Joint Task Offloading and Resource Allocation in Heterogeneous Edge Environments

Yu Liu*, Yingling Mao*, Zhenhua Liu†, Fan Ye*, and Yuanyuan Yang*
*Department of Electrical and Computer Engineering, Stony Brook University, USA
†Department of Applied Mathematics and Statistics, Stony Brook University, USA

IEEE INFOCOM 2023 - IEEE Conference on Computer Communications | 979-8-3503-3414-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/INFOCOM53939.2023.10229015

***Abstract*—Mobile edge computing is becoming one of the ubiquitous computing paradigms to support applications requiring low latency and high computing capability. FPGA-based reconfigurable accelerators have high energy efficiency and low latency compared to general-purpose servers. Therefore, it is natural to incorporate reconfigurable accelerators in mobile edge computing systems. This paper formulates and studies the problem of joint task offloading, access point selection, and resource allocation in heterogeneous edge environments for latency minimization. Due to the heterogeneity in edge computing devices and the coupling between offloading, access point selection, and resource allocation decisions, it is challenging to optimize over them simultaneously. We decomposed the proposed problem into two disjoint subproblems and developed algorithms for them. The first subproblem is to jointly determine offloading and computing resource allocation decisions and is NP-hard, where we developed an algorithm based on semidefinite relaxation. The second subproblem is to jointly determine access point selection and communication resource allocation decisions, where we proposed an algorithm with a provable approximation ratio of 2.62. We conducted extensive numerical simulations to evaluate the proposed algorithms. Results highlighted that the proposed algorithms outperformed baselines and were near-optimal over a wide range of settings.**

***Index Terms*—Edge Computing, Reconfigurable Accelerators**

I. Introduction

Recently, the development of information technology has given birth to a large number of novel applications requiring extra-low response time, e.g., augmented reality, virtual reality, the internet of things, and autonomous vehicles [1]. Due to the computing capability and energy consumption limitations, implementing such applications on wireless devices can not meet the low latency requirement of such applications. On the other hand, nor is cloud computing able to meet the low latency requirement due to network congestion and long physical distances. Since edge servers are located in close proximity to end-users and have powerful enough computing capability, offloading tasks of wireless devices to edge servers is becoming a ubiquitous computing paradigm for such applications.

With the advancement of Field Programmable Gate Arrays (FPGAs), FPGA-based reconfigurable accelerators have been widely adopted for various tasks [2]. Running specific jobs on FPGA has high efficiency in both computing time and energy consumption. For example, an AlexNet accelerator with 16-bit fixed point implemented on Xilinx Virtex-7 is 62× faster and uses 22× less energy compared to ARM Cortex A15 [3], [4]. FPGAs are frequently attached to a CPU in traditional computing systems, acting as powerful auxiliaries. Recently, a more advanced structure [5], [6] has been proposed where FPGAs can connect to networks as standalone computing resources following existing infrastructure as a service (IaaS) mechanisms. The availability of FPGAs as independent resources will enable high scalability, flexibility, and easy maintenance.

Existing task offloading works at the edge mainly focus on homogeneous systems that contain either general-purpose processors or FPGAs. In contrast, we consider a heterogeneous edge computing system where both FPGAs and general-purpose servers exist. Wireless devices (WDs) communicate with computing devices via access points (APs). There are four categories of decisions in such systems, namely offloading, access point selection, computing resource management, and communication resource management decisions. The goal is to minimize the average latency of all wireless devices.

Despite the advantages of edge computing and incorporating standalone FPGAs, it is challenging to choose offloading, AP selection, computing resource management, and communication resource management decisions jointly. First, computing devices in the system are heterogeneous and thus suitable for different jobs. For example, some jobs can be accelerated significantly on FPGAs, while others have less latency reduction on FPGAs. The amounts of resources required for performing tasks of different WDs on FPGAs are different. Therefore, to efficiently use limited FPGAs, offloading decisions must be made carefully, so tasks are offloaded to suitable computing devices. Second, the computing resources of general-purpose servers are limited and must be shared among multiple WDs [7]. The resource allocation among WDs must be balanced to ensure that the overall latency is as low as possible. Third, WDs can communicate with edge computing devices via different APs, and the channel conditions between WDs and APs can vary greatly. Consequently, we need to coordinate the AP selections for WDs to minimize their collective latencies. Lastly, the bandwidth of access points is limited, and the bandwidths have to be allocated to WDs in a collective optimal manner. Existing methods can not deal with the above challenges collectively, and this is the first work considering the joint tasking offloading and resource management problem in heterogeneous edge environments.

This work was supported in part by the National Science Foundation under grant numbers CCF-1730291, CCF-2046444, CNS-2146909, CNS-2106027, and CNS-2214980.

Authorized licensed use limited to: SUNY AT STONY BROOK. Downloaded on August 06,2024 at 18:45:00 UTC from IEEE Xplore. Restrictions apply.

Our main contributions are summarized as follows.

- We formulate the joint task offloading, AP selection, and resource management problem (JOAM) in heterogeneous edge environments, which we prove is NP-hard. JOAM can be divided into two disjoint NP-hard problems, namely Joint task Offloading and computing resource Management problem (JOM) for computing latency minimization and Joint AP selection and communication resource Management problem (JAM) for communication latency minimization, and see Section III-E for details.
- For the first subproblem (JOM) minimizing computing latency, we show that there is no polynomial-time approximation algorithm. Therefore, we develop a semidefinite relaxation-based algorithm to make offloading decisions and derive the closed-form optimal computing resource management decisions.
- For the second subproblem (JAM) minimizing communication latency, we design a game-theoretic algorithm to choose APs and allocate communication resources. We prove the approximation ratio of the proposed algorithm is 2.62. In addition, we show that there exists a trade-off between the time complexity and the approximation ratio of the proposed algorithm.
- We evaluate the algorithms by extensive simulations. The results highlight that the proposed algorithms outperform popular baselines, e.g., Markov chain Monte Carlo methods [8], and are near-optimal (1.02× and 1.05× the optimal latencies).

The remainder of this paper is organized as follows. Section II discusses related works. Section III presents the problem formulation. Sections IV and V propose algorithms for JOM and JAM, respectively. Section VI shows the performance evaluation of our algorithms. Section VII concludes this paper.

II. Related Works

A large number of works of job offloading with various purposes and settings in edge computing systems are well-studied, e.g., [7], [9]–[13]. In particular, [7], [9], [10] and [14] focus on minimizing the latency of jobs, [12] and [15] consider job offloading problems with the goal of optimizing the computation energy efficiency, and [11] formulates a job offloading problem that considers minimizing a combination of computing latency and energy consumption. In [13], Ali *et al.* consider the problem of minimizing energy consumption under latency constraints in a mobile cloud computing system. Different methodologies have been exploited for choosing decision variables. Game theoretic-based algorithms are used in [7], [9]. [11] proposes a reinforcement learning-based algorithm for choosing decision variables. [10] applies Lyapunov optimization to the proposed problem. [12] proposes an iterative and gradient descent method for minimizing its objective value. Some works assume the computing resource assigned to jobs are adjustable [7], [9], [11], [12], while others fix the amount of resources required by jobs [10]. In this paper, we consider a more general case where the amount of resources for jobs on general-purpose servers is adjustable and that for jobs on FPGAs is fixed.

[7], [9] are two papers that consider offloading jobs and resource allocating, where the amount of computing resources allocated to wireless devices is adjustable. In [7], Jošilo *et al.* consider the problem of allocating wireless and computing resources to wireless devices in an edge computing system for latency minimization, and an algorithm with a provable approximation ratio is proposed. In [9], Jošilo *et al.* extend the model in [7] by enabling network slicing. The amount of computing resource allocated to wireless devices is adjustable in both [7] and [9], while standalone FPGAs are included in this paper, and we can not change the amounts of FPGA resources assigned to FPGAs to tune the processing latency (detailed reasons can be found in Section III-D). The computing latencies of general-purpose servers and FPGAs follow different natures, leading to the difficulty of the latency minimization problem, and the previous methods in the literature can not solve the problem considered in this paper. The algorithm proposed in Section V can solve the problems in [7], [9], where our algorithm converges in polynomial steps and their algorithms may need exponential steps.

III. System Model

We consider the problem of joint task offloading and resource allocation in heterogeneous edge environments with the goal of minimizing the system latency. Some important notations are shown in Table I.

A. Edge Task Offloading System

a) System Components: We consider an edge system consisting of wireless edge devices (WDs), access points (APs), and edge computing devices. There are I WDs in the system, and $\mathcal{I} = [I] \triangleq \{1, 2, \cdots, I\}$ denotes the set of WDs. There are K APs in the system, and $\mathcal{K} = [K] \triangleq \{1, 2, \cdots, K\}$ represents the set of APs. For each AP $k \in \mathcal{K}$, it has an uplink bandwidth of $\overline{B}_k$ bits/s and a downlink bandwidth of $\underline{B}_k$ bits/s. There are two types of edge computing devices, namely general-purpose servers and FPGAs. We use $\mathcal{N} = [N] \triangleq \{1, 2, \cdots, N\}$ to denote the set of servers where N is the number of servers. Similarly, $\mathcal{M} = [M]$ represents the set of FPGAs where M is the number of FPGAs. We use F_n to denote the computing capability of server n, e.g., numbers of floating-point operations per second (FLOPs). For each $m \in \mathcal{M}$, $\mathbf{A}_m = \{A_{m,1}, A_{m,2}, \cdots, A_{m,L}\}$ is the vector representing the amounts of L different resources of FPGA m, i.e., the number of configurable logic blocks (CLBs), Flip-Flops, DSPs, BRAMs and so on [16]. $\mathcal{L} = \{1, 2, \cdots, L\}$ denotes the set the L types of resources. Multiple applications can share the resources of an FPGA board simultaneously [17], [18].

b) Network Topology: WDs can communicate with edge computing devices through APs. Each AP $k \in \mathcal{K}$ has a coverage area. A WD can be covered by more than one AP. We use $\mathcal{K}_i$ to denote the set of APs that cover the location of WD i where $\mathcal{K}_i \subseteq \mathcal{K}$. APs communicate with edge computing

Authorized licensed use limited to: SUNY AT STONY BROOK. Downloaded on August 06,2024 at 18:45:00 UTC from IEEE Xplore. Restrictions apply.

