A Game-theoretic Framework for Revenue Sharing in Edge-Cloud Computing System

Zhi Cao*, Honggang Zhang*, Benyuan Liu†, Bo Sheng*
* UMass Boston, Boston, MA. Email: {zhi.cao001,honggang.zhang,bo.sheng}@umb.edu
† UMass Lowell, Lowell, MA. Email: bliu@cs.uml.edu

***Abstract*—We introduce a game-theoretic framework to explore revenue sharing in an Edge-Cloud computing system, in which computing service providers at the edge of the Internet (*edge providers*) and computing service providers at the cloud (*cloud providers*) collectively provide computing resources to clients (e.g., end users or applications) at the edge. Different from traditional cloud computing, the providers in an Edge-Cloud system are independent and self-interested. To achieve high system-level efficiency, the manager of the system adopts a task distribution mechanism to maximize the total revenue received from clients and also adopts a revenue sharing mechanism to split the received revenue among computing servers (and hence service providers). Under those system-level mechanisms, service providers attempt to game with the system in order to maximize their own utilities, by strategically allocating their resources (e.g., computing servers).**

Our framework models the competition among the providers in an Edge-Cloud system as a non-cooperative game. We have shown the existence of Nash equilibrium in the game both theoretically and practically through simulations and experiments on an emulation system that we have developed. We find that revenue sharing mechanisms have a significant impact on the system-level efficiency at Nash equilibria, and surprisingly the revenue sharing mechanism based directly on actual contributions can result in significantly worse system performance than Shapley value sharing mechanism and Ortmann proportional sharing mechanism. Our framework provides an effective economics approach to the understanding and designing of efficient Edge-Cloud computing systems.

I. Introduction

Edge computing [1]–[6] is an emerging computing paradigm that is transforming the landscape of provision and consumption of computing services for a wide range of applications and end users at the edge of the Internet. This paradigm will be particularly helpful to those latency-sensitive and bandwidth-hungry applications brought by Internet of Things (IoT) systems.

In this paper, we are interested in a hybrid system where computing service providers at the edge of the Internet (referred to as *edge providers*, which are close to IoT sensors, mobile devices, and end users) and the providers at the cloud (referred to as *cloud providers*) collectively provide computing services to the mobile clients at the edge. Such a system is referred to as an *Edge-Cloud system*. 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 providers in an Edge-Cloud system are independent and located at various distances away from mobile clients, and they can 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. It adopts a revenue sharing mechanism to fairly split its received revenue among computing servers (and hence service providers). Under any given system-level mechanism, the providers 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.

The self-interested behaviors of those providers might result in an inefficient system with low overall system performance, as their individual self-interested objectives do not collectively align with the system-wide objective. Therefore, it is important to choose a system-level mechanism that will minimize the loss of system-wide efficiency. To that end, we introduce a game-theoretic framework to investigate the impact of revenue sharing mechanisms on an Edge-Cloud system's overall efficiency.

Our major contributions are summarized below.

1) We introduce a game-theoretic framework to investigate an Edge-Cloud computing system of edge providers and cloud providers that offer their computing resources to mobile clients at the edge. Our findings demonstrate that 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 have demonstrated the existence of Nash equilibrium in the game between edge and cloud providers, under three revenue sharing mechanisms, and across a wide range of system/networking settings. We find that when the servers from different providers have different capacities, different revenue sharing mechanisms can result in drastically different system-level utility loss at equilibria when compared with maximum system utility (which is achieved when providers do not game with the system).
3) Our results show that at the Nash equilibria of the game, Direct-contribution-based sharing results in the worst system-level utility. This seemingly counter-intuitive re-

978-1-5386-6808-5/18/$31.00

sult is not surprising, as under Direct-contribution-based sharing, a provider with very low transmission bandwidth keeps placing many servers in the system even if doing so actually hurts the overall system performance. On the other hand, Shapley mechanism gives the least utility loss in most cases, and Ortmann mechanism's utility loss is close to Shapley's. This is because Shapley mechanism and Ortmann mechanism discourage a low bandwidth provider from offering many servers by setting its revenue share as a decreasing function of its number of servers placed in the system.

