Multi-server Multi-user Game at Edges for Heterogeneous Video Analytics

Yu Chen†, Sheng Zhang*†, Yibo Jin*†, Zhuzhong Qian†, and Sanglu Lu†
†State Key Lab. for Novel Software Technology, Nanjing University, P.R. China
Email: sheng@nju.edu.cn

***Abstract*—In past years, artificial intelligence related services and applications have boomed, which require high computation, high bandwidth and low latency. Edge computing is regarded as an appropriate solution for them, especially video analytics. In this paper, we study the multi-server multi-user heterogeneous video analytics offloading problem, where users select appropriate edge servers and then offload their raw video data to the servers for essential analytics. To deal with the cooperation and conflicts among users and get a stable situation where each user has no incentive to change the offloading decision unilaterally, we formulate the video analytics offloading problem as a multi-player game. Based on the goal of minimizing the overall delay, we design the potential optimal server selection strategy and then propose a game theory-based algorithm, through which the Nash equilibrium can be reached. Furthermore, we analyze its near-optimal performance via rigorous proof. Finally, extensive trace-driven experiments show that our method improves the overall delay by 48% on average, compared with other algorithms.**

I. Introduction

With the advent of smart devices and a host of new applications, network traffic is growing rapidly. Due to the high transmission delay and heavy loads on the backhaul links, traditional centralized network architectures cannot satisfy the requirements of users [1]. Edge computing is a new emerging paradigm, and it allows the data produced by end devices to be processed at the edge of networks, instead of sending it to the cloud or data center along long routes. Meanwhile, the artificial intelligence (AI) services and applications based on deep learning have boomed in recent years. Among them, video analytics has been envisioned as a killer application for edge computing [2, 3]. Most of the video analytics applications running on edge servers process video data to detect some specific objects, including missing children, abandoned luggage, causing-trouble vehicles, etc. In general, video analytics tasks collect numerous high-definition videos and require high computation, high bandwidth and low latency. Thus, edge computing is regarded as an appropriate solution to satisfy these strict requirements.

In the edge computing environment, there exist multiple edge servers with diverse capacities (e.g., computation capacity and memory), and they are physically distributed in different locations, which causes different data transmission delay. A large number of users offload their video data to the edge servers for video analytics services, as shown in Fig. 1. Some video data can be divided into smaller units

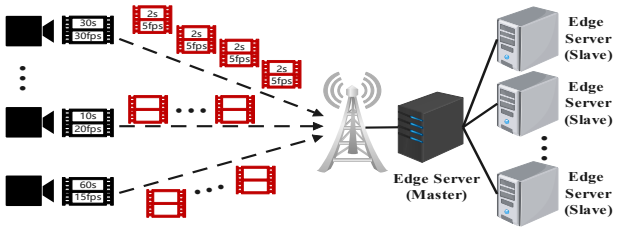


Fig. 1. Multi-server Multi-user Video Analytics Task Offloading.

based on the length and configuration (i.e., frame rate and resolution) [4, 5], and then the video units are separately analyzed through video analytics services. It is challenging for users to select appropriate edge servers and offload their video data (or video units) to them for computation. Edge servers have limited computation capacity and transmission bandwidth, and improper offloading decisions may overload some edge servers and cause the wastage of CPU cycles. For example, if too much video data is allocated to the same edge server, the processing time will be significantly prolonged as a result of the server overloading. On the other hand, if too little video data is allocated to an edge server, it will result in the low utilization of computational resources on the server.

From the user's point of view, the objective is to minimize the overall delay of computation and transmission. Each user needs to determine where to offload the video data (e.g., one edge server or multiple edge servers), how to allocate video units among edge servers, and so on. Usually, the edge network resources are limited, and each user interacts with others to get the stable allocation of network resources and achieve the goal of minimizing the computation and transmission delay. At this point, we apply the method of game theory (GT) to improve the utilization of resources. It helps to analyze the interactions among multiple independent and self-interested players, and then design a system where no player has an incentive to deviate unilaterally [6].

In this paper, we study the multi-server multi-user heterogeneous video analytics task offloading problem, where users select appropriate edge servers and offload their video data to the servers for video analytics. We first present the approach of dividing the video data into smaller units based on the length and frame rate. Then we formulate both of the computation and communication models and represent the overall delay when users offload their video units to the selected servers for computation. In order to achieve the goal of minimizing the overall delay and obtaining a stable situation where no user has an incentive to change its offloading decision unilaterally, we formulate the multi-server multi-user heterogeneous video

*The Corresponding Authors are Sheng Zhang (sheng@nju.edu.cn) and Yibo Jin (yibo.jin@smail.nju.edu.cn). This work was supported in part by NSFC (61872175, 61832008), and Collaborative Innovation Center of Novel Software Technology and Industrialization.

analytics task offloading problem as a multi-player game. To tackle the game and reach the Nash equilibrium, we study the general distributed scenario where each user's video units can be separately offloaded to multiple servers. Based on the GT-based potential optimal server selection algorithm (**Alg. 2**) and the notion of *cut-off value*, the key factor of minority game, we propose the GT-based video unit allocation algorithm (**Alg. 1**). Via rigorous proofs, we analyze its near-optimal performance and prove that the Nash Equilibrium (NE) can be reached. Furthermore, extensive trace-driven experiments using the videos from the AI City Datasets 2019 [7] and the object detector YOLOv3 [8] confirm the performance of our proposed algorithm. Compared with other algorithms, our method improves the overall delay by 48% on average.