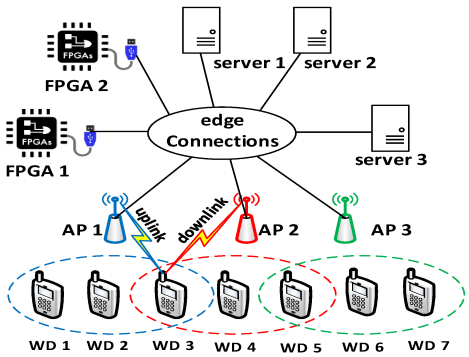

Fig. 1: Example of an edge computing system with $N = 3, M = 2, K = 3$ and $I = 7$. WD 3 uses AP 1 and AP 2 for uploading and downloading data, respectively.

devices by wired links [19], e.g., cellular base stations using fiber optic cables with a speed of up to 200 Gbps and wireless routers using twisted pair cables with a speed of up to 10 Gbps. Therefore, compared with the wireless links between WDs and APs, the latency of the wired links between APs and edge computing devices are negligible, which is also assumed by the literature [7], [9]. The edge computing devices are typically in an edger server room where the traditional baseband unit is, and each AP connects to the edge server room by optical fiber fronthaul link. Figure 1 shows an example of the network topology.

c) Tasks: WDs generate computing tasks periodically under a given frequency [20]. Each task of WD i has an input data size of $\overline{c}_i$ bits and an output data size of $\underline{c}_i$ bits. Each WD i offloads its tasks to either a server or an FPGA. If WD i offloads its tasks on a server, it takes f_i FLOPs to complete a task. When WD i offloads its tasks on FPGA m, denote by $\mathbf{a}_{i,m} = \{a_{i,m,1}, \cdots, a_{i,m,L}\}$ the required resources to implement the function of WD i. The time to complete a task of the function is denoted by $t_{i,m}$. $t_{i,m}$ and $a_{i,m,l}$ can be different across FPGA types m. Specifically, experiments in [21] showed that function completion time and its resource consumption vary across families of Xilinx, Altera, Actel, and Quick Logic FPGAs.

B. AP Selection and Wireless Resource Management

a) AP selection Decisions: Each WD $i \in \mathcal{I}$ has to choose an AP $\overline{k}_i \in \mathcal{K}_i$ for uploading input data. Similarly, WD $i \in \mathcal{I}$ has to choose an $\underline{k}_i \in \mathcal{K}_i$ for WD i for downloading output data. We use variable $\overline{z}_{i,k} \in \{0,1\}$ and variable $\underline{z}_{i,k} \in \{0,1\}$ to represent weather WD i choose AP k as its uploading AP and downloading AP, respectively. Let $\mathbf{z}_i = \{\overline{z}_{i,k}, \underline{z}_{i,k}\}_{k\in\mathcal{K}}$ be the collection of AP selection decisions of WD i. There are two constraints for the AP selection decisions of each WD i as follows:

$$\sum_{k\in\mathcal{K}_i} \overline{z}_{i,k} = 1 \text{ and } \sum_{k\in\mathcal{K}_i} \underline{z}_{i,k} = 1 \text{ for } i \in \mathcal{I}. \tag{1}$$

In addition, $\mathbf{z} = \{\overline{z}_{i,k}, \underline{z}_{i,k}\}_{i\in\mathcal{I},k\in\mathcal{K}}$ is the set of all AP selection decisions. For each AP $k \in \mathcal{K}$, let $\overline{\mathcal{O}}_k(\mathbf{z})$ ($\underline{\mathcal{O}}_k(\mathbf{z})$) be the collection of WDs using AP k as their uploading (downloading, respectively) AP. WDs in $\overline{\mathcal{O}}_k(\mathbf{z})$ ($\underline{\mathcal{O}}_k(\mathbf{d})$) share the uplink (downlink, respectively) bandwidth of AP k.

$\mathcal{I}, \mathcal{N}, \mathcal{M}$, and $\mathcal{K}$	sets of WDs, servers, FPGAs, and APs
$\mathbf{x} = \{x_{i,n} \mid i \in \mathcal{I}, n \in \mathcal{N}\}$	offloading decisions related to servers
$\mathbf{y} = \{y_{i,m} \mid i \in \mathcal{I}, m \in \mathcal{M}\}$	offloading decisions related to FPGAs
$\alpha = \{\alpha_{i,n} \mid i \in \mathcal{I}, n \in \mathcal{N}\}$	computing resource decisions
$\delta = \{\delta_{i,n} \mid i \in \mathcal{I}, n \in \mathcal{N}\}$	suitability between WDs and servers
$\mathbf{z} = \{\overline{z}_{i,k}, \underline{z}_{i,k} \mid i \in \mathcal{I}, k \in \mathcal{K}\}$	AP selection decisions
$\beta = \{\overline{\beta}_{i,k}, \underline{\beta}_{i,k} \mid i \in \mathcal{I}, k \in \mathcal{K}\}$	communication resource decisions
$\gamma = \{\gamma_{i,k} \mid i \in \mathcal{I}, k \in \mathcal{K}\}$	wireless channel condition
$T^P(\mathbf{x}, \mathbf{y}, \alpha) = \sum T_i^P(\mathbf{x}, \mathbf{y}, \alpha)$	total processing latency
$T^C(\mathbf{z}, \beta) = \sum T_i^C(\mathbf{z}, \beta)$	total communication latency

TABLE I: Important Notations

b) Wireless Channel Conditions: For each $i \in \mathcal{I}$ and $k \in \mathcal{K}$, there is a bandwidth utilization, denoted by $\gamma_{i,k} \in [0,1]$, associated with WD i and AP k, which reflects the condition of the wireless condition between WD i and AP k. $\gamma_{i,k}$ is given in advance, which is affected by the distance between WD i and AP k, the noise power of the channel between WD i and AP k, and so on. In particular, we set $\gamma_{i,k} = 0$ if WD i is not covered by AP k, i.e., $\gamma_{i,k} = 0$ if $k \notin \mathcal{K}_i$.

c) Wireless Bandwidth Resource Management: If WD $i \in \overline{\mathcal{O}}_k(\mathbf{z})$, there is a variable, $\overline{\beta}_{i,k}$, representing the proportion of the uplink bandwidth of AP k allocated to WD i. Similarly, if WD $i \in \underline{\mathcal{O}}_k(\mathbf{z})$, $\underline{\beta}_{i,k}$ represents the proportion of the downlink bandwidth of AP k allocated to WD i. Since the total bandwidth allocated to WDs can not exceed the total bandwidth of AP k, we have two constraints as follows:

$$\begin{aligned} \sum_{i\in\mathcal{I}} \overline{z}_{i,k}\overline{\beta}_{i,k} &= \sum_{i\in\overline{\mathcal{O}}_k(\mathbf{z})} \overline{\beta}_{i,k} \le 1 \\ \sum_{i\in\mathcal{I}} \underline{z}_{i,k}\underline{\beta}_{i,k} &= \sum_{i\in\underline{\mathcal{O}}_k(\mathbf{z})} \underline{\beta}_{i,k} \le 1. \end{aligned} \tag{2}$$

For the sake of simplicity, we use β to denote the collection of all communication resource management variables, i.e., $\beta = \{\overline{\beta}_{i,k}, \underline{\beta}_{i,k} \mid i \in \mathcal{I}, k \in \mathcal{K}\}$.

We distinguish the uplink and downlink bandwidth to handle the case that uplink and downlink use different frequency bands, e.g., frequency division multiplexing (FDD) protocols [22]. Our algorithm can handle the case that there is no distinction between uplink and downlink bandwidth, i.e, each AP k has only one bandwidth constraint $\sum_{i\in\overline{\mathcal{O}}_k(\mathbf{z})} \overline{\beta}_{i,k} + \sum_{i\in\underline{\mathcal{O}}_k(\mathbf{z})} \underline{\beta}_{i,k} \le 1$, which is a degenerated case.