4) We demonstrate that our framework is a valid and effective economics approach to the understanding and designing of efficient Edge-Cloud computing systems, based on our extensive simulations driven by the empirical data derived from experiments on an emulation system we have developed and from Google cloud trace [7].

In the rest of the paper, we present the architecture of an Edge-Cloud system and give an overview of our game-theoretic framework in Section II. Then in Sections III and IV, we describe task distribution mechanisms and revenue sharing mechanisms. Section V presents our findings via experiments and simulations, and we conclude the paper in Section VI.

II. Edge-Cloud System

In this section, we first discuss background and related work. Then we give an overview of an Edge-Cloud system.

A. Background and Related Work

This paper studies a computing system in the emerging edge computing paradigm, which broadly includes cloudlets, mobile edge computing, fog computing, fog networks, and mobile cloud computing [1]–[6]. Besides the low latency benefit of edge computing, recent research on 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 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 [8], [9]. 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 [10]. 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 [11]–[13].

B. System Architecture Overview

An Edge-Cloud computing system is a hybrid system that integrates both edge computing and cloud computing to provide services to mobile 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 Figure 1. (1) Mobile 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., those computing service providers at the edge and close 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 software component that implements mechanisms/algorithms for various management 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. To make the system scalable, we let each manager be regional, i.e., only responsible for a specific region where the amounts of resources and clients are limited. Clients communicate with and submit their tasks to the system manager through apps on their devices.

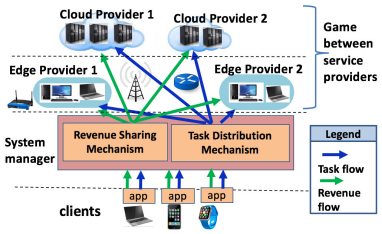


Fig. 1. System architecture. Computing service providers compete with each other under system-level mechanisms: task distribution and revenue sharing.

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). 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 [14], Ortmann proportional sharing [15], and Direct-contribution-based sharing.

C. A Game-theoretic Framework for an Edge-Cloud System

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 and offer service

to the clients in the system, and such a cloud provider is treated equally as other edge providers in the same system.

We assume that the two types of mechanisms work on individual servers, without any consideration of the identities of the owners (i.e., providers) of those servers, as the objectives of those system-level mechanisms are to optimally utilize available *servers* to maximize total received revenue and fairly distribute the revenue[1] among participating *servers*. Those mechanisms 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.

2) The game: We model the competition among service providers in an Edge-Cloud system as a non-cooperative game [16], 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

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

where $v_p(n_p)$ 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. We focus on a linear cost function $f_{cost}(n_p) = \alpha_p n_p$ with $\alpha_p > 0$.

Now we define a Nash equilibrium [16] for the game through a game of two players: an edge provider E and a cloud provider C. Let v_E and v_C denote the revenue received by the edge player and the cloud player respectively. Let M_{opt} and M_{share} denote a task distribution mechanism and a revenue sharing mechanism respectively. Then we know v_E is a function of M_{share} and M_{opt}. Note that M_{opt} is a function of (n_E, n_C).

The edge player and the cloud player attempt to solve the following optimization problems respectively

$$\begin{aligned}\max_{n_E} U_E(n_E, n_C) &= v_E(M_{share}(M_{opt}(n_E, n_C))) - \alpha_E n_E \\ \max_{n_C} U_C(n_C, n_E) &= v_C(M_{share}(M_{opt}(n_E, n_C))) - \alpha_C n_C\end{aligned}$$

A Nash equilibrium [16] of the game is a particular combination of all players' strategies from which a player has no incentive to unilaterally deviate, as any unilateral deviation will not increase its utility. The Nash equilibrium is denoted by $\{n_E^*, n_C^*\}$, where $n_E^* = \text{argmax}_{n_E} U_E(n_E, n_C^*)$ and $n_C^* = \text{argmax}_{n_C} U_C(n_C, n_E^*)$. Note that the definition here can be easily generalized to the definition of a Nash equilibrium of a m-player game: $\{n_i^*\}, \forall i \in \{1, 2, ..., m\}$ with $n_i^* = \text{argmax}_{n_i} U_i(n_i, n_{-i}^*)$ (where $-i$ denotes the set of all players except i).