II. Related Work

We summarize the prior studies by the following categories and highlight their shortcomings compared with our work.

Task Offloading at Edges: Gao *et al.* [9] developed a joint design of task partitioning and offloading for a DNN-task enabled MEC network that consisted of a single server and multiple mobile devices. Yan *et al.* [10] proposed the optimal task offloading policy and resource allocation that minimized the weighted sum of energy consumption and task execution time under the task-dependency model. Wu *et al.* [11] studied a blockchain scenario where edge computing and cloud computing collaborated toward secure task offloading. These works focus on the task offloading problem at edges, but fail to consider the video analytics applications.

Video Analytics at Edges: NoScope [12] was designed based on a difference detector that highlighted temporal differences across frames to speed up video analysis in edge computing environments. Vantage [13] was presented as a live-streaming upload solution, which used the selective quality enhancing retransmission instead of real-time frames to improve the quality of experience of time-shifted viewers. Ren *et al.* [14] presented an algorithm for multi-user video compression and offloading in edge computing which minimized the latency in local, edge and cloud compression. These works study the video analytics configuration adaptation and task offloading at edges. However, almost no work applies GT-based methods to tackle them for the stable network situation.

Game Theory for Edge Provisioning: Hu *et al.* [15] formulated the heterogeneous task offloading problem as the minority game, and the players ending up in the minority won. Zheng *et al.* [16] proposed a performance metric to evaluate user's quality of experience, and a QoE-oriented resource allocation problem was modeled as a local cooperation game. Zhan *et al.* [17] designed a decentralized offloading game in which each user decided the portion of its task offloaded to the edge server. However, these works fail to consider the method of dividing the video data into smaller units upon length and frame rate for video analytics task offloading at edges.

III. Problem Formulation

In this section, we present the system model and formulate the considered multi-server multi-user heterogeneous video analytics task offloading problem as a multi-player game.

A. System Model

Edge Network: At the network edge, we consider N edge users, denoted as $\boldsymbol{\mathcal{N}} = \{1, 2, ..., N\}$. The edge servers consist of one master server and M slave servers, denoted as $\boldsymbol{\mathcal{M}} = \{1, 2, ..., M\}$. The master edge server is the entry point of task offloading from edge users, and the slave edge servers process the video analytics tasks. Video analytics applications are deployed on the edge servers, and users offload their video analytics tasks to them for computation, as shown in Fig. 1.

Video Segmentation: Based on the length and configuration (i.e., frame rate and resolution), some video data can be divided into smaller units, and then each of the video units is analyzed through the video analytics applications. For example, in object detection, the task is to find some object (e.g., a lost wallet) in the offloaded video data. The length of the video is 600 seconds, and the frame rate is 30 frames per second (fps). We can divide the video data into smaller units, of which the length and frame rate are 2 seconds and 5 fps, respectively. After that, we can apply the object detector to the $\frac{600s \cdot 30fps}{2s \cdot 5fps} = 1800$ video units in parallel.

More generally, each user n has a video analytics task $T_n = \langle l_n, f_n, r_n \rangle$, where the length, frame rate and resolution are denoted as l_n, f_n and r_n, respectively. In some video analytics applications, the video data is captured from the surveillance cameras [4], and the video resolution is fixed. Besides, the size of the video frame input into convolutional neural networks, which is used for video analysis, is usually set to be constant. Thus, in this paper, we assume that the resolutions of all users' video data are the same constant R (i.e., $\forall n \in \boldsymbol{\mathcal{N}}, r_n = R$), and we divide the video data into smaller units upon the length and frame rate. Then we can calculate the number of video units for the user n as $s_n = l_n f_n / L_u F_u$, where L_u is set to a common divisor of all users' video lengths, and F_u is a common divisor of all users' video frame rates. It is worth mentioning that we can always obtain the proper values of L_u and F_u (e.g., L_u equals 1 second and F_u equals 1 fps).

Task Offloading: When user n offloads its video units to edge servers for computation, the task offloading decisions of user n are denoted as $\boldsymbol{x}_n = [x_{n,1}, x_{n,2}, ..., x_{n,M}]^{\mathsf{T}}$, where $x_{n,m}$ represents the number of video units offloaded to edge server m by user n. For each edge server $m \in \boldsymbol{\mathcal{M}}$, $x_{n,m}$ is a non-negative integer, and we have $\sum_{m \in \boldsymbol{\mathcal{M}}} x_{n,m} = s_n$.