C. Task Offloading and Computing Resource Management

a) Offloading to Server: For each WD $i \in \mathcal{I}$ and server $n \in \mathcal{N}$, there is a variable $x_{i,n} \in \{0,1\}$. In particular, $x_{i,n} = 1$ if WD i offloads its tasks on server i, and $x_{i,n} = 0$ otherwise. $\mathbf{x}_n = (x_{1,n}; x_{2,n}; \cdots; x_{I,n})$ denotes the collection of $x_{i,n}$ related to server n. In addition, $\mathbf{x}$ denotes the collection of $x_{i,n}$ for $i \in \mathcal{I}, n \in \mathcal{N}$. $\mathcal{O}_n(\mathbf{x})$ is the set of WDs that place their tasks on server n, i.e., $x_{i,n} = 1$ if $i \in \mathcal{O}_n(\mathbf{x})$. If WD i offloads its tasks on server n, i.e., $i \in \mathcal{O}_n(\mathbf{x})$, we use $\alpha_{i,n}$

Authorized licensed use limited to: SUNY AT STONY BROOK. Downloaded on August 06,2024 at 18:45:00 UTC from IEEE Xplore. Restrictions apply.

to denote the proportion of computing capability of server n allocated to WD i. Note that the server can be not only a CPU but also a GPU because the MPS scheme allows different jobs running on different address spaces of a GPU [23]. There is a constraint limiting that the amount of computing capability allocated to WDs in $\mathcal{O}_n(\mathbf{x})$ can not exceed the total computing capability of server n as follows:

$$\sum_{i\in\mathcal{I}} x_{i,n}\alpha_{i,n} \triangleq \sum_{i\in\mathcal{O}_n(\mathbf{x})} \alpha_{i,n} \leq 1. \tag{3}$$

For the sake of simplicity, we use $\alpha = \{\alpha_{i,n} | i \in \mathcal{I}, n \in \mathcal{N}\}$ to denote the collection of all computing resource management variables.

b) Suitability Between servers and Tasks: Since different servers may be equipped with different amounts of CPUs, GPUs, etc., some servers are more suitable for running some types of tasks. For each WD i and server n, there is a suitability $\delta_{i,n} \in [0,1]$ [14]. $\delta_{i,n} \in [0,1]$ depicts how well a server n is fitting for running tasks of WD i. The larger $\delta_{i,n}$, the better the suitability of offloading tasks of WD i to server n.

c) Offloading to Field Programmable Gate Arrays: For each WD $i \in \mathcal{I}$ and FPGA $m \in \mathcal{M}$, there is a decision variable $y_{i,m} \in \{0,1\}$. In particular, $y_{i,m} = 1$ if WD i places its tasks on FPGA m, and $y_{i,m} = 0$ otherwise. $\mathbf{y}$ is the collection of $y_{i,m}$ for $i \in \mathcal{I}, m \in \mathcal{M}$. Moreover, $\mathcal{O}_m(\mathbf{y})$ represents the set of WDs that place their tasks on FPGA m, i.e., $i \in \mathcal{O}_m(\mathbf{y})$ if $y_{i,m} = 1$. There is a constraint limiting the total amounts of resources required by WDs in $\mathcal{O}_m(\mathbf{y})$ can not exceed the resource amounts of FPGA m as follows:

$$\sum_{i\in\mathcal{I}} y_{i,m}a_{i,m,l} \triangleq \sum_{i\in\mathcal{O}_m(\mathbf{y})} a_{i,m,l} \leq A_{m,l} \text{ for } m \in \mathcal{M}, l \in \mathcal{L}. \tag{4}$$

d) Constraint of Offloading Decision: The collection of offloading decisions is $(\mathbf{x}, \mathbf{y})$. Since each WD i offloads its tasks on either a server $n \in \mathcal{N}$ or an FPGA $m \in \mathcal{M}$, we have the following constraint regarding $(\mathbf{x}, \mathbf{y})$:

$$\sum_{n\in\mathcal{N}} x_{i,n} + \sum_{m\in\mathcal{M}} y_{i,m} = 1 \text{ for } i \in \mathcal{I}. \tag{5}$$

D. Goal of the System

The goal of the system is to minimize the summation of latency of all WDs. The latency of each WD i consists two parts, namely processing latency T_i^P and communication latency T_i^C.

a) Processing Latency: If WD i places its task on server n, the average processing latency can be expressed as a function of the amount of computing capability allocated to WD i [24], i.e.,

$$T_i^P = x_{i,n} \cdot f_i / \big(F_n \delta_{i,n} \alpha_{i,n}\big) \tag{6}$$

where $\delta_{i,n}$ is a fixed parameter reflecting the suitability of running tasks of WD i on server n. For example, $\delta_{i,n}$ is different in two cases where the server is a CPU and a GPU. We can tune the above latency by varying $\alpha_{i,n}$ [7], [23]. On the other hand, if WD i offloads its tasks on FPGA m, the processing latency of WD i is $t_{i,m}$, i.e.,

$$T_i^P = y_{i,m} t_{i,m}. \tag{7}$$

Different from the latencies of tasks on servers, latencies of tasks on FPGAs can not be decreased by increasing the number of configurable logic blocks for the following reasons. First, the functions running on FPGAs are described by hardware description language in advance. Once the hardware description code is given and an FPGA is specified, the resource consumption and the latency are also fixed. Second, it is wasteful and time-consuming to develop different versions of FPGA implementation. Last and most importantly, varying the implementation (reprogramming FPGA) takes time, e.g., hundreds of ms up to tens of seconds, which is fatal to applications requiring extra-low latency.

From (6) and (7), the processing latency is a function of $(\mathbf{x}, \mathbf{y}, \alpha)$, and we use $T^P(\mathbf{x}, \mathbf{y}, \alpha)$ to denote the summation of processing latency of all WDs, i.e.,

$$T^P(\mathbf{x}, \mathbf{y}, \alpha) = \sum_{i\in\mathcal{I}} \Big(\sum_{n\in\mathcal{N}} \frac{x_{i,n} f_i}{F_n \delta_{i,n} \alpha_{i,n}} + \sum_{m\in\mathcal{M}} y_{i,m} t_{i,m} \Big). \tag{8}$$

b) Communication Latency: Since we focus on edge computing systems, the WDs and APs are in close vicinity; therefore, the propagation delay is negligible, and we only need to consider the transmission delay. The transmission latency of WD i consists of input data uploading latency and output data downloading latency. If AP k is the uploading AP of WD i, the uploading transmission delay of WD i is denoted by $\overline{T}_i^C$, i.e., $\overline{T}_i^C = \overline{c}_i / (\gamma_{i,k} \cdot \overline{B}_k \cdot \overline{\beta}_{i,k})$ if $\overline{z}_{i,k} = 1$. Similarly, if AP k is the downloading AP of WD i, the downloading transmission delay of WD i is denoted by $\underline{T}_i^C$, i.e., $\underline{T}_i^C = \underline{c}_i / (\gamma_{i,k} \cdot \underline{B}_k \cdot \underline{\beta}_{i,k})$ if $\underline{z}_{i,k} = 1$. Let T_i^C be the average communication latency of WD i, i.e., $T_i^C = \overline{T}_i^C + \underline{T}_i^C$. In addition, the summation of communication latencies of all WDs is a function of $\mathbf{z}, \beta$ as follows:

$$T^C(\mathbf{y}, \beta) = \sum_{i\in\mathcal{I}} T_i^C(\mathbf{y}, \beta) = \sum_{i\in\mathcal{I}} \big(\overline{T}_i^C + \underline{T}_i^C\big). \tag{9}$$

E. Problem Formulation

In this subsection, we formally state the problem we formulated above as an optimization problem, and we refer to the problem as JOAM which is short for Joint job Offloading, AP selection, and resource Management. JOAM is as follows:

$$\min_{\mathbf{x},\mathbf{y},\mathbf{z},\alpha,\beta} \quad T^P(\mathbf{x}, \mathbf{y}, \alpha) + T^C(\mathbf{y}, \beta) \tag{JOAM}$$

$$\text{s.t.} \quad (1) - (5)$$

$$x_{i,n} \in \{0,1\} \text{ for } i \in \mathcal{I} \text{ and } n \in \mathcal{N} \tag{10}$$

$$y_{i,m} \in \{0,1\} \text{ for } i \in \mathcal{I} \text{ and } m \in \mathcal{M} \tag{11}$$

$$\alpha_{i,n} \in [0,1] \text{ for } i \in \mathcal{O}_n(\mathbf{x}) \text{ and } n \in \mathcal{N} \tag{12}$$

$$\overline{z}_{i,k}, \underline{z}_{i,k} \in \{0,1\} \text{ for } i \in \mathcal{I} \text{ and } k \in \mathcal{K} \tag{13}$$

$$\overline{\beta}_{i,k} \in [0,1] \text{ for } i \in \overline{\mathcal{O}}_k(\mathbf{z}) \text{ and } k \in \mathcal{K} \tag{14}$$

$$\underline{\beta}_{i,k} \in [0,1] \text{ for } i \in \underline{\mathcal{O}}_k(\mathbf{z}) \text{ and } k \in \mathcal{K}. \tag{15}$$

Authorized licensed use limited to: SUNY AT STONY BROOK. Downloaded on August 06,2024 at 18:45:00 UTC from IEEE Xplore. Restrictions apply.

The decision variables of JOAM can be partitioned into two sets, namely $(\mathbf{x}, \mathbf{y}, \alpha)$ and $(\mathbf{z}, \beta)$. There is no coupling between $(\mathbf{x}, \mathbf{y}, \alpha)$ and $(\mathbf{z}, \beta)$ in the constraints. In addition, communication latency T^C is merely determined by $(\mathbf{z}, \beta)$, and processing latency T^P is merely determined by $(\mathbf{x}, \mathbf{y}, \alpha)$. Therefore, JOAM can be divided into two disjoint subproblems, namely the Joint task Offloading and computing resource Management problem (JOM) of minimizing T^P over $(\mathbf{x}, \mathbf{y}, \alpha)$ and the AP selection and communication resource Management problem (JAM) of minimizing T^C over variables $(\mathbf{z}, \beta)$. The two disjoint subproblems are as follows:

$$\begin{aligned} &\min_{\mathbf{x},\mathbf{y},\alpha} && T^P(\mathbf{x}, \mathbf{y}, \alpha) && \text{(JOM)} \\ &\text{s.t.} && (3)-(5), (12)-(14) \\ &\min_{\mathbf{z},\beta} && T^C(\mathbf{z}, \beta) && \text{(JAM)} \\ &\text{s.t.} && (1)-(2), (15)-(17) \end{aligned}$$

In what follows, we show the NP-hardness of JOAM. Actually, both JOM and JAM are NP-hard.