[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.

III. Mechanism for Distributing Computing Tasks

A. 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. A task has a value, which can be regarded as the payment that the task's owner 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*. We present an optimization formulation for the case where tasks arrive in a batch to illustrate the characteristic of task distribution, and then we present a greedy algorithm to address a practical dynamic task arrival setting.

B. Optimal Task Distribution Formulation

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 is the same as the order of their arrivals, and 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 $i = 1, ..., N_J$, and servers are indexed by $j = 1, ..., N_S$. Let x_{ij} denote the assignment of task i to server j. Then $x_{ij} = 1$ represents 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 server (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.

$$\max_{x_{ij}} \quad \sum_{j=1}^{N_S} \sum_{i=1}^{N_J} v_i x_{ij} \tag{2}$$

$$s.t. \quad 0 \leqslant \sum_{j=1}^{N_S} x_{ij} \leqslant 1, \quad \forall i \tag{3}$$

$$x_{ij} \in \{0, 1\}, \forall i; \forall j \tag{4}$$

$$x_{ij} d_{ij} + \sum_{k=1}^{i-1} q_{ijk} d_{kj} \leqslant L_i, \quad \forall i, \forall j \tag{5}$$

$$x_{ij} = 0 \rightarrow q_{ijk} = 0, \forall i, \forall k \in \{1, ..., i-1\}; \forall j \tag{6}$$

$$x_{ij} = 1 \rightarrow q_{ijk} = x_{kj}, \forall i, \forall k \in \{1, ..., i-1\}; \forall j \tag{7}$$

$$q_{ijk} \in \{0, 1\}, \forall i, \forall k \in \{1, ..., i-1\}; \forall j \tag{8}$$

where (3) and (4) say that a task can be assigned to at most one server. The three constraints (5), (6), and (7) collectively say 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 (5). The two constraints (6) and (7) indicate that q_{ijk} is equivalent to $x_{ij}x_{kj}$. Note that (6) and (7) are called indicator constraints in CPLEX solver [17]. 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 $k \in \{1, ..., i-1\}$) 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, (6) says that when task i is not assigned to server j, its queuing delay constraint (5) on server j should be removed.

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. The idea of the algorithm is: whenever a server is available, it should be given 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.

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 value among all tasks that can be finished before 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**

IV. Mechanisms for revenue sharing

In this section, we investigate the following three revenue sharing mechanisms: (1) Shapley value [14]; (2) A proportional sharing mechanism proposed by Ortmann [15], 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.

A. Shapley-value revenue sharing mechanism

Shapley value [14] is a well-known revenue sharing mechanism. For an Edge-Cloud system, Shapley value defines 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:

$$\phi_i(\mathbb{N}_S) = \frac{1}{N_S!} \sum_{\mathbb{S} \subseteq \mathbb{N}_S - \{i\}} |\mathbb{S}|!(N_S - |\mathbb{S}| - 1)!\big(v(\mathbb{S} \cup \{i\}) - v(\mathbb{S})\big) \tag{9}$$

This revenue distribution mechanism satisfies the following desired properties [11], [14], [18], [19]: fairness or balanced contribution, symmetry, efficiency and dummy.

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 Equation (9). 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 are identical 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 $N_k = |\mathbb{N}_S^k|$.

2) Time complexity of Algorithm 2: Consider a system of two providers, in which an edge provider has a set of servers $\{e_1, e_2, \cdots, e_{N_1}\}$, and a cloud provider has a set of servers $\{c_1, c_2, \cdots, c_{N_2}\}$. Note 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 provider (e.g., AT&T [10])[2]. All servers of the same type (i.e., in the same group) are equivalent so that they receive the same Shapley value. Suppose we would like to apply Algorithm 2 to calculate the Shapley value of a particular edge server

[2]It is not practical 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.

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 $\mathbb{N}_S^k$ is the set of servers of provider k, $N_k = |\mathbb{N}_S^k|$, $N = |\mathbb{N}_S|$, $k = 1, 2, ..., m$.