Computation Model: Similar to the previous work [4], videos are divided into smaller units, and the computation requirement of each video unit is denoted as C_u [CPU cycles]. Furthermore, we use virtual parallel processing [18] to support the processing of multiple tasks. Then the computation delay on the edge server m can be calculated as

$$\tau_m = \sum\nolimits_{n \in \boldsymbol{\mathcal{N}}} x_{n,m} C_u / Cap_m, \tag{1}$$

where we use Cap_m [CPU cycles per second] to represent the computation capacity of the edge server m. Thus, we calculate the overall computation delay for user n as

$$\mathcal{D}_n^{comp} = \max\nolimits_{m \in \boldsymbol{\mathcal{M}}, x_{n,m} \neq 0} \{\tau_m\}. \tag{2}$$

From the above equations, we observe that when user n offloads all of its video units to only one edge server m, the

overall computation delay just depends on the computation delay on that server. On the contrary, when the video units are offloaded to multiple edge servers, the overall computation delay is the maximum among the computation delays on those servers. Therefore, it is challenging for each user to make and adjust its offloading decision corresponding to other users' so that the overall computation delay can be minimized.

Communication Model: We calculate user n's uplink data rate $r_{n,0}$ [bits per second] of video offloading to master edge server, according to the Shannon-Hartley formula [18], as

$$r_{n,0} = W log_2(1 + P_n H_{n,0}/\sigma^2), \tag{3}$$

where W represents the channel bandwidth and P_n represents user n's transmission power, which can be determined by some power control algorithms [18]. Besides, $H_{n,0}$ denotes the channel gain between user n and master edge server, and σ^2 is the background noise variance.

Similar to the existing work [12–14], the time overhead for the edge servers to send back the computation outcome is neglected, due to the fact that in most of video analytics applications, the size of computation outcome is much smaller than the offloaded video data size. Thus, when user n offloads its s_n video units to the appropriate slave edge servers through master edge server, the transmission delay is calculated as

$$\mathcal{D}_n^{comm} = s_n D_u / r_{n,0}, \tag{4}$$

where D_u [bits] represents the data size of each video unit.

B. Game Formulation

We formulate the multi-server multi-user heterogeneous video analytics task offloading problem as a multi-player game, which is denoted as $\mathcal{G} = \langle \boldsymbol{\mathcal{N}}, \boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{U}} \rangle$, where the user set $\boldsymbol{\mathcal{N}}$ is regarded as the player set. The set of all players' strategy spaces $\boldsymbol{\mathcal{S}}$ is denoted as $\{\boldsymbol{\mathcal{S}}_1, \boldsymbol{\mathcal{S}}_2, ..., \boldsymbol{\mathcal{S}}_N\}$, where $\boldsymbol{\mathcal{S}}_n$ represents the strategy space of player n. One decision $\boldsymbol{x}_n = [x_{n,1}, x_{n,2}, ..., x_{n,M}]^{\mathsf{T}}$ can also denote a task offloading strategy for each player n, and $\boldsymbol{\mathcal{S}}_n$ is the union of all strategies that user n can choose. We use $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_N)$ to represent the task offloading strategies of all players, and $\boldsymbol{x}_{-n} = (\boldsymbol{x}_1, ..., \boldsymbol{x}_{n-1}, \boldsymbol{x}_{n+1}, ..., \boldsymbol{x}_N)$ denotes all of other players' offloading strategies except player n. All of players' utility functions are denoted as a set $\boldsymbol{\mathcal{U}} = \{U_n(\boldsymbol{x}_n, \boldsymbol{x}_{-n})\}_{n \in \boldsymbol{\mathcal{N}}}$. Based on the overall delay of computation and transmission, we construct the utility function for each player n in $\mathcal{G}$ as

$$U_n(\boldsymbol{x}_n, \boldsymbol{x}_{-n}) = 1/(\mathcal{D}_n^{comp} + \mathbb{E}_n[\mathcal{D}_n^{comm}]) \tag{5a}$$

$$= 1/(\max_{m \in \boldsymbol{\mathcal{M}}, x_{n,m} \neq 0}\{\tau_m + \varphi\}), \tag{5b}$$

where we use $\varphi = \mathbb{E}_n[\mathcal{D}_n^{comm}]$ to represent the expected value of transmission delay for all players [15]. It is noted that the overall delay of computation and transmission for each user can be minimized if the user maximizes its utility function.

It is challenging for each player to select the optimal strategy to maximize its utility function and win the game, due to the fact that the payoff of each player depends on not only its own strategy but also others'. To deal with the cooperation and conflicts among players and get a stable situation where each player has no incentive to change the task offloading strategy unilaterally, we need to design the algorithm to achieve the Nash equilibrium defined as follows.