Theorem 1. *JOAM is NP-hard.*

Proof. JOM is a special version of JOAM. To be more specific, JOM is equivalent to JOAM if there is only one AP covering all WDs and having infinite uplink and downlink bandwidth. JOAM is NP-hard because JOM, a special version of JOM, is NP-hard. The NP-hardness of JOM is shown in Theorem 2. □

IV. Algorithm Design for Task Offloading and Computing Resource Management

In this section, we focus on the first subproblem of JOAM, i.e., JOM. We first show the hardness of JOM. Then, We design an algorithm for JOM.

Theorem 2. *JOM is NP-hard, and no polynomial-time approximation algorithm is possible unless there are some additional assumptions.*

The main idea of the proof is to show a special version of JOM is equivalent to the Generalized Assignment Problem (GAP) (Chapter 48 of [25]). Under the special version of JOM, there is no general-purpose server and only one type of resource constraint, i.e., $N = 0$ and $m = 1$. Detailed proof of Theorem 2 is omitted due to space limitations.

A. Computing Resource Management Under Given Offloading Decisions

We then consider the problem of finding the optimal computing resource management decision α under any given offloading decision $(\mathbf{x}, \mathbf{y})$. For WDs placing their tasks on gate arrays, their latencies are fixed. Therefore, if $(\mathbf{x}, \mathbf{y})$ is given, JOM is equivalent to minimizing the summation of T_i^P for $i \in \bigcup_{n\in\mathcal{N}} \mathcal{O}_n(\mathbf{x})$ over α as follows:

$$\begin{aligned} \min_{\alpha} \quad & \sum_{n\in\mathcal{N}} \sum_{i\in\mathcal{O}_n(\mathbf{x})} f_i/\left(F_n \delta_{i,n} \alpha_{i,n}\right) \\ \text{s.t.} \quad & \alpha_{i,n} \in [0,1] \text{ for } n \in \mathcal{N}, i \in \mathcal{O}_n(\mathbf{x}) \\ & \sum_{i\in\mathcal{O}_n(\mathbf{x})} \alpha_{i,n} \le 1 \text{ for } n \in \mathcal{N}. \end{aligned} \tag{16}$$

We use $\alpha^*(\mathbf{x}) = \{\alpha_{i,n}^* | n \in \mathcal{N}, i \in \mathcal{O}_n(\mathbf{x})\}$ to denote the optimal solution of (16) under offloading decision $\mathbf{x}$. The optimal computing resource management decision $\alpha^*(\mathbf{x})$ is shown in Lemma 1.

Lemma 1. For any given $\mathbf{x}$, $\alpha^*(\mathbf{x})$ is as follows:

$$\alpha_{i,n}^* = \frac{\sqrt{f_i/\delta_{i,n}}}{\sum_{j\in\mathcal{O}_n(\mathbf{x})} \sqrt{f_j/\delta_{i,n}}} \text{ for } n \in \mathcal{N}, i \in \mathcal{O}_n(\mathbf{x}). \tag{17}$$

The proof of Lemma 1 is omitted due to space limitations. The main idea of the proof is to exploit the KKT conditions. Substituting α^* into (8) and changing the order in the double sum, the optimal processing latency under offloading decision $(\mathbf{x}, \mathbf{y})$ is equal to

$$T^P = \sum_{n\in\mathcal{N}} \sum_{i\in\mathcal{I}} \sum_{j\in\mathcal{I}} \frac{x_{i,n} x_{j,n}}{F_n} \sqrt{\frac{f_i f_j}{\delta_{i,n}\delta_{j,n}}} + \sum_{i\in\mathcal{I}} \sum_{m\in\mathcal{M}} y_{i,m} t_{i,m}.$$

That is, by substituting α^* into JOM, JOM is equivalent to P1 as follows:

$$\begin{aligned} \min_{\mathbf{x},\mathbf{y}} \quad & \sum_{n\in\mathcal{N}} \frac{1}{F_n} \sum_{i\in\mathcal{I}} \sum_{j\in\mathcal{I}} x_{i,n} x_{j,n} \sqrt{\frac{f_i f_j}{\delta_{i,n}\delta_{j,n}}} + \sum_{i\in\mathcal{I}} \sum_{m\in\mathcal{M}} y_{i,m} t_{i,m} \\ \text{s.t.} \quad & \sum_{i\in\mathcal{O}_m(\mathbf{y})} a_{i,m,l} \le A_{m,l} \text{ for } m \in \mathcal{M}, l \in \mathcal{L} \\ & \sum_{n\in\mathcal{N}} x_{i,n} + \sum_{m\in\mathcal{M}} y_{i,m} = 1 \text{ for } i \in \mathcal{I} \\ & x_{i,n}, y_{i,m} \in \{0,1\} \text{ for } i \in \mathcal{I}, n \in \mathcal{N}, m \in \mathcal{M}. \end{aligned} \tag{P1}$$

We use $(\mathbf{x}^*, \mathbf{y}^*)$ to denote the optimal solution of P1. Accordingly, $\left(\mathbf{x}^*, \mathbf{y}^*, \alpha^*(\mathbf{x}^*)\right)$ is the optimal solution of JOM. Since we have the optimal solution $\alpha^*(\mathbf{x})$ under any given $(\mathbf{x}, \mathbf{y})$, we then focus on choosing $(\mathbf{x}, \mathbf{y})$, i.e., solving P1.

B. Algorithm Design for P1

Although we eliminate decision α, P1 is still NP-hard and there is no polynomial-time approximation algorithm for it. The proof is similar to that of Theorem 2, and we omit the proof due to space limitations. Semidefinite relaxation to P1 is a practical approach for finding a feasible solution nearing the optimal solution in polynomial time. Next, we focus on developing a semidefinite programming (SDP) relaxation approach for P1.

First, we rewrite P1 in the standard binary quadratic programming (BQP) form. Let u be the column vector unifying all variables of P1, i.e., $\mathbf{x}$ and $\mathbf{y}$. In particular, we have $u = (\mathbf{x}_1; \mathbf{x}_2; \cdots; \mathbf{x}_N; \mathbf{y}_1; \mathbf{y}_2; \cdots; \mathbf{y}_M)$ where $\mathbf{x}_n = (x_{1,n}; \cdots; x_{I,n})$ and $\mathbf{y}_n = (y_{1,m}; y_{2,m}; \cdots; y_{I,m})$. $J = (M+N)I$ is the number of binary variables of P1. By introducing a new variable $U = u \cdot u^T$, we can reformulate P1 as:

$$\begin{aligned} \min_{u,U} \quad & \mathbf{Tr}(PU) + q^T u \\ \text{s.t.} \quad & h_i^T u = 1 \text{ for } i \in \mathcal{I} \\ & d_{m,l}^T u \le A_{m,l} \text{ for } m \in \mathcal{M}, l \in \mathcal{L} \\ & U = u \cdot u^T. \end{aligned} \tag{18}$$

P is a $J \times J$ matrix corresponding to the quadratic terms in the objective function of P1. $q, h_i, d_{m,l}$ are column vectors of

Authorized licensed use limited to: SUNY AT STONY BROOK. Downloaded on August 06,2024 at 18:45:00 UTC from IEEE Xplore. Restrictions apply.

Algorithm 1: SDPR

Input: $F_n, n \in \mathcal{N}$, $f_i, i \in \mathcal{I}$, $\mathbf{A}_m, m \in \mathcal{M}$, $\mathbf{a}_i, i \in \mathcal{I}$, $t_i, i \in \mathcal{I}$

Parameter: IterNum

Output: A feasible solution to P1: $(\mathbf{x}^{best}, \mathbf{y}^{best})$

Calculate P, q, c_i for $i \in \mathcal{I}$, and d_m for $m \in \mathcal{M}$;
Solve SDP (19) to get u^* and U^*;
Initialize $(\mathbf{x}^{best}, \mathbf{y}^{best}) = \mathbf{round}(u^*)$;
Initialize $L^{best} = T^P(\mathbf{x}^{best}, \mathbf{y}^{best}, \alpha^*(\mathbf{x}^{best}, \mathbf{y}^{best}))$;
Set $\mu = u^*$, $\Sigma = U^* - u^* \cdot u^{*T}$, and $l = 0$;
while $l <$ ***IterNum*** **do**
  Randomly sample $v^{(l)}$ from $\mathcal{N}(\mu, \Sigma)$;
  Calculate $(\mathbf{x}^{(l)}, \mathbf{y}^{l}) = \mathbf{round}(v^l)$;
  Calculate $L^{(l)} = T^P\left(\mathbf{x}^{(l)}, \mathbf{y}^{(l)}, \alpha^*(\mathbf{x}^{(l)}, \mathbf{y}^{(l)})\right)$;
  if $L^{(l)} < L^{best}$ **then**
    $L^{best} := L^{(l)}$;
    $(\mathbf{x}^{best}, \mathbf{y}^{best}) := (\mathbf{x}^{(l)}, \mathbf{y}^{(l)})$;
  end
  $l := l + 1$;
end

length J. P can be written in the form of $Q \cdot Q^T$. Therefore, P is positive semidefinite, i.e., $P \succcurlyeq 0$. The only non-convex constraint in (18) is $U = u \cdot u^T$. By relaxing $U = u \cdot u^T$ to $U \succcurlyeq u \cdot u^T$ and adding constraint $u_j(1-u_j) \leq 1$, (18) can be relaxed to a convex optimization problem as follows:

$$\begin{aligned} \min_{u,U} \quad & \mathbf{Tr}(PU) + q^T u \\ \text{s.t.} \quad & h_i^T u = 1 \text{ for } i \in \mathcal{I}, \\ & d_{m,l}^T u \leq A_{m,l} \text{ for } m \in \mathcal{M}, l \in \mathcal{L}, \\ & \mathbf{diag}(U) \leq u \cdot u^T, \end{aligned} \quad \left\| \begin{bmatrix} U & u \\ u^T & 1 \end{bmatrix} \right. \succcurlyeq 0. \tag{19}$$