1: **for** $j = 1, 2, ..., m$ **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 $(n_1, n_2, ..., n_{j-1}, n_{j+1}, ..., n_m)$, where each number n_k (with $k \in \{1, 2, ..., j-1, j+1, ..., m\}$) varies from 0 to N_k.
5: For each $(n_1, n_2, ..., n_{j-1}, n_j, n_{j+1}, ..., n_m)$, 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: $V_1 = V[n_1][n_2]...[n_{j-1}][n_j][n_{j+1}]...[n_m]$, i.e., the value of the set that contains n_k servers from provider k, with $k \in \{1, 2, ..., m\}$.
8: $V_2 = V[n_1][n_2]...[n_{j-1}][n_j + 1][n_{j+1}]...[n_m]$. (Similar to V_1, except that $n_j + 1$ is for provider j).
9: Calculate $C_{coeff} = C_{N_1}^{n_1} \cdot C_{N_2}^{n_2} \cdots C_{N_{j-1}}^{n_{j-1}} \cdot C_{N_j - 1}^{n_j} \cdot C_{N_{j+1}}^{n_{j+1}} \cdots C_{N_m}^{n_m}$. ($N_j - 1$ is for provider j).
10: Let $S = \sum_{k=1}^{m} n_k$.
11: Calculate $V_{n_1, n_2, ... n_{j-1}, n_j, n_{j+1} ... n_m}^{inc} = \frac{S!}{N!}(N - S - 1)! \cdot C_{coeff} \cdot (V_2 - V_1)$.
12: Increase V_j^s by $V_{n_1, n_2, ... n_{j-1}, n_j, n_{j+1} ... n_m}^{inc}$.
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 $j = 1, 2, ..., m$.

e_i. For any two subsets S_1 and S_2 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 numbers of edge/cloud servers in S_1 and S_2 are equal respectively. 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 (9), 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 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 $O(N_1 N_2 \cdots N_m)$.

B. Direct-contribution-based and Ortmann proportional sharing mechanisms

The idea of Direct-contribution-based sharing is as follows. A server is rewarded with a share of revenue that is proportional to the actual contribution that it has made. In the case where a system manager distributes all of its received revenue among participating servers, 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 [15] is a sharing mechanism that is similar to Shapley value, in the sense that it also relies on the calculation of some marginal contribution of a server, instead of relying directly on the server's actual contribution. [15] gives a formal definition of Ortmann proportional sharing. Note that Ortmann proportional sharing's balanced contribution property is in the form of ratio equality, instead of the difference equality of Shapley's.

C. Equilibrium state of an Edge-Cloud system

Recall that we model the competition among multiple service providers as a non-cooperative game. We have derived the following theorem. Due to space limitations, we put its proof in [20].

Theorem 1 *The game of multiple resource providers in an Edge-Cloud system admits a pure strategy Nash equilibrium.*

V. Impact of Revenue Sharing Mechanisms

We investigate in this section the impact of revenue sharing mechanisms on the performance of an Edge-Cloud system, through a combination of emulations and simulations[3].

A. 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 [21] and Ubuntu laptops), a system manager (a distributed software component), and a pool of servers. 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. The edge servers have higher bandwidth and shorter propagation delays than the cloud servers. Once a server receives a task, a Docker container [22] will be launched on the server to process the task. Once the task is completed, the server will notify the manager and sends the result of the task back to the edge client. The communication between the clients, the manager, and the servers is through Web Application Messaging Protocol (WAMP) [23].

B. Determining Simulation Parameters

We utilize image processing in our simulations to investigate the impact of revenue sharing mechanisms. We next discuss how to derive system parameters used in our simulations.

[3]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.

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. Then the system assigns the task to a server. The server launches a Docker container [22] to process the images using OpenCV [24].

A simulation run is characterized by a group of system parameters: $\{T, \mathbb{N}_J(T), \mathbb{N}_S, \lambda, f_{delay}, k_{latency}, k_{bw}\}$, where T is the system time duration that 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 in range $[1, 20]$ MB. The average task size 10 MB roughly corresponds to a batch of 6 images with a regular image size about 1.6 MB. The average task size also roughly corresponds to a collection of 63 images from the Microsoft COCO image dataset [25] with an average image size of 159 KB. The value of a task is a number chosen uniformly at random from range $[1, 5]$.