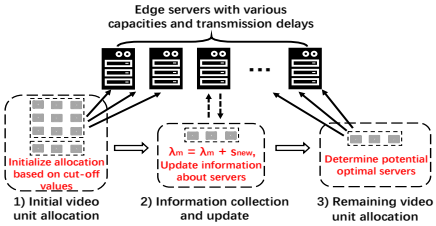


Fig. 2. Three main procedures of Alg. 1.

Definition 1. (**Nash Equilibrium**) For each player n, the strategy set $\boldsymbol{x}^* = (\boldsymbol{x}_1^*, \boldsymbol{x}_2^*, ..., \boldsymbol{x}_N^*)$ constitutes a Nash equilibrium in the game model $\mathcal{G}$ if and only if the individual utility function for each player n cannot be improved by changing its own task offloading strategy unilaterally, i.e.,

$$\forall n \in \boldsymbol{\mathcal{N}}, \forall \boldsymbol{x}_n \in \boldsymbol{\mathcal{S}}_n, U_n(\boldsymbol{x}_n^*, \boldsymbol{x}_{-n}^*) \geq U_n(\boldsymbol{x}_n, \boldsymbol{x}_{-n}^*). \tag{6}$$

The Nash equilibrium has a self-stability property such that users at the equilibrium can obtain mutually satisfactory solutions. Through achieving the Nash equilibrium, we can get a stable situation where each user has no incentive to change its video analytics task offloading decision unilaterally.

IV. Game Theory-based Algorithm Design

In this section, we investigate the general distributed scenario where each user's video units can be separately offloaded to multiple servers. We first introduce the concept of minority game with a key factor, *cut-off value*. After that, we design the GT-based video unit allocation algorithm (**Alg. 1**), which invokes the potential optimal server selection algorithm (**Alg. 2**) to tackle the considered problem in distributed scenarios.

A. Preliminary of Minority Game

As we know, the minority game [6], which is also referred to as the El Faro Bar problem, is a representative resource allocation problem. We first introduce the concept of *cut-off value*, the key factor of the minority game, which explains how players determine the strategies. For example, in a simplified version of the El Faro Bar problem, 100 people decide to go to the town bar for entertainment, and they have 2 identical bars A and B as candidates. The capacity of each bar is limited, such as the limited room space. Thus, it will be wiser for the 100 people to choose the bar with fewer people because they can enjoy a higher quality service and a more elegant environment. Then the cut-off value for the bar A or B in the problem is calculated as $100/2 = 50$. If the number of people in the bar A is smaller than the cut-off value, the people in A win the minority game. Otherwise, the people in B win.

Then we define the cut-off value of our considered problem in **Definition 2** and determine its value in **Theorem 1**.

Definition 2. (**Cut-off Value**) The cut-off value ϕ_m denotes the threshold for the number of video units offloaded to each edge server m, such that the maximal delay of computation and communication among all edge servers can be minimized. All cut-off values form the set $\boldsymbol{\Phi} = \{\phi_1, \phi_2, ..., \phi_M\}$, where $\sum_{m \in \boldsymbol{\mathcal{M}}} \phi_m = \sum_{n \in \boldsymbol{\mathcal{N}}} s_n$ is satisfied.

Theorem 1. *In order to minimize the maximal overall delay of computation and communication among all edge users, for each edge server $m \in \mathcal{M}$, we have*

$$\phi_m = (Cap_m \sum\nolimits_{n \in \boldsymbol{\mathcal{N}}} s_n) / \sum\nolimits_{j \in \boldsymbol{\mathcal{M}}} Cap_j. \tag{7}$$

Algorithm 1: GT-based Video Unit Allocation

Input: video unit number s_n, computing capacity Cap_m and transmission delay φ;

1 **for** *each edge user* $n \in \mathcal{N}$ **do**
2 Publish its number of video unit s_n;
3 Collect the video unit numbers of other users;
4 Calculate $\mathbf{\Phi} \leftarrow \{\phi_1, \phi_2, ..., \phi_M\}$ following Eq. (7);
5 **for** *each server* $m \in \mathcal{M}$ **do**
6 $x'_{n,m} \leftarrow \left\lfloor s_n \phi_m / \sum_{j \in \mathcal{M}} \phi_j \right\rfloor$;
7 Offload $x'_{n,m}$ video units to server m;
8 **for** *each edge user* $n \in \mathcal{N}$ **do**
9 Collect information of other users' allocation;
10 $\forall m \in \mathcal{M}, \lambda_m \leftarrow \lambda_m + s_{new}$;
11 $r_n = s_n - \sum_{m \in \mathcal{M}} x'_{n,m}$;
12 **for** *each remaining video unit* u **do**
13 Offload u to its potential optimal server obtained through **Alg. 2**;
14 Publish information of video unit allocation;

Proof. See Appendix A. □

According to **Definition 2**, when the number of video units offloaded to edge server m is smaller than ϕ_m, there must be at least one other edge server, of which the computation and communication delay is higher. On the contrary, when the number of video units offloaded to m is significantly larger than ϕ_m, it may raise the overall delay. Therefore, to minimize the maximal delay of computation and communication among all edge servers, the number of video units offloaded to the each edge server m should be close to the cut-off value ϕ_m.