The last constraint in (19) holds if and only if $U \succcurlyeq 0$ and $U - u \cdot u^T \succcurlyeq 0$, which can be proved by exploiting Schur complement. Let (u^*, U^*) be the optimal solution to (19). Since (19) is a relaxation of P1, the optimal objective value of (19) is a natural lower bound of the minimum value of P1.

Next, we show a physical interpretation of the semidefinite relaxation (19). Let v be a J-dimensional joint normal random vector. Let μ and Σ be the mean and covariance matrix of v, respectively. That is, $v \sim \mathcal{N}(\mu, \Sigma)$. Then, from [26], if constraint $\mathbf{diag}(U) \leq u \cdot u^T$ is not in (19), $\mu = u^*$ and $\Sigma = U^* - \mu \cdot \mu^T$ minimizes the following problem:

$$\begin{aligned} \min \quad & \mathbb{E}\left[v^T P v + q^T v\right] \\ \text{s.t.} \quad & \mathbb{E}\left[h_i^T v\right] = 1 \text{ for } i \in \mathcal{I} \\ & \mathbb{E}\left[d_{m,l}^T v\right] \leq A_{m,l} \text{ for } m \in \mathcal{M}, l \in \mathcal{L}. \end{aligned} \tag{20}$$

Intuitively, v drawn from $\mathcal{N}(\mu, \Sigma)$ under $\mu = u^*$ and $\Sigma = U^* - \mu \cdot \mu^T$ has a cost close to the optimal objective value of (19). We can draw a number of samples from $\mathcal{N}(\mu, \Sigma)$ with $\mu = u^*$ and $\Sigma = U^* - \mu \cdot \mu^T$, round each of them to a feasible point, and choose the feasible point with the lowest cost. Let **round** be the operator rounding u to a feasible solution to P1. A specific rounding algorithm is proposed in the following section. We then formally state the proposed algorithm, named SDPR, for P1 in Algorithm 1.

Algorithm 2: Round (Rounding Algorithm)

Input: $F_n, n \in \mathcal{N}$, $f_i, i \in \mathcal{I}$, $\mathbf{A}_m, m \in \mathcal{M}$, $\mathbf{a}_i, i \in \mathcal{I}$, $t_i, i \in \mathcal{I}$, $\mathbf{x}^c$, and $\mathbf{y}^c$

Parameter: IterNum

Output: A feasible solution to P1: $(\mathbf{x}^{fsb}, \mathbf{y}^{fsb})$

$\mathcal{I}_F := \emptyset$, $\mathcal{I}_S := \mathcal{I}$, $l := 1$,$\mathbf{x}^{fsb} = 0$,$\mathbf{y}^{fsb} = 0$;
while $\exists j \in \mathcal{I}, m \in \mathcal{M}$ *such that* $\mathbf{a}_j \leq \mathbf{A}_m$ **do**
  $i := \arg\max_{j \in \mathcal{I}_S} \max\{y_{j,1}^c, \cdots, y_{j,M}^c\}$;
  $m^* := \arg\max_{m \in \mathcal{M}} y_{i,m}^c$;
  if $\mathbf{a}_i \leq \mathbf{A}_{m^*}$ **then**
    $y_{i,m}^{fsb} := 0$ for $m \in \mathcal{M} \setminus \{m^*\}$, $y_{i,m^*}^{fsb} := 1$;
    $\mathbf{A}_{m^*} := \mathbf{A}_{m^*} - \mathbf{a}_i$;
    $\mathbf{a}_i := \infty$;
    $y_{i,m}^c := 0$ for $i \in \mathcal{I}$;
    $\mathcal{I}_F = \mathcal{I}_F \cup \{i\}$ and $\mathcal{I}_S = \mathcal{I}_S \setminus \{i\}$;
  else
    $y_{i,m^*}^c := 0$;
  end
end
$x_{i,n}^{fsb} := 0$ for $i \in \mathcal{I}$ and $n \in \mathcal{N}$;
$x_{i,n^*}^{fsb} := 1$ for $i \in \mathcal{I}_S$ and $n^* = \arg\max_n x_{i,n}^c$;
while $l <$ ***IterNum*** *and* $\exists i \in \mathcal{I}_S$ *such that* $T_i^P(\mathbf{x}^{fsb}) \overset{(a)}{<} \min_{n \in \mathcal{N}} \frac{\sqrt{f_i/\delta_{i,n}}}{F_n}\left(\frac{\sqrt{f_i}}{\sqrt{\delta_{i,n}}} + \sum_{j \in \mathcal{I}} x_{j,n} \frac{\sqrt{f_j}}{\sqrt{\delta_{j,n}}}\right)$ **do**
  Randomly choose an i satisfying inequality (a);
  $\hat{n} := \arg\min_n \frac{\sqrt{f_i/\delta_{i,n}}}{F_n}\left(\frac{\sqrt{f_i}}{\sqrt{\delta_{i,n}}} + \sum_{j \in \mathcal{I}} x_{j,n} \frac{\sqrt{f_j}}{\sqrt{\delta_{j,n}}}\right)$;
  $x_{i,n}^{fsb} := 0$ for $n \in \mathcal{N} \setminus \{\hat{n}\}$ and $x_{i,\hat{n}}^{fsb} := 1$;
  $l := l + 1$;
end

C. Rounding the Solution of the Semidefinite Relaxation

Next, we propose a rounding algorithm that rounds vector v of length J to a feasible solution $(\mathbf{x}, \mathbf{y})$ for P1. Before the rounding process, we unzip v to $(\mathbf{x}^c, \mathbf{y}^c)$, where v is the input vector. In particular, $x_{i,n}^c = v_{(n-1)\cdot I + i}$ for $i \in \mathcal{I}$ and $n \in \mathcal{N}$ and $y_{i,m}^c = v_{(N+m-1)\cdot I + i}$ for $i \in \mathcal{I}$ and $m \in \mathcal{M}$. The rounding algorithm is formally stated in Algorithm 2.

We first consider rounding $\mathbf{y}^c$ to a feasible $\mathbf{y}$, where $\mathbf{y}$ is an I-by-M binary matrix, $\mathbf{y}_m$ is the m^{th} column of $\mathbf{y}$. We sort WDs $\mathcal{I}$ by the value of $\max_m y_{i,m}^c$ in descending order. WDs in the order take turns to round $\{y_{i,m}^c | m \in \mathcal{M}\}$ to $\{y_{i,m} | m \in \mathcal{M}\}$. In particular, WD i set $y_{i,m} = 1$ if $m = \arg\max_{m'} y_{i,m'}^c$ and $\mathbf{a}_i$ is no greater then the available space of FPGA m, and

Authorized licensed use limited to: SUNY AT STONY BROOK. Downloaded on August 06,2024 at 18:45:00 UTC from IEEE Xplore. Restrictions apply.

$y_{i,m} = 0$ otherwise. Let $\mathcal{I}_F$ be the set of WDs offloading its tasks to $\mathcal{M}$, i.e., $i \in \mathcal{I}_F$ if and only if $\sum_{m\in\mathcal{M}} y_{i,m} = 1$. Then, since we have the constraint that $\sum_{m\in\mathcal{M}} y_{i,m} + \sum_{n\in\mathcal{N}} x_{i,n} = 1$, we have $x_{i,n} = 0$ for $i \in \mathcal{I}_F$. That is, we only need to consider $x_{i,n}$ for $i \in \mathcal{I}_S \triangleq \mathcal{I} \setminus \mathcal{I}_F$ in the following.

Then, we consider rounding $x^c_{i,n}$ to binary for $i \in \mathcal{I}_S$ and $n \in \mathcal{N}$. Each WD i sets $x_{i,n}$ to 1 if $n = \arg\max_{n'} x^c_{i,n'}$ and sets $x_{i,n}$ to 0 otherwise. Then, WDs in $\mathcal{I}_S$ take turns to adjust their decision. To be more specific, if $x_{i,n} = 1$ and there exists $\bar{n} \in \mathcal{N}$ such that moving WD i from n to $\bar{n}$ can lower the latency of WD i given other WDs' decisions, WD i resets $x_{i,\bar{n}} = 1$ and $x_{i,n} = 0$ for $n \neq \bar{n}$. We can set a maximum number of iterations for WDs to adjust their decisions. In fact, if there is no limit for maximum iteration, it can be proved that the decision adjustment process for WDs in $\mathcal{I}_S$ will terminate after a finite number of iterations (see [28] for details).

The time complexity of rounding $\mathbf{y}$ (Line 1-14) is $O(I^2 \cdot \log(I))$ and the time complexity of rounding $\mathbf{x}$ (Line 15-22) is $O(IN + I \cdot \text{IterNum})$. That is, the time complexity of the rounding algorithm is polynomial to the system parameters.

V. Algorithm Design for AP selection and Communication Resource Management

In this section, we focus on designing an algorithm for JAM. First, we show the hardness of JAM. Then, we propose an algorithm for solving JAM.

The hardness of JAM is shown in Theorem 3.

Theorem 3. *JAM is strongly NP-hard, and there is no fully polynomial-time approximation scheme (FPTAS) and pseudo-polynomial time algorithm for JAM.*

The proof of Theorem 3 is similar to that of Theorem 1 in [9], so we omit it.

A. Communication Resource Management Under Given AP selection Decisions

We first consider finding communication resource management variables β under any given AP selection decision $\mathbf{z}$. If $\mathbf{z}$ is given, JAM is equivalent to the following problem.

$$\begin{aligned} \min_{\beta} \quad & \sum_{i\in\mathcal{I}} \Big(\overline{T}^C_i(\mathbf{z},\beta) + \underline{T}^C_i(\mathbf{z},\beta)\Big) \\ \text{s.t.} \quad & \sum_{i\in\overline{\mathcal{O}}_k(\mathbf{z})} \overline{\beta}_{i,k} \le 1, \sum_{i\in\underline{\mathcal{O}}_k(\mathbf{z})} \underline{\beta}_{i,k} \le 1 \text{ for } k \in \mathcal{K} \\ & \overline{\beta}_{i,k} \in [0,1] \text{ for } i \in \overline{\mathcal{O}}_k(\mathbf{z}) \text{ and } k \in \mathcal{K} \\ & \underline{\beta}_{i,k} \in [0,1] \text{ for } i \in \underline{\mathcal{O}}_k(\mathbf{z}) \text{ and } k \in \mathcal{K}. \end{aligned} \tag{21}$$

We use $\beta^*(\mathbf{z}) = \{\overline{\beta}^*_{i,k} | k \in \mathcal{K}, i \in \overline{\mathcal{O}}_k(\mathbf{z})\} \cup \{\underline{\beta}^*_{i,k} | k \in \mathcal{K}, i \in \underline{\mathcal{O}}_k(\mathbf{z})\}$ to denote the optimal solution of (21) under AP selection decision $\mathbf{z}$. The optimal communication resource management decision $\beta^*(\mathbf{z})$ is shown in Lemma 2.

Lemma 2. For any feasible $\mathbf{z}$, optimal communication resource management decision $\beta^*(\mathbf{z})$ is as follows:

$$\overline{\beta}^*_{i,n} = \frac{\sqrt{\overline{c}_i/\gamma_{i,k}}}{\sum_{j\in\overline{\mathcal{O}}_k(\mathbf{z})}\sqrt{\overline{c}_j/\gamma_{j,k}}} \text{ for } k \in \mathcal{K}, i \in \overline{\mathcal{O}}_k(\mathbf{z}), \tag{22}$$

$$\underline{\beta}^*_{i,n} = \frac{\sqrt{\underline{c}_i/\gamma_{i,k}}}{\sum_{j\in\underline{\mathcal{O}}_k(\mathbf{z})}\sqrt{\underline{c}_j/\gamma_{j,k}}} \text{ for } k \in \mathcal{K}, i \in \underline{\mathcal{O}}_k(\mathbf{z}). \tag{23}$$

The proof of Lemma 2 is omitted due to space limitations. The main idea of Lemma 2 is to derive the optimal solution by exploiting KKT conditions.

Substituting β^* into (21), the optimal communication latency under AP selection decision $\mathbf{z}$ is equal to

$$\begin{aligned} &\sum_{i\in\mathcal{I}} \big(\overline{T}^C_i + \underline{T}^C_i\big) = \sum_{k\in\mathcal{K}} \sum_{i\in\overline{\mathcal{O}}_k(\mathbf{z})} \overline{T}^C_i + \sum_{k\in\mathcal{K}} \sum_{i\in\underline{\mathcal{O}}_k(\mathbf{z})} \underline{T}^C_i \\ &= \sum_{k\in\mathcal{K}} \frac{1}{\overline{B}_k} \sum_{i\in\overline{\mathcal{O}}_k(\mathbf{z})} \sqrt{\overline{c}_i/\gamma_{i,k}} \Big(\sum_{j\in\overline{\mathcal{O}}_k(\mathbf{z})} \sqrt{\overline{c}_j/\gamma_{j,k}} \Big) \\ &+ \sum_{k\in\mathcal{K}} \frac{1}{\underline{B}_k} \sum_{i\in\underline{\mathcal{O}}_k(\mathbf{z})} \sqrt{\underline{c}_i/\gamma_{i,k}} \Big(\sum_{j\in\underline{\mathcal{O}}_k(\mathbf{z})} \sqrt{\underline{c}_j/\gamma_{j,k}} \Big). \end{aligned} \tag{24}$$

For the sake of convenience, we introduce some short-formed terms as follows. Let $\mathcal{R} \triangleq \{(k, upload), (k, download) | k \in \mathcal{K}\}$, where each element in set $\mathcal{R}$ is a tuple representing a kind of communication resource. For example, $(k, upload)$ and $(k, download)$ represent the uploading and downloading bandwidth resources of AP k, respectively. For each resource $r \in \mathcal{R}$, there is a weight m_r associated with it. In particular, $m_r = \frac{1}{\overline{B}_k}$ if $r = (k, upload)$ and $m_r = \frac{1}{\underline{B}_k}$ if $r = (k, download)$. For each WD i, let $\mathcal{Z}_i$ be the set of all feasible $\mathbf{z}_i$. In particular, from the constraints of JAM, we have $\mathcal{Z}_i = \big\{\mathbf{z}_i \big| \sum_{k\in\mathcal{K}} \overline{z}_{i,k} = 1, \sum_{k\in\mathcal{K}} \underline{z}_{i,k} = 1, \text{and } \overline{z}_{i,k}, \underline{z}_{i,k} \in \{0,1\}\big\}$. For any given $\mathbf{z}_i \in \mathcal{Z}_i$, $\mathbf{z}_i$ decides the uploading AP and the downloading AP of WD i, and we use $\mathcal{R}_i(\mathbf{z}_i)$ to denote resources that WD i chooses. For example, if WD i chooses AP k_1 and AP k_2 as its uploading and downloading APs respectively, we have $\mathcal{R}_i(\mathbf{z}_i) = \{(k_1, upload), (k_2, download)\}$. Let $p_{i,r}$ be a value corresponding with the pair of WD i and resources r. In particular, $p_{i,r} = \sqrt{\overline{c}_i/\gamma_{i,k}}$ if $r = (k, uploading)$, and let $p_{i,r} = \sqrt{\underline{c}_i/\gamma_{i,k}}$ if $r = (k, downloading)$. In addition, for each $r \in \mathcal{R}$, there is a value p_r, which is a function of $\mathbf{z}$. In particular, $p_r(\mathbf{z}) = \sum_{i\in\overline{\mathcal{O}}_k(\mathbf{z})} \overline{p}_{i,k}$ if $r = (k, upload)$, and $p_r(\mathbf{z}) = \sum_{i\in\underline{\mathcal{O}}_k(\mathbf{z})} \underline{p}_{i,k}$ if $r = (k, download)$.

Then, substituting (24) and the terms defined above into JAM, we have that JAM is equivalent to P2 as follows:

$$\begin{aligned} \min_{\mathbf{z}} \quad & T^C(\mathbf{z}) = \sum_{i\in\mathcal{I}} T^C_i(\mathbf{z}) = \sum_{i\in\mathcal{I}} \sum_{r\in\mathcal{R}_i(\mathbf{z}_i)} m_r p_{i,r} p_r(\mathbf{z}) \\ \text{s.t.} \quad & \mathbf{z}_i \in \mathcal{Z}_i, i \in \mathcal{I}. \end{aligned} \tag{P2}$$

B. Algorithm Design for AP Selection

Next, we propose an algorithm, called generalized Congestion Game Based Algorithm (*CGBA*), for P2 in Algorithm 3. *CGBA* has a parameter $\lambda \ge 0$ that we can tune. We use *CGBA*(λ) to denote *CGBA* with parameter λ. We use $\hat{\mathbf{z}}$ to denote the AP selection decision made by Algorithm 3, and use $\mathbf{z}^* = (\mathbf{z}^*_1, \cdots, \mathbf{z}^*_I)$ to denote the optimal AP selection

Authorized licensed use limited to: SUNY AT STONY BROOK. Downloaded on August 06,2024 at 18:45:00 UTC from IEEE Xplore. Restrictions apply.

Algorithm 3: *CGBA*(λ)

Input: $\overline{B}_k, \underline{B}_k$ for $k \in \mathcal{K}$, $\bar{c}_i, \underline{c}_i$ for $i \in \mathcal{I}$, $\gamma_{i,k}$ for $i \in \mathcal{I}, k \in \mathcal{K}$

Output: A feasible solution to P2: $\hat{\mathbf{z}}$

Initialization: choose $\mathbf{z}_i$ from $\mathcal{Z}_i$ randomly for $k \in \mathcal{K}$;

while $\{\exists i \in \mathcal{I},\ (1-\lambda)T_i^C(\mathbf{z}) > \min_{\bar{\mathbf{z}}_i \in \mathcal{Z}_i} T_i^C(\bar{\mathbf{z}}_i, \mathbf{z}_{-i})\}$ **do**

  $i := \arg\max_{j \in \mathcal{I}} \Big\{ T_j^C(\mathbf{z}) - \min_{\bar{\mathbf{z}}_j \in \mathcal{Z}_j} T_i^C(\bar{\mathbf{z}}_i, \mathbf{z}_{-i}) \Big\}$;

  $\hat{\mathbf{z}}_i := \arg\min_{\bar{\mathbf{z}}_i \in \mathcal{Z}_i} T_i^C(\bar{\mathbf{z}}_i, \mathbf{z}_{-i})$;

  $\mathbf{z} := (\hat{\mathbf{z}}_i, \mathbf{z}_{-i})$;

end

$\hat{\mathbf{z}} := \mathbf{z}$;

decision. In addition, $\mathbf{z}_{-i}$ represents the decision of all WDs except WD i, i.e., $\mathbf{z} = (\mathbf{z}_{-i}, \mathbf{z}_i)$.

In what follows, we analyze the performance of *CGBA*(λ). First, we show that P2 can be interpreted as an exact potential game, as shown in Lemma 3.

Lemma 3. There exists an potential function $P(\mathbf{z})$ such that $T_i^C(\mathbf{z}_i, \mathbf{z}_{-i}) - T_i^C(\bar{\mathbf{z}}_i \mathbf{z}_{-i}) = P(\mathbf{z}_i, \mathbf{z}_{-i}) - P(\bar{\mathbf{z}}_i, \mathbf{z}_{-i})$ holds for all feasible $\mathbf{z}_i$, $\bar{\mathbf{z}}_i$, and $\mathbf{z}_{-i}$, and $P(\mathbf{z}) < T^C(\mathbf{z})$ holds for all feasible $\mathbf{z}$.

The proof of Lemma 3 is standard and can be found in [7], [9], [29], so we omit it. Based on Lemma 3, we have the performance guarantee for *CGBA*(0) as shown in the following theorem.

Theorem 4. *CGBA(0) terminates to a decision* $\hat{\mathbf{z}}$ *with* $2.62 \cdot T^C(\mathbf{z}^*) \geq T^C(\hat{\mathbf{z}})$ *after a finite number of iterations.*

The approximation ratio in Theorem 4 is a special case of Theorem 5, and Lemma 3 implies that the algorithm terminates after a finite number of iterations [30]. The theorem shows that *CGBA*(0) is a 2.62-approximation algorithm. Note that each iteration of *CGBA* takes polynomial time, where the proof is straightforward and omitted due to space limitations. Simulation results show that the time complexity of *CGBA*(0) is linear to I and the average cost under *CGBA*(0) is around $1.02\times$ the optimum. Then, we consider the case that $\lambda > 0$. To show the performance of *CGBA*(λ) with $\lambda > 0$, we first introduce a Lemma appears in [29] as follows.

Lemma 4. Let $\mathbf{z}$ be any feasible decision and $\mathbf{z}^*$ be the optimal decision, then we have $\sum_{i \in \mathcal{I}} T_i^C(\mathbf{z}_i^*, \mathbf{z}_{-i}) \leq \sqrt{T^C(\mathbf{z}) T^C(\mathbf{z}^*)} + T^C(\mathbf{z}^*)$.

Next, the performance guarantee for *CGBA*(λ) is as follows.

Theorem 5. *For* $\lambda \in (0, \frac{1}{8}]$, *CGBA*($\lambda$) *generates a decision* $\hat{\mathbf{z}}$ *with* $T^C(\hat{\mathbf{z}}) \leq \frac{2.62}{1-8\lambda} T^C(\mathbf{z}^*)$ *in at most* $\mathcal{O}\big(\frac{I}{\lambda} \log(\frac{P^0}{P^*})\big)$ *iterations.*

P^0 and P^* are the initial and the minimum value of the potential function $P(\mathbf{z})$, respectively, and both P^0 and P^* are finite positive values. The proof of Theorem 5 is found in our technical report [28].

VI. Numerical Evaluation

In this section, we evaluate the performance of the proposed algorithms under a wide range of settings. We implement our simulations using MATLAB R2021a in a DELL Alienware desktop with 32GB RAM and AMD Ryzen7 2700X Eight-Core Processor running of Windows 10 OS.

A. Simulation Setup

The computing capability of each server is measured by floating-point operations per second (FLOPS), and the capacity of each server i is set to the real-world FLOPS of EC2 instances from [31]. We consider a system with $N = 10$ general-purpose servers (servers) and $M = 5$ FPGAs. Task size of WDs $f_i, i \in \mathcal{I}$ are drawn from the real-world computing complexity (in terms of floating-point operations (FLOPs)) of 6 neural network models, the first six entries of Table 5 in [32]. Similar to [14], $\delta_{i,n}$ is drawn from $[0.5, 1]$. We set the numbers of CLBs and DSP slices are the two bottleneck resources, i.e., $L = 2$, and set $A_{m,1}, m \in \mathcal{M}$ and $A_{m,2}, m \in \mathcal{M}$ to real-world values of different types of FPGAs [33]. The number of CLBs required for implementing a fuzzy neural network varies with network size, where the numbers of CLBs required for implementing fuzzy neural networks with 60 input neurons, 10 to 18 neurons in the hidden layer, and three output neurons are in the range from 5000 to 6000 [34]. We drew the number of CLBs required of WDs from $[4000, 8000]$ and the number DSP slices of WDs from $[20, 40]$. From [3], [4], $t_{i,m}, i \in \mathcal{I}, m \in \mathcal{M}$ are set to 10 to 60 times faster than the optimal average latencies of WDs on servers under the case that $I = 100$ and $M = 0$. The number of WDs varies in different simulations and will be specified in the following.

We assume WDs are located in a square area of $1km \times 1km$, similar to that used in [7]. We divide the $1km \times 1km$ area into six $1/3km \times 1/2km$ subareas, and there are 6 APs located in the center of the 6 subareas. WD i and an AP k can communicate if the distance between them, denoted by $d_{i,k}$, is less than $0.5km$. For convenience, we set the bandwidth of APs to the maximum achievable speed rather than its true physical bandwidth. We randomly draw the uplink bandwidth $\overline{B}_k$ from the set of $[1, 3]$ Gbps and the downlink bandwidth $\underline{B}_k$ from the set of $[2, 5]$ Gbps. $\bar{c}_i, i \in [N]$ and $\underline{c}_i, i \in [N]$ are randomly drawn from $[0.1, 0.5]$ and $[0.2, 1]$ Megabits, respectively [9]. The parameter of bandwidth utilization ratio $\gamma_{i,k}$ corresponding to WD i and AP k is randomly chosen from 0.1 to 1.

B. Baselines

We use three baselines for comparison with *SDPR*. The first baseline, named by *MCMC*, which is similar to the algorithm proposed in [8]. *MCMC* is short for Monte Carlo Markov Chain technique and is similar to the simulated annealing technique. *MCMC* randomly chooses initial states $(\mathbf{x}, \mathbf{y})$ and $\mathbf{z}$ for JOM and JAM, respectively. Then, at each iteration, *MCMC* chooses a neighbor of the previous decision and moves to the neighbor with a probability related to the cost difference of the decisions. Details of *MCMC* can be found

Authorized licensed use limited to: SUNY AT STONY BROOK. Downloaded on August 06,2024 at 18:45:00 UTC from IEEE Xplore. Restrictions apply.

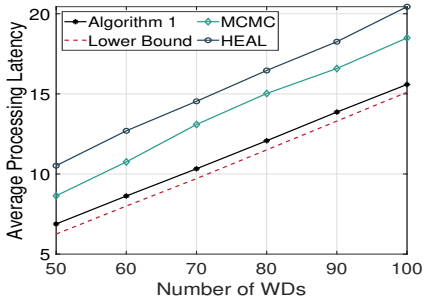

Fig. 2: Average Processing Latency

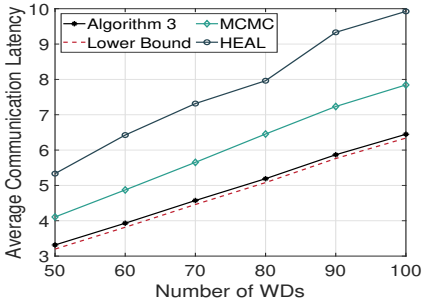

Fig. 3: Average Communication Latency

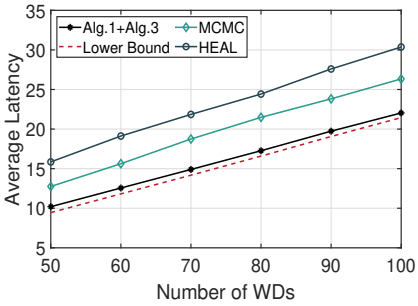

Fig. 4: Average Total Latency

in Algorithm 1 of [8]. The second baseline is named *HEAL* (short for HEuristic ALgorithm), similar to the baseline used in [14]. For JOM, *HEAL* first chooses $\mathbf{y}$ by a greedy algorithm similar to the greedy algorithm [35] for the knapsack problem. In particular, *HEAL* sorts WDs in ascending order of $t_{i,m}$, and WDs in the order take turns to be placed on an FPGA until there is no sufficient space available. Then, *HEAL* chooses $\mathbf{x}$ for WDs that are not placed on $\mathcal{M}$. In particular, *HEAL* chooses the best server n for each i under the assumption that there is no other WD. *HEAL* chooses the optimal computing resource allocation decisions under the selected $(\mathbf{x}, \mathbf{y})$. For JAM, *HEAL* chooses the best uplink and downlink AP for each i under the assumption that there is no other WD and chooses the optimal communication resource allocation decisions under the selected $\mathbf{z}$. Moreover, we also compare the performance of our algorithms and corresponding lower bounds (*LB*). Lower bounds of JOM and JAM are set to the optimal objective values of the convex SDP relaxation problems of JOM and JAM got by the cvx solver, respectively.

C. Simulation Results

We first compare the performance of *SDPR* (Algorithm 1) with that of *MCMC*, *HEAL*, and *LB*. Figure 2 shows that the average latencies of *SDPR* (Algorithm 1), *MCMC*, *HEAL* and *LB* under $I = \{50, 60, \cdots, 100\}$. *SDPR* (Algorithm 1) outperforms *MCMC* and *HEAL* under all the settings of I. From Figure 2, the average ratio of the latency under *SDPR* (Algorithm 1) to the optimal objective value of the SDP relaxation of JOM (lower bound) is around 1.05. As I, the number of WDs, increases, the average latency increases due to congestion. In addition, simulation results show that the time complexity of Algorithm 1 (in terms of the running time) is linear to I, which shows that *SDPR* has good scalability. We then compare the average communication latencies of *CGBA*(0) (Algorithm 3) and that of the baselines under $I = \{50, 60, \cdots, 100\}$. As shown in Figure 3, the communication latency of *CGBA*(0) (Algorithm 3) is lower than that of *MCMC* and *HEAL* under all different settings of I. As I increases, the average latencies of *CGBA*(0) (Algorithm 3) and the baselines increase due to congestion. In addition, we use the objective value of an SDP relaxation of JAM as a lower bound of the optimal objective value. As shown in Figure 3, the average ratio of the latency of *CGBA*(0) (Algorithm 3) to the lower bound is around 1.02. By merging Figure 3 and Figure 2, the total latency (processing latency plus communication latency) is shown in Figure 4.

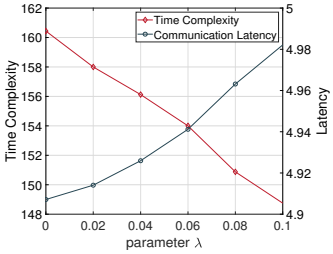

Fig. 5: Trade-off Between Communication Latency and Time Complexity

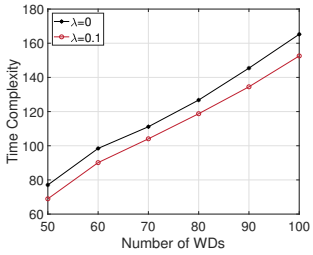

Fig. 6: Time Complexity vs. Number of WD under $\lambda = 0$ and $\lambda = 0.1$

Next, we show the performance of *CGBA* (Algorithm 3) under different λ. In Figure 5, we show the time complexity and communication latency under different settings of λ. From Figure 5, there is a trade-off between the time complexity and the objective value (communication latency) of *CGBA*(λ). As λ increases, the time complexity decreases, and the communication latency increases, which matches the statement in Theorem 5. The time complexity of Algorithm 3 (in terms of the number of iterations) is linear to the number of WDs under different settings of λ as shown in Figure 6.

VII. Conclusion

In this paper, we have studied the joint task offloading, AP selection, and resource allocation problem (JOAM) in heterogeneous edge environments to minimize the overall system latency. We decomposed JOAM into two subproblems, namely JOM and JAM. We designed an algorithm named *SDPR* for JOM based on semidefinite relaxation and proposed a 2.62-approximation algorithm named *CGBA* for JAM. We proved that there is a trade-off between the approximation ratio and the time complexity of *CGBA*. Simulation results have shown that the proposed algorithms outperform the popular baselines and are near-optimal. In particular, the average processing latency under *SDPR* is around 1.05 times the optimum, and the average communication latency under *CGBA* is around 1.02 times the optimum.

Authorized licensed use limited to: SUNY AT STONY BROOK. Downloaded on August 06,2024 at 18:45:00 UTC from IEEE Xplore. Restrictions apply.

REFERENCES

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[2] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "A survey of fpga-based neural network accelerator," *arXiv preprint arXiv:1712.08934*, 2017.

[3] Y. Shen, M. Ferdman, and P. Milder, "Maximizing cnn accelerator efficiency through resource partitioning," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017, pp. 535–547.

[4] P. Milder, "Advanced digital system design and generation [powerpoint slides]." in *ESE 507 Blackboard*, 2021.

[5] J. Weerasinghe, F. Abel, C. Hagleitner, and A. Herkersdorf, "Enabling fpgas in hyperscale data centers," in *UIC-ATC-ScalCom*. IEEE, 2015, pp. 1078–1086.

[6] J. Weerasinghe, R. Polig, F. Abel, and C. Hagleitner, "Network-attached fpgas for data center applications," in *2016 International Conference on Field-Programmable Technology (FPT)*, 2016, pp. 36–43.

[7] S. Jošilo and G. Dán, "Wireless and computing resource allocation for selfish computation offloading in edge computing," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 2467–2475.

[8] X. Ma, A. Zhou, S. Zhang, and S. Wang, "Cooperative service caching and workload scheduling in mobile edge computing," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 2076–2085.

[9] S. Jošilo and G. Dán, "Joint wireless and edge computing resource management with dynamic network slice selection," *IEEE/ACM Transactions on Networking*, pp. 1–14, 2022.

[10] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, 2018.

[11] T. Liu, Y. Zhang, Y. Zhu, W. Tong, and Y. Yang, "Online computation offloading and resource scheduling in mobile-edge computing," *IEEE Internet of Things Journal*, vol. 8, no. 8, pp. 6649–6664, 2021.

[12] H. Sun, F. Zhou, and R. Q. Hu, "Joint offloading and computation energy efficiency maximization in a mobile edge computing system," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 3, pp. 3052–3056, 2019.

[13] A. Al-Shuwaili, O. Simeone, A. Bagheri, and G. Scutari, "Joint uplink/downlink optimization for backhaul-limited mobile cloud computing with user scheduling," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 3, no. 4, pp. 787–802, 2017.

[14] Y. Liu, X. Shang, and Y. Yang, "Joint sfc deployment and resource management in heterogeneous edge for latency minimization," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 8, pp. 2131–2143, 2021.

[15] C. You, Y. Zeng, R. Zhang, and K. Huang, "Asynchronous mobile-edge computation offloading: Energy-efficient resource management," *IEEE Transactions on Wireless Communications*, vol. 17, no. 11, pp. 7590–7605, 2018.

[16] S. I. Venieris and C.-S. Bouganis, "f-cnnx: A toolflow for mapping multiple convolutional neural networks on fpgas," in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, 2018, pp. 381–3817.

[17] Y. Zha and J. Li, "Virtualizing fpgas in the cloud," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 845–858.

[18] O. Knodel, P. Lehmann, and R. G. Spallek, "Rc3e: Reconfigurable accelerators in data centres and their provision by adapted service models," in *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, 2016, pp. 19–26.

[19] P. Chanclou and e. Pizzinat, "Optical fiber solution for mobile fronthaul to achieve cloud radio access network," in *2013 Future Network Mobile Summit*, 2013, pp. 1–11.

[20] S. Jošilo and G. Dán, "Decentralized scheduling for offloading of periodic tasks in mobile edge computing," in *2018 IFIP Networking Conference (IFIP Networking) and Workshops*, 2018, pp. 1–9.

[21] R. K. James, K. P. Jacob, and S. Sasi, "Performance analysis of double digit decimal multiplier on various fpga logic families," in *2009 5th Southern Conference on Programmable Logic (SPL)*. IEEE, 2009, pp. 165–170.

[22] Wikipedia contributors, "5g nr frequency bands — Wikipedia, the free encyclopedia," 2022.

[23] Nvidia Developers, "Multi-process service (mps)." [Online]. Available: https://docs.nvidia.com/deploy/pdf/CUDA_Multi_Process_Service_Overview.pdf

[24] S. Jošilo and G. Dán, "Wireless and computing resource allocation for selfish computation offloading in edge computing," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019, pp. 2467–2475.

[25] T. F. Gonzalez, *Handbook of approximation algorithms and metaheuristics*. Chapman and Hall/CRC, 2007.

[26] P. Wang, C. Shen, A. v. d. Hengel, and P. H. S. Torr, "Large-scale binary quadratic optimization using semidefinite relaxation and applications," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 3, pp. 470–485, 2017.

[27] H. Jiang, T. Kathuria, Y. T. Lee, S. Padmanabhan, and Z. Song, "A faster interior point method for semidefinite programming," in *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, 2020, pp. 910–918.

[28] Y. Liu, Y. Mao, Z. Liu, and Y. Yang, "Technical report for paper "joint task offloading and resource allocation in heterogeneous edge environments"," [Online]. Available: https://drive.google.com/drive/folders/1w8YmSQfYURBR3n5xBw0uqu6asTHjC-KD?usp=sharing.

[29] B. Awerbuch, