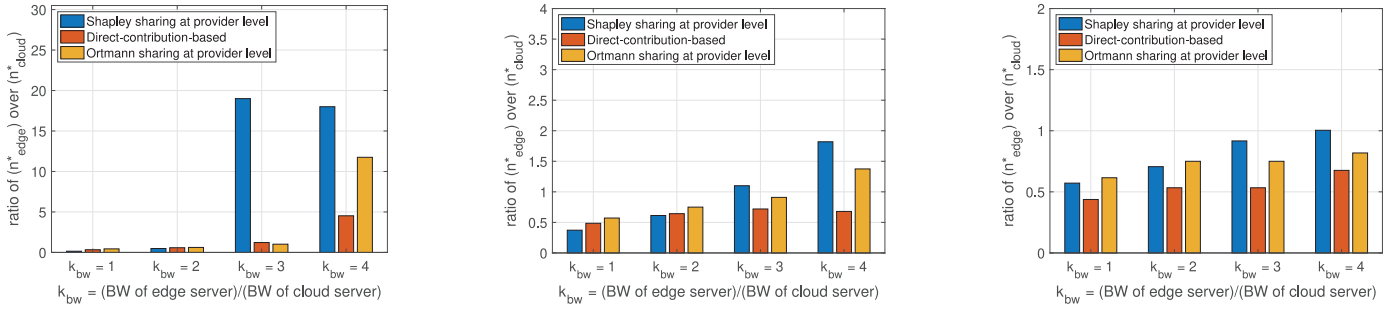
(a) $k_{latency} = 1.4$ (b) $k_{latency} = 2$ (c) $k_{latency} = 4$

Fig. 6. Utility loss of Shapley and Ortmann Sharing compared at provider and server levels at Nash equilibria.



(a) $k_{latency} = 1.4$ (b) $k_{latency} = 2$ (c) $k_{latency} = 4$

Fig. 7. Utility loss comparison of three sharing mechanisms at Nash equilibria.



(a) $k_{latency} = 1.4$ (b) $k_{latency} = 2$ (c) $k_{latency} = 4$

Fig. 8. Ratio of the number of servers of the edge player over that of the cloud player at Nash equilibria.

When latency requirement is not stringent ($k_{latency} = 2$), Shapley mechanism and Ortmann mechanism have roughly the same utility loss. When tasks have very flexible latency requirements ($k_{latency} = 4$), the utility loss of Shapley mechanism is higher than that of Ortmann mechanism at provider level.

5.4.3. *Numbers of servers at equilibria*

In addition, we have also examined the ratio of the number of the edge player's servers over the number of the cloud player's servers at Nash equilibrium[11]. Fig. 8 shows that when the bandwidth difference between the two players is not big ($k_{bw} = 1$ or 2), the edge player has fewer number of servers at equilibria than the cloud player across all three different revenue sharing mechanisms and all three different levels of latency requirements. This is because the edge player's cost of placing servers in the system is higher than that of the cloud player.

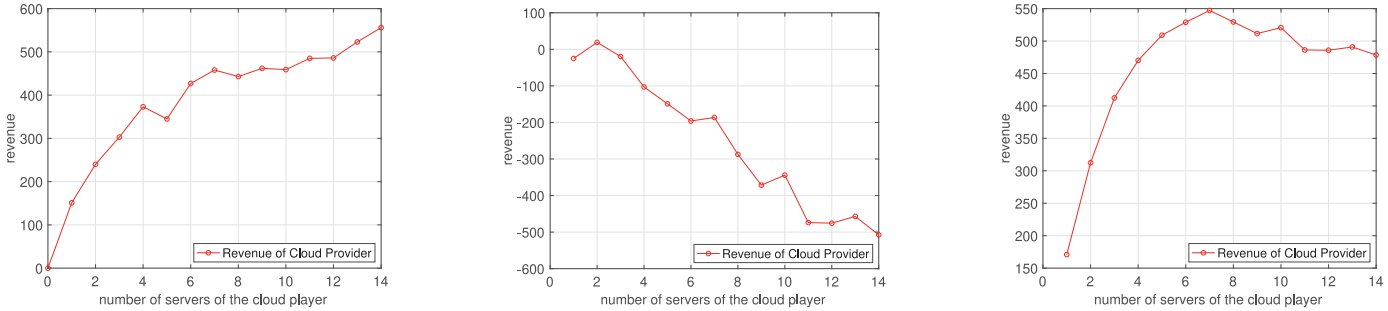
When k_{bw} gets higher ($k_{bw} = 4$) and task latency requirement is stringent or normal ($k_{latency} = 1.4$ or 2), the cloud player will place fewer number of servers (than the edge player) at equilibria under Shapley or Ortmann mechanism. This is because the cloud player's servers are less likely able to meet tasks' deadline requirements, and the two revenue sharing mechanisms discourage the cloud player from putting more servers in the competition. This discouragement leads to a better system performance (i.e., lower system utility loss) than the Direct-contribution-based sharing.