B. GT-based Algorithm Design

Note that the cut-off values calculated in Eq. (7) might not be an integer, and we cannot directly use the cut-off values as the number of video units offloaded to the servers, which poses a challenge. In this subsection, we propose a GT-based video unit allocation algorithm (**Alg. 1**) based on the potential optimal server selection method (**Alg. 2**). Through the algorithms, the Nash equilibrium can be achieved, and users have no incentive to change their task offloading decisions unilaterally. We present the three main procedures of **Alg. 1**, which are shown in Fig. 2, as follows.

Initial Video Unit Allocation (Lines 2-7): Based on the cut-off value set $\mathbf{\Phi} = \{\phi_1, \phi_2, ..., \phi_M\}$, we get the number of video units that user n initially allocates to edge server m as

$$x'_{n,m} = \left\lfloor s_n \phi_m / \sum\nolimits_{j \in \mathcal{M}} \phi_j \right\rfloor, \tag{8}$$

where we use a rounding integer to approximate the initial video unit allocation $x'_{n,m}$. Notably more video units will be initially allocated to the servers with larger cut-off values.

Information Collection and Update (Lines 9-10): Users collect the information of other users' offloading decisions and learn the resource utilization of edge servers. For each edge server m, the current video unit number λ_m on it is updated as

$$\lambda_m = \lambda_m + s_{new}, \tag{9}$$

where s_{new} represents the number of video units newly offloaded to server m, and it depends on users' video unit allocation. Through the information collection and update, users make preparations for the remaining video unit allocation.

Algorithm 2: Potential Optimal Server Selection

Input: remaining video unit number r_n, computing capacity Cap_m and transmission delay φ;

1 **for** *each edge user* $n \in \mathcal{N}$ **do**
2 Collect the resource utilization λ_m of edge servers;
3 **for** *each remaining video unit* u *of user* n **do**
4 Calculate potential optimal server m^* upon Eqs. (11) and (12);
5 $\lambda_{m^*} \leftarrow \lambda_{m^*} + 1$;
6 Offload u to potential optimal server m^*;

Remaining Video Unit Allocation (Lines 11-14): Since we use the rounding integer to approximate the initial video unit allocation, there exists a gap between the initial video unit allocation and the theoretically optimal allocation. Thus, for each user n, we get the remaining video unit number as

$$r_n = s_n - \sum\nolimits_{m \in \mathcal{M}} x'_{n,m}. \tag{10}$$

For the remaining video units, **Alg. 2** is called, and users offload them to potential optimal servers defined in **Definition 3**.

Definition 3. (**Potential Optimal Server**) Before each remaining video unit for user n is offloaded, we calculate the potential delay $\Gamma_{n,m}$ for each server m as

$$\Gamma_{n,m} = \underbrace{\lambda_m C_u / Cap_m + \varphi}_{current\ delay} + \underbrace{C_u / Cap_m}_{added\ delay}, \tag{11}$$

where λ_m denotes the number of video units on the server m currently. For each remaining video unit of user n, the edge server m is the potential optimal server if and only if the potential delay of server m^* is the least among all servers, i.e.,

$$m^* = \arg\min\nolimits_{j \in \mathcal{M}} \{\Gamma_{n,j}\}. \tag{12}$$

As shown in the potential optimal server selection algorithm (**Alg. 2**), each user first collects the resource utilization (i.e., the current video unit number λ_m) of each server m. With the information collected, the potential optimal server m^* for each remaining video unit u is calculated upon Eqs. (11) and (12); after that, λ_{m^*} is updated in line 5. Finally, the remaining video unit u is offloaded to the potential optimal server m^*.

C. Algorithm Performance Analysis

In this subsection, we show that the NE can be achieved by the proposed algorithms in **Theorem 2**, and then we analyze their near-to-optimal performance in **Theorem 3**.

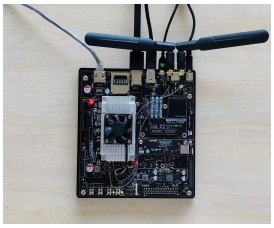
Theorem 2. *After the video units of all users are offloaded to the edge servers for video analytics through* ***Algorithm 1****, the Nash equilibrium is achieved, and no user has an incentive to change its task offloading decision unilaterally.*

Proof. See Appendix B. □

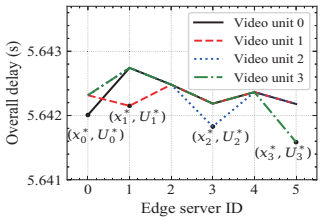
Theorem 3. *The gap between the practical overall delay $\mathcal{D}$ derived by* ***Algorithm 1*** *and the theoretical optimum $\mathcal{D}^*$ is bounded by $C_u / \min_m\{Cap_m\}$, where C_u is the computation requirement of each video unit and $\min_m\{Cap_m\}$ is the minimal computing capacity among all edge servers.*

Proof. See Appendix C. □

In practice, when the computing capacity of edge servers is about 10 GHz and C_u is set to about 10^6 CPU cycles, the gap between $\mathcal{D}$ and $\mathcal{D}^*$ can be bounded within 1 ms, which shows the superior performance of our design.