We assume that tasks arrive at the system in a Poisson process with rate λ (number of tasks per minute). We choose $\lambda = 40$ in our simulations, which is the average job arrival rate in Google cloud trace [7]. Since mobile edge computing (MEC) is an emerging paradigm, there is no publicly available measurement data of a real MEC system, to the best of our knowledge. Thus, we use Google trace here instead.

The completion time f_{delay} of a task on a server depends on the server's CPU and bandwidth. Let t_i (sec) denote the computation time of processing a batch of images of s_i MB. Through our experiments, a linear regression analysis shows that the computation time of processing a batch of images is linearly proportional to the size of the batch, $t_i = 2.6s_i$. We will utilize this function to calculate the computation time of a task in our simulations.

A task i has a latency requirement, denoted by L_i (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. Assume that the average upload bandwidth of the paths from clients to servers is B Mbps. Then, $L_{i,avg} = 2.6s_i + (8s_i/B) + d_{prop}$ 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 or a cloud server be 24 or $24/k_{bw}$ Mbps respectively. The bandwidth factor $k_{bw} = 1, 2, 3, 4$ models the reality, where a cloud server usually has lower bandwidth to clients than an edge server. Then, we let L_i be a number chosen uniformly at random from range $[L_{i,avg}, k_{latency}L_{i,avg}]$, with the latency factor $k_{latency} \geq 1$. The rationale of choosing such a latency requirement is that, 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.

C. Existence of Nash equilibrium and efficiency loss metric

We have conducted extensive simulations of dynamic task arrivals via an event-driven simulator that we developed in Python. Also we have applied CPLEX [17] to solve the optimization problem for tasks arriving in a batch. Our results have demonstrated the existence of Nash equilibrium. In this subsection, we first illustrate the structure of the game between two players, through our results on the case of batch task arrivals. Then we introduce a metric to measure the efficiency of an Edge-Cloud system. Finally, we discuss the impact of revenue sharing via simulations of dynamic task arrivals.

Recall that for a batch arrival of tasks, the solution of optimization problem (2) gives a system manager the maximum total received value or revenue, and then the manager utilizes a revenue sharing mechanism to split the revenue among its servers.

According to our experiments, the revenue received by a provider is an increasing and concave function of its number of servers. Its utility first increases and then drops as a function of its number of servers. Based on the utilities of the two players, we derive their best response curves [12], [16], and we draw them in Figure 2. 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. Figure 2 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.

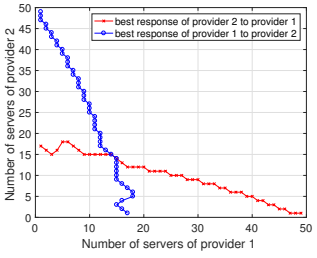


Fig. 2. Best response curves of two providers in a symmetric game.

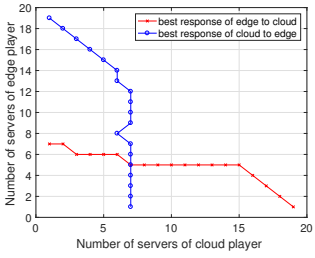


Fig. 3. Best responses of two providers in an asymmetric game.

Note that a typical metric to measure a system's performance at a Nash equilibrium is efficiency loss [12], [26], 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

$$U_{loss} = \frac{\left(\max_{\{n_p\}} \sum_p U_p(n_p)\right) - \left(\sum_p U_p^{NE}(n_p^{NE})\right)}{\max_{\{n_p\}} \sum_p U_p(n_p)} \quad (10)$$

where 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[4]. 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[5]. 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). Figure 3 shows the existence of Nash equilibrium. Note the difference of the settings in Figures 2 and 3. The servers in Figure 2 are all of the same type (edge servers), and the servers in Figure 3 are of two different types (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 Figure 2 is greater than the total number of servers in Figure 3.

D. Impact of Revenue Sharing on System Performance

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 the worst system-level efficiency than Shapley and Ortmann mechanisms*. 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 Figures 4 and 5. 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 [7] are similar to the results presented in this subsection.

1) Utility loss: We observe from Figure 4 that when the transmission bandwidth difference between cloud servers and edge servers is small (i.e., $k_{bw} = 1$ or 2), the losses of system-level utility are roughly the same for all three different revenue sharing mechanisms and across different latency requirement levels of tasks ($k_{latency} = 1.4, 2, 4$)[6].

However, when the cloud player's transmission bandwidth is significantly lower than that of the edge player ($k_{bw} = 4$), Figure 4 shows that Direct-contribution-based sharing gives the worst utility loss, across different latency requirement levels ($k_{latency}$) of tasks. In addition, when tasks have very stringent latency requirement (i.e., $k_{latency} = 1.4$), Shapley mechanism gives the lowest utility loss (only 1.98%). When latency requirement becomes less 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, but still they both are lower than that of Direct-contribution-based mechanism.

2) 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 a Nash equilibrium[7]. Figure 5 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-level performance (i.e., low system utility loss) than the Direct-contribution-based sharing, as shown in Figure 4 (a) and (b).

3) 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. Figure 4 (a) shows that its utility loss (20.08%) is significantly higher than that of Shapley's (1.98%) and Ortmann's (6%).

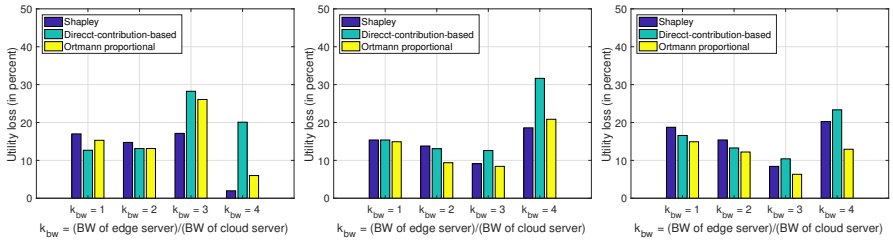
This is because that under the Direct-contribution-based sharing, the very low bandwidth provider (i.e., the cloud player) still aggressively utilizes many servers in order to gain a high revenue (as they are rewarded directly based on their actual contributions), which leads to a very low overall system efficiency. But Shapley and Ortmann mechanisms discourage such an aggressive behavior of the provider with very low bandwidth, because 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 the addition of a cloud server with very low bandwidth will bring negative marginal contribution to the system); and similarly, Ortmann

[4]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 the providers' individual utilities.

[5]In some games, there are two Nash equilibria and they are very close to each other, similar to the case shown in Figure 2. 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.

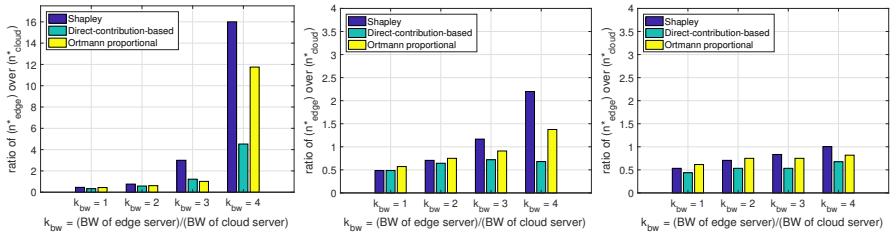
[6]If $k_{latency} = 1.4$, then $1.4L_{i,avg}$ is the amount of time to complete task i (without queuing delay) on the server with the lowest bandwidth.

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



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

Fig. 4. Utility loss (compared with the maximum utility) at Nash equilibria.



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

Fig. 5. Ratio of the number of servers of the edge player over that of the cloud player at Nash equilibria.

mechanism will assign lower and lower revenues to cloud servers. Our results also show that in this case, Shapley and Ortmann mechanisms bring significant more revenue to the system than Direct-contribution-based sharing at Nash equilibria. Our finding suggests that distributing 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.

VI. Conclusions and Future Work

We have proposed a game-theoretic framework to investigate the impact of revenue sharing mechanisms on the performance of 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 found that the revenue sharing based directly on actual contributions of servers can result in significantly worse system-level performance than Shapley value and Ortmann proportional sharing mechanisms at the Nash equilibria of the game between providers. For future work, we will conduct further theoretic analysis, study dynamic game playing processes, and conduct large scale experiments of Edge-Cloud systems.

Acknowledgements. 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.

References

[1] F. Bonomi and *et al*, "Fog computing and its role in the internet of things," in *ACM MCC 2012*.

[2] Y. C. Hu and *et al*, "Mobile edge computing: A key technology towards 5g," *ETSI White Paper*, 2015.

[3] W. Shi and *et al*, "Edge computing: Vision and challenges," *IEEE Journal of Internet of Things*, vol. 3, 2016.

[4] S. Yi, C. Li, and 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] P. Liu, D. Willis, and S. Banerjee, "Paradrop: Enabling lightweight multi-tenancy at the network's extreme edge," in *IEEE/ACM SEC 2016*.

[6] J. Yoon, P. Liu, and S. Banerjee, "Low-cost video transcoding at the wireless edge," in *IEEE/ACM SEC*, 2016.

[7] Google. Google cloud cluster data. http://googleresearch.blogspot.com/2010/01/google-cluster-data.html.

[8] T. Zhang and *et al*, "The design and implementation of a wireless video surveillance system," in *ACM MobiCom 2015*.

[9] K. Ha and *et al*, "Towards wearable cognitive assistance," in *ACM MobiSys 2014*.

[10] $AT\&T$. The cloud comes to you. http://about.att.com/story/reinventing_the_cloud_through_edge_computing.html. Accessed: 2017-07-27.

[11] V. Misra, S. Ioannidis, A. Chaintreau, and L. Massoulié, "Incentivizing peer-assisted services: a fluid shapley value approach," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 38, no. 1, 2010.

[12] H. Zhang, D. Towsley, and W. Gong, "TCP connection game: A study on the selfish behavior of TCP users," in *IEEE ICNP 2005*.

[13] H. Zhang, B. Liu, H. Susanto, G. Xue, and T. Sun, "Incentive mechanism for proximity-based mobile crowd service systems," in *IEEE INFOCOM*, 2016.

[14] L. S. Shapley, "A value for n-person games," *Contributions to the Theory of Games, Princeton University Press*, vol. 28, 1953.

[15] K. M. Ortmann, "The proportional value for positive cooperative games," *Math. Methods of Operations Research*, vol. 51, pp. 235–248, 2000.

[16] T. Basar and G. J. Olsder, *Dynamic Noncooperative Game Theory*. New York: Academic Press, 1998.

[17] Cplex. https://www.ibm.com/us-en/marketplace/ibm-ilog-cplex. Accessed: 2017-07-20.

[18] R. B. Myerson, "Graphs and cooperation in games," *Mathematics of operations research*, vol. 2, no. 3, pp. 225–229, 1977.

[19] H. Susanto, B. Kaushik, B. Liu, and B.-G. Kim, "Pricing and revenue sharing mechanism for secondary re-distribution of data service for mobile devices," in *IEEE IPCCC 2014*.

[20] Z. Cao, H. Zhang, B. Liu, and B. Sheng, "A game-theoretic framework for revenue sharing in edge-cloud computing system (extended version)," *arXiv preprint arXiv:1711.10102*, 2017.

[21] Raspberry pi. https://www.raspberrypi.org. Accessed: 2017-07-20.

[22] docker. http://www.docker.com. Accessed: 2017-03-10.

[23] The web application messaging protocol. http://wamp-proto.org. Accessed: 2017-03-10.

[24] G. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008.

[25] T.-Y. Lin and *et al.*, "Microsoft coco: Common objects in context," in *European Conference on Computer Vision*. Springer, 2014, pp. 740–755.

[26] R. Johari and J. N. Tsitsiklis, "Efficiency loss in a network resource allocation game," *Mathematics of Operations Research*, vol. 29, no. 3, pp. 407–435, 2004.