5.4.4. *The case of large bandwidth difference ($k_{bw} = 4$) and stringent latency requirements of tasks ($k_{latency} = 1.4$)*

The disadvantage of Direct-contribution-based sharing mechanism is quite obvious in this case. Fig. 7 (a) shows that its utility loss (20.08%) is significantly higher than that of Shapley's (near 0%) and Ortmann's (6%). We observe a similar pattern later in Fig. 10.

This can be explained through Fig. 9. Under the Direct-contribution-based sharing, the very low bandwidth provider (i.e., the cloud player)

[11] In the case where there are multiple Nash equilibria, the ratio is calculated as the average of those equilibria, similar to the calculation of utility loss.



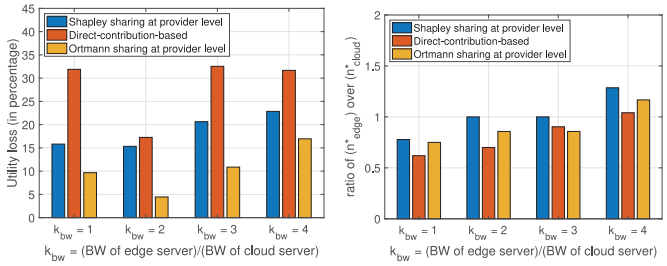
(a) Direct-contribution-based sharing. (b) Shapley sharing at provider level. (c) Ortmann sharing at provider level.

Fig. 9. Revenue of cloud provider with $k_{latency} = 1.4$, $k_{bw} = 5$ and 16 servers of edge provider at Nash equilibria.

still aggressively utilizes many servers in order to gain a high revenue (as it is rewarded directly based on its actual contributions), which leads to a very low overall system efficiency, i.e., low overall system utility at equilibrium states. But Shapley and Ortmann mechanisms discourage such an aggressive behavior of the provider with very low bandwidth, because in this case, Shapley and Ortmann mechanisms give penalty instead of reward to the low bandwidth cloud servers. Specifically, Shapley mechanism will start to assign decreasing or even negative revenue to cloud servers once the number of cloud servers increases over a threshold (which implies that placing in the system more cloud servers with very low bandwidth will bring negative marginal contribution to the system); and similarly, Ortmann mechanism will assign lower and lower revenues to cloud servers, which is shown in Fig. 9. Therefore, compared with Direct-contribution-based mechanism, Shapley and Ortman mechanisms make cloud player (i.e., the provider with lower bandwidth) place fewer number of servers in the system. This is also shown as the higher ratio of the number of edge severs over the number of cloud servers under Shapley and Ortman mechanims in Fig. 8. Our results show that in this case, Shapley and Ortmann mechanisms bring significant more revenue to the system than Direct-contribution-based sharing at Nash equilibria, which means lower utility loss, as shown in Figs. 6 and 7. Our finding suggests that Shapley and Ortmann mechanisms, the two mechanisms that distribute revenue based on marginal contributions instead of directly on actual contributions, can help improve a system's overall utility in the face of the self-interested behavior of providers.

5.5. Similar results from google cloud trace

We have also conducted simulations based on Google cloud trace data [10]. We pre-process the data, as described in Section 5.2. Each simulation presented in this section lasts $T = 1200$ minutes. Here we



(a) (b)

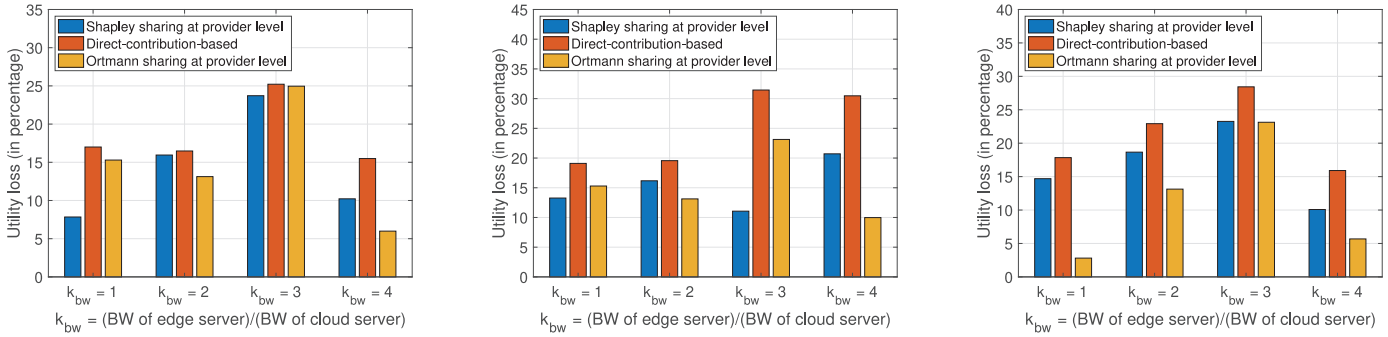
Fig. 10. Utility loss and ratio of the number of servers of the edge player over that of the cloud player at Nash equilibria.

report results when latency requirement level $k_{latency} = 2$, and the transmission bandwidth difference between cloud servers and edge servers being $k_{bw} = 1, 2, 3, 4$. The cost coefficients of edge and cloud player are $\alpha_{edge} = 4$ and $\alpha_{cloud} = 3$ respectively.

Fig. 10 shows our simulation results based on Google data, and these results are consistent with the results of our simulation reported in previous subsections. Both Shapley and Ortmann sharing mechanisms perform better at provider level than at server level. They both have less utility loss at Nash equilibria than Direct-contribution-based sharing.

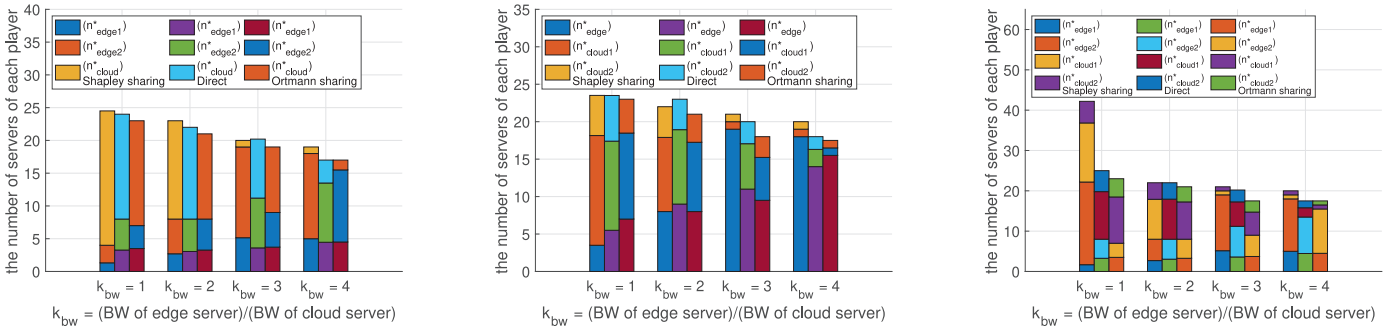
5.6. Results of simulations with more than two players

We have also conducted simulations with more than two players. Here we present the results of the game with the following settings: three players with one cloud player and two edge players, three players with one edge player and two cloud players, and four players with two edge players and two cloud players. For the three settings, we have



(a) 2 edge players, 1 cloud player (b) 1 edge player, 2 cloud players (c) 2 edge players, 2 cloud players

Fig. 11. Utility loss comparison of three sharing mechanisms at Nash equilibria with more than two players and $k_{latency} = 1.4$.



(a) 2 edge players, 1 cloud player (b) 1 edge player, 2 cloud players (c) 2 edge players, 2 cloud players

Fig. 12. The number of servers of each player at Nash equilibria with more than two players and $k_{latency} = 1.4$.

latency requirement level $k_{latency} = 2$, and the transmission bandwidth difference between cloud servers and edge servers being $k_{bw} = 1, 2, 3, 4$. For ease of exposition, we let the servers belonging to different cloud players have the same CPU capacity and the bandwidth of their paths to clients are the same. And we let the servers belonging to different edge players have the same capacities as well. The bandwidth of edge servers' paths to clients are also the same. The cost coefficients of edge and cloud players are $\alpha_{edge} = 4$ and $\alpha_{cloud} = 3$ respectively.

The system utility loss of a system and the number of servers of different players at Nash equilibrium states are shown in Fig. 11 and Fig. 12. Specifically, Figs. 11(a) and 12(a) show the results of the game between one cloud provider and two edge providers. Figs. 11(b) and 12(b) show the game between one edge provider and two cloud providers. Figs. 11(c) and 12(c) show the game between two edge and two cloud providers. The results reported in these figures are consistent with the conclusion in the previous sections. That is, at system equilibrium states, marginal contribution based sharing mechanisms give better system performance than Direct-contribution-based sharing mechanism.

Summary. Marginal contribution based sharing mechanisms, i.e., Shapley sharing and Ortmann sharing, give better system-level performance than Direct-contribution-based sharing mechanism at system equilibrium state, despite that providers game with the system to pursue their self-interested optimization goals.

6. Conclusions and future work

We have proposed a game-theoretic framework to explore the design of revenue sharing mechanism in an Edge-Cloud system in which edge providers and cloud providers compete with each other and game with the system in order to maximize their own utilities. We have shown the existence of Nash equilibrium in the game between providers, and we have found that at equilibrium system state, the revenue sharing mechanism based on marginal contributions of providers can result in significantly better system performance than revenue sharing based directly on actual contributions of providers. We have provided fundamental insights into the design of revenue sharing mechanisms to deal with self-interested providers. For example, under Direct-contribution-based sharing, a provider with very low transmission bandwidth attempts to place as many servers as possible in the system (as it is rewarded directly based on its actual contribution) even when doing so actually hurts the overall system performance. On the other hand, the revenue sharing based on marginal contributions discourage a low bandwidth provider from aggressively offering many servers by setting its revenue share as a decreasing function of its offered resources. In addition, our non-cooperative game theoretic framework and revenue sharing mechanism design can also be adopted by the giant players, such as AWS, Azure, and IBM Cloud, in today's cloud/edge computing market where customers may place their application jobs on the servers/data centers of different major providers, a so-called multi-cloud approach.

For future work, we will conduct further theoretic analysis, study dynamic game playing processes, and conduct large scale experiments of Edge-Cloud systems.

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.

CRediT authorship contribution statement

Zhi Cao: Methodology, Software, Validation, Formal analysis, Investigation, Writing - original draft, Writing - review & editing, Visualization. **Honggang Zhang:** Conceptualization, Methodology, Software, Writing - original draft, Writing - review & editing, Supervision, Funding acquisition, Project administration. **Benyuan Liu:** Conceptualization, Supervision, Writing - review & editing. **Bo Sheng:** Resources, Writing - review & editing.

Acknowledgment

This research was supported in part by NSF grants CNS-1527303 and CNS-1562264. The information reported here does not reflect the position or the policy of the federal government of USA.

Supplementary material

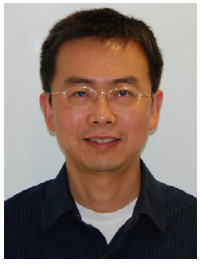
Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.comnet.2020.107286.

References

[1] F. Bonomi, R. Milito, J. Zhu, S. Addepalli, Fog computing and its role in the internet of things, ACM MCC, 2012.
[2] M. Chiang, Fog Networking: An Overview on Research Opportunities (2016) arXiv:1601.00835.
[3] W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu, Edge computing: vision and challenges, IEEE J. Internet Things 3 (2016).
[4] S. Yi, C. Li, Q. Li, A survey of fog computing: concepts, applications and issues, in: Proceedings of the 2015 Workshop on Mobile Big Data, ACM, 2015, pp. 37–42.
[5] Kubernetes, (https://kubernetes.io/). Accessed: 2018-05-05.
[6] docker, (http://www.docker.com). Accessed: 2017-03-10.
[7] Foglamp, (https://foglamp.readthedocs.io/en/latest/). Accessed: 2017-12-11.
[8] L.S. Shapley, A Value for n-Person Games, Contributions to the Theory of Games, Princeton University Press 28 (1953).
[9] K.M. Ortmann, The proportional value for positive cooperative games, Math. Method. Oper. Res. 51 (2000) 235–248.
[10] Google, Google cloud cluster data, (http://googleresearch.blogspot.com/2010/01/google-cluster-data.html).
[11] Y.C. Hu, M. Patel, D. Sabella, N. Sprecher, V. Young, Mobile edge computing: a key technology towards 5g, ETSI White Paper (2015).
[12] P. Liu, D. Willis, S. Banerjee, Paradrop: enabling lightweight multi-tenancy at the network's extreme edge, IEEE/ACM SEC, 2016.

[13] J. Yoon, P. Liu, S. Banerjee, Low-cost video transcoding at the wireless edge, IEEE/ACM SEC, 2016.
[14] R. Mahmud, R. Kotagiri, R. Buyya, Fog computing: a taxonomy, survey and future directions, Internet Everything (2018) 103–130.
[15] M. Mukherjee, L. Shu, D. Wang, Survey of fog computing: fundamental, network applications, and research challenges, IEEE Commun. Surv. Tutor. 20 (3) (2018) 1826–1857.
[16] J. Gedeon, From cell towers to smart street lamps: placing cloudlets on existing urban infrastructures, in: 2018 IEEE/ACM Symposium on Edge Computing (SEC), IEEE/ACM, 2018, pp. 187–202.
[17] Z. Cao, H. Zhang, B. Liu, B. Sheng, A game-theoretic framework for revenue sharing in edge-cloud computing system, in: International Performance Computing and Communications Conference, IEEE, 2018.
[18] T. Zhang, A. Chowdhery, P.V. Bahl, K. Jamieson, S. Banerjee, The design and implementation of a wireless video surveillance system, ACM MobiCom, 2015.
[19] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, M. Satyanarayanan, Towards wearable cognitive assistance, ACM MobiSys, 2014.
[20] Z. Wen, D.L. Quoc, P. Bhatotia, R. Chen, M. Lee, Approxiot: approximate analytics for edge computing, in: 38th IEEE International Conference on Distributed Computing Systems, ICDCS 2018, IEEE, 2018, pp. 411–421.
[21] X. Cao, J. Zhang, H.V. Poor, An optimal auction mechanism for mobile edge caching, in: 38th IEEE International Conference on Distributed Computing Systems, ICDCS 2018, IEEE, 2018, pp. 388–399.
[22] S. Jang, Y. Lee, B. Shin, D. Lee, Application-aware IoT camera virtualization for video analytics edge computing, in: 2018 IEEE/ACM Symposium on Edge Computing (SEC), IEEE/ACM, 2018, pp. 132–144.
[23] J. Wang, Bandwidth-efficient live video analytics for drones via edge computing, in: 2018 IEEE/ACM Symposium on Edge Computing (SEC), IEEE/ACM, 2018, pp. 159–173.
[24] S. Khare, Scalable edge computing for low latency data dissemination in topic-based publish/subscribe, in: 2018 IEEE/ACM Symposium on Edge Computing (SEC), IEEE/ACM, 2018, pp. 214–227.
[25] S. Maheshwari, Scalability and performance evaluation of edge cloud systems for latency constrained applications, in: 2018 IEEE/ACM Symposium on Edge Computing (SEC), IEEE/ACM, 2018, pp. 286–299.
[26] AT&T The cloud comes to you, (http://about.att.com/story/reinventing_the_cloud_through_edge_computing.html). Accessed: 2017-07-27.
[27] A. Samanta, Z. Chang, Adaptive service offloading for revenue maximization in mobile edge computing with delay-constraint, Internet Things J. 6 (2019) 3864–3872.
[28] X. Yu, L. Tang, Competition and cooperation between edge and remote clouds, in: International Conference on Computer and Communications, IEEE, 2018.
[29] R.T. Ma, D.M. Chiu, J. Lui, V. Misra, D. Rubenstein, Internet economics: the use of shapley value for isp settlement, IEEE/ACM Trans. Netw. (TON) 18 (3) (2010) 775–787.
[30] V. Misra, S. Ioannidis, A. Chaintreau, L. Massoulié, Incentivizing peer-assisted services: a fluid shapley value approach, ACM SIGMETRICS Performance Evaluation Review, 38, 2010.
[31] H. Zhang, D. Towsley, W. Gong, TCP connection game: a study on the selfish behavior of TCP users, IEEE ICNP, 2015.
[32] H. Zhang, B. Liu, H. Susanto, G. Xue, T. Sun, Incentive mechanism for proximity-based mobile crowd service systems, IEEE INFOCOM, 2016.
[33] A. Rubinstein, Settling the complexity of computing approximate two-player nash equilibria, in: 57th Annual Symposium on Foundations of Computer Science (FOCS), IEEE, 2016, pp. 258–265.
[34] C. Daskalakis, P.W. Goldberg, C.H. Papadimitriou, The complexity of computing a nash equilibrium, SIAM J. Comput. 39 (2009) 195–259.
[35] X. Chen, X. Deng, S.-H. Teng, Settling the complexity of computing two-player nash equilibria, J. ACM (JACM) 56 (2009) 1–57.
[36] A. Czumaj, M. Fasoulakis, M. Jurdziski, Multi-player approximate nash equilibria, in: Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, ACM, 2017, pp. 1511–1513.
[37] A. Rubinstein, Inapproximability of nash equilibrium, SIAM J. Comput. 47 (2018) 917–959.
[38] T. Basar, G.J. Olsder, Dynamic Noncooperative Game Theory, Academic Press, New York, 1998.
[39] Cplex, (https://www.ibm.com/us-en/marketplace/ibm-ilog-cplex). Accessed: 2017-07-20.
[40] R.B. Myerson, Graphs and cooperation in games, Math. Oper. Res. 2 (3) (1977) 225–229.
[41] H. Susanto, B. Kaushik, B. Liu, B.-G. Kim, Pricing and revenue sharing mechanism for secondary re-distribution of data service for mobile devices, IEEE IPCCC, 2014.
[42] J. Nash, Non-cooperative games, Ann. Math. 54 (1951) 286–295.
[43] Raspberry pi, (https://www.raspberrypi.org). Accessed: 2017-07-20.
[44] The web application messaging protocol, (http://wamp-proto.org). Accessed: 2017-03-10.
[45] G. Bradski, A. Kaehler, Learning opencv: computer vision with the opencv library, " O'Reilly Media, Inc.", 2008.
[46] T.-Y. Lin, Microsoft coco: common objects in context, in: European Conference on Computer Vision, Springer, 2014, pp. 740–755.
[47] R. Johari, J.N. Tsitsiklis, Efficiency loss in a network resource allocation game, Math. Oper. Res. 29 (3) (2004) 407–435.

Zhi Cao received her BS degree and MS degree in Mathematics from the University of Science and Technology Beijing in 2014 and 2016 respectively. She is currently working towards the doctoral degree in the Computer Science Department at University of Massachusetts Boston. Her primary research interests include Edge Computing, Internet of Things and Deep Reinforcement Learning.



Honggang Zhang holds a PhD in Computer Science (2006) from the University of Massachusetts, Amherst, USA. He received his BS degree from the Central South University of China, and his MS degree from Tianjin University of China. He also received an MS degree from Purdue University, West Lafayette, IN, USA. He is currently an Associate Professor of Computer Engineering in the Engineering Department at University of Massachusetts Boston, Boston, MA, USA. His research interests span a wide range of topics in the area of computer networks and distributed systems. His current research focuses primarily on Edge Computing, Internet of Things, and Mobile Computing. He was a recipient of the National Science Foundation (NSF) CAREER Award in 2009.

Benyuan Liu received the BS degree in physics from the University of Science and Technology of China (USTC), the MS degree in physics from Yale University, and the PhD degree in computer science from the University of Massachusetts Amherst. He is currently a faculty member in the Department of Computer Science, University of Massachusetts Lowell. His primary research interests are in the area of application, algorithm design and performance analysis of computer networks. His research has been published in premium computer science conferences and journals, and has been widely reported by many news media, including MIT Technology Review, Wired, CNN, etc. He is a recipient of the NSF CAREER Award.

Bo Sheng is an Associate Professor in the Department of Computer Science at University of Massachusetts Boston. He received his B.S. from Nanjing University, and his Ph.D. from the College of William and Mary, both in Computer Science. His research interests include big data analytics, mobile computing, Internet of Things, wireless networks, and security.