(a) NVIDIA Jetson TX2



(b) Nash Equlibrium

Fig. 3. Experimental Testbed and Results for Nash Equilibrium.

V. Experiments and Result Analysis

In this section, we evaluate the performance of our design through extensive trace-driven experiments with various settings, and compare it against some other existing approaches.

A. Experiment Settings

Similar to previous studies [15, 19], we consider an edge computing system containing 24 users and 6 edge servers. According to the experimental measurements in [20], the computing capacity Cap_m follows the Gaussian distribution $N(5, 4)$ GHz, and the variance of computing capacities varies in our experiments. Following the existing work [19], the channel bandwidth W is set to 20 MHz, the background noise σ is set to 50 dBm, and each user holds the transmission power $P_n \sim N(1, 0.1)$W. We set the channel gain $H_{n,0} = (dist_{n,0})^{\mu}$, where $dist_{n,0}$ is the distance between user n and master edge server, and the path loss factor μ is set to 4.

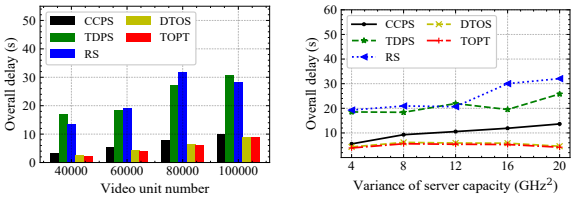
For task offloading, we use the videos derived from the AI City Datasets 2019 [7] for video analytics upon the object detector YOLOv3 [8]. The video length l_n varies from 10 seconds to 5 minutes, and the video frame rate f_n is assigned from the set $\{2, 3, 5, 10, 30\}$ fps as referred in [4]. We set L_u to 1 s and set F_u to 1 fps in our experiments. To properly set the value of C_u, we use YOLOv3 to process an image with the size of 1920×1080 on NVIDIA Jetson TX2 shown in Fig. 3(a). This image can be regarded as a video unit in our evaluation, and based on the experimental measurement result, C_u is set to 2.25×10^6 CPU cycles.

We refer to the GT-based video unit allocation algorithm in distributed scenario as Distributed Task Offloading Scheme (**DTOS**). We compare our designs with the following schemes:

- Computation Capacity Prior Scheme (**CCPS**): Users give priority to the computation capacity when making the decisions in the task offloading game.
- Transmission Delay Prior Scheme (**TDPS**): Users give priority to the transmission delay when making the decisions in the task offloading game.
- Random Scheme (**RS**): Users randomly select the edge servers for task offloading in the game.
- Theoretically OPTimal scheme (**TOPT**): TOPT calculates the theoretically optimal overall delay $\mathcal{D}^*$, which probably cannot be achieved in practice.

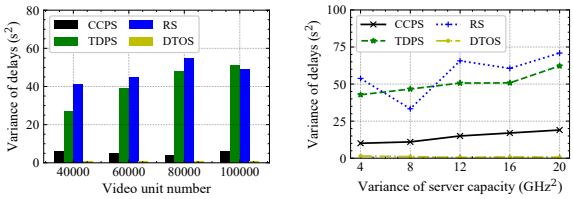
B. Experiment Results

Result for Nash Equilibrium: We first study the result for the achieved Nash equilibrium after running the GT-based video unit allocation algorithm. As shown in Fig. 3(b), we arbitrarily select four video units among all users and show the overall delays of the different edge servers to which the video units are offloaded. For video unit 0, which is from user 0, we observer that when it is on edge server 0, the overall delay is about 5.642 seconds. However, when video unit 0 is offloaded to other servers, the overall delay will become higher, and it is the same for the other video units. Therefore, it is demonstrated in Fig. 3(b) that after running the proposed algorithm, all users have no incentives to change their decisions unilaterally, and the Nash equilibrium is achieved.



(a) Video Unit Number (b) Server Capacity Variance

Fig. 4. Overall Delay with Varying Video Data Amount and Server Capacity.



(a) Video Unit Number (b) Server Capacity Variance

Fig. 5. Delay Variance with Varying Video Data Amount and Server Capacity.

Comparisons in Overall Delay: As shown in Fig. 4, we compare our designs with other baseline approaches with respect to the overall delay of transmission and computation. When the number of video units increases, it will cost more time for servers to run the computing tasks, and the overall delay gets higher, as shown in Fig. 4(a). It can also be observed that our design DTOS has a better performance in overall delay than other approaches. Then, we vary the variance of server capacity from 4 to 20, and find that the overall delays derived from CCPS show an increasing trend in Fig. 4(b). This might be due to the fact that more video data will be offloaded to the servers with high computing capacities, resulting in the high overall delay. In comparison, our design DTOS can get a steady overall delay in different settings. Furthermore, we compare DTOS with the theoretically optimal scheme TOPT in Fig. 4, and we find that the overall delay derived from DTOS is slightly more than that from TOPT. Therefore, our design can obtain the near-to-optimal solution.

Comparisons in Variance of Delays: As shown in Fig. 5, we investigate the differences in the delays of all edge servers, and calculate the delay variances with varying video data amount and server capacity. It is worth mentioning that in our experiments, the transmission delay seems relatively small when compared to the computation delay, i.e., the transmission delay is less important, and it conforms to the real world. Hence, in terms of both overall delay and variance of delays, CCPS performs better than TDPS. We observe that in various settings, the delay variances obtained from our designs are very close to 0, and we can learn that the gap between the maximum and minimum delay of all servers is very small, indicating our design can minimize the overall delay for users.

VI. Conclusion

To minimize the overall delay and obtain a stable situation where no user has an incentive to change its offloading decision unilaterally, we formulate the multi-server multi-user heterogeneous video analytics task offloading problem as a multi-player game. Through the proposed GT-based video unit allocation and potential optimal server selection algorithms, users select appropriate edge servers and offload their video data to them for video analytics. Via rigorous proof, our design achieves the near-optimal performance, and Nash equilibrium can be reached. Extensive trace-driven experiments show the improvement of our design compared with other algorithms.

Appendix

A. Proof of Theorem 1

Proof. Similar to the El Faro Bar problem, the maximal overall delay among all users is minimized when all servers share the equal delay of computation and communication, i.e.,

$$\tau_1 + \varphi = \tau_2 + \varphi = ... = \tau_M + \varphi = \mathcal{D}^*, \quad (13)$$

where the constant $\mathcal{D}^*$ is the theoretic optimal overall delay. According to Eq. (1), for each cut-off value ϕ_m, we have

$$\phi_m C_u / Cap_m + \varphi = \mathcal{D}^*. \quad (14)$$

By moving the terms in Eq. (14) and adding $\sum_{m\in\boldsymbol{\mathcal{M}}}$ to both sides, based on **Definition 2**, we get

$$\mathcal{D}^* = (\sum\nolimits_{n\in\boldsymbol{\mathcal{N}}} s_n C_u + \sum\nolimits_{m\in\boldsymbol{\mathcal{M}}} \varphi Cap_m) / \sum\nolimits_{m\in\boldsymbol{\mathcal{M}}} Cap_m. \quad (15)$$

By plugging Eq. (15) into (14), **Thm. 1** is proved. □

B. Proof of Theorem 2

Proof. To prove that NE is achieved, we only need to prove that the delay of computation and communication for any video unit u will not be reduced if we move it to another server from the potential optimal server m^*. Notably, there are two cases for the video unit u on server m^*: (a) Video unit u is the last one offloaded to server m^*. (b) Video unit u is not the last one offloaded to server m^*. For case (a), based on the definition of the potential optimal server, we easily get that the delay of computation and communication for u does not exceed the delay on any other server. For case (b), we denote the last video unit offloaded to server m^* as u_0. Since all the video units share the same computation requirement C_u, the delay of computation and communication for u will not decrease if it is moved to another server, similar to u_0 in case (a). Therefore, **Thm. 2** is proved. □

C. Proof of Theorem 3

Proof. Through **Alg. 1**, the video units of all edge users are offloaded to the appropriate edge servers for video analytics, and we have the practical overall delay as

$$\mathcal{D} = \max\nolimits_{m\in\boldsymbol{\mathcal{M}}}\{\tau_m + \varphi\}. \quad (16)$$

The practical overall delay $\mathcal{D}$ is the maximal delay of computation and communication among all of the edge servers, and we let m_1 denote the server of which the delay is $\mathcal{D}$. Besides, we let m_2 denote the server of which the delay of computation and communication is minimal, and we set

$$\mathcal{D}' = \tau_{m_2} + \varphi. \quad (17)$$

Since the cut-off value calculated in Eq. (7) might not be an integer, it cannot be directly utilized as the number of video units offloaded to servers. Thus it is certain that the number of video units offloaded to server m_1 is not smaller than cut-off value ϕ_{m_1}, and the number of video units offloaded to server m_2 does not exceed ϕ_{m_2}. Then we have $\mathcal{D}' \le \mathcal{D}^* \le \mathcal{D}$.

Based on **Thm. 2** and the idea in **Alg. 1**, for each video unit on the edge server m_1, if it is transferred to the other server m_2, the delay of computation and transmission will not decrease, i.e., $\mathcal{D} \le \mathcal{D}' + C_u/Cap_{m_2}$. Thus, we have

$$\mathcal{D}\text{–}\mathcal{D}^* \le \mathcal{D}\text{–}\mathcal{D}' \le C_u/Cap_{m_2} \le C_u/\min\nolimits_{m\in\boldsymbol{\mathcal{M}}}\{Cap_m\}, \quad (18)$$

and **Thm. 3** is proved. □

References

[1] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

[2] C.-C. Hung, G. Ananthanarayanan, P. Bodik, L. Golubchik, M. Yu, P. Bahl, and M. Philipose, "Videoedge: Processing camera streams using hierarchical clusters," in *IEEE/ACM SEC 2018*, pp. 115–131.

[3] Y. Chen, S. Zhang, Y. Jin, Z. Qian, M. Xiao, J. Ge, and S. Lu, "Locus: User-perceived delay-aware service placement and user allocation in mec environment," *IEEE TPDS*, 2021.

[4] C. Wang, S. Zhang, Y. Chen, Z. Qian, J. Wu, and M. Xiao, "Joint configuration adaptation and bandwidth allocation for edge-based real-time video analytics," in *IEEE INFOCOM 2020*, pp. 1–10.

[5] Z. Lu, K. S. Chan, and T. La Porta, "A computing platform for video crowdprocessing using deep learning," in *IEEE INFOCOM 2018*, pp. 1430–1438.

[6] Y. Wang, J. Yuan, G. Yu, Q. Chen, and R. Yin, "Minority game for distributed user association in unlicensed heterogenous networks," *IEEE TWC*, 2020.

[7] M. Naphade, Z. Tang, M.-C. Chang, D. C. Anastasiu, A. Sharma, R. Chellappa, S. Wang, P. Chakraborty, T. Huang, J.-N. Hwang, and S. Lyu, "The 2019 ai city challenge," in *IEEE CVPR Workshops*, 2019, p. 452–460.

[8] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *IEEE CVPR*, 2016, pp. 779–788.

[9] M. Gao, R. Shen, L. Shi, W. Qi, J. Li, and Y. Li, "Task partitioning and offloading in dnn-task enabled mobile edge computing networks," *IEEE TMC*, 2021.

[10] J. Yan, S. Bi, Y. J. Zhang, and M. Tao, "Optimal task offloading and resource allocation in mobile-edge computing with inter-user task dependency," *IEEE TWC*, vol. 19, no. 1, pp. 235–250, 2019.

[11] H. Wu, K. Wolter, P. Jiao, Y. Deng, Y. Zhao, and M. Xu, "Eedto: an energy-efficient dynamic task offloading algorithm for blockchain-enabled iot-edge-cloud orchestrated computing," *IEEE IoTJ*, vol. 8, no. 4, pp. 2163–2176, 2020.

[12] D. Kang, J. Emmons, F. Abuzaid, P. Bailis, and M. Zaharia, "Noscope: optimizing neural network queries over video at scale," *arXiv preprint arXiv:1703.02529*, 2017.

[13] D. Ray, J. Kosaian, K. Rashmi, and S. Seshan, "Vantage: optimizing video upload for time-shifted viewing of social live streams," in *ACM SIGCOMM 2019*, pp. 380–393.

[14] J. Ren, G. Yu, Y. Cai, and Y. He, "Latency optimization for resource allocation in mobile-edge computation offloading," *IEEE TWC*, vol. 17, no. 8, pp. 5506–5519, 2018.

[15] M. Hu, Z. Xie, D. Wu, Y. Zhou, X. Chen, and L. Xiao, "Heterogeneous edge offloading with incomplete information: A minority game approach," *IEEE TPDS*, vol. 31, no. 9, pp. 2139–2154, 2020.

[16] J. Zheng, Y. Cai, Y. Liu, Y. Xu, B. Duan, and X. Shen, "Optimal power allocation and user scheduling in multicell networks: Base station cooperation using a game-theoretic approach," *IEEE TWC*, vol. 13, no. 12, pp. 6928–6942, 2014.

[17] Y. Zhan, S. Guo, P. Li, and J. Zhang, "A deep reinforcement learning based offloading game in edge computing," *IEEE TC*, vol. 69, no. 6, pp. 883–893, 2020.

[18] S. Ranadheera, S. Maghsudi, and E. Hossain, "Minority games with applications to distributed decision making and control in wireless networks," *IEEE Wirel Commun*, vol. 24, no. 5, pp. 184–192, 2017.

[19] J. Zou, T. Hao, C. Yu, and H. Jin, "A3C-DO: A regional resource scheduling framework based on deep reinforcement learning in edge scenario," *IEEE TC*, 2020.

[20] N. Li, J.-F. Martinez-Ortega, and V. H. Diaz, "Distributed power control for interference-aware multi-user mobile edge computing: A game theory approach," *IEEE Access*, vol. 6, pp. 36 105–36 114, 2018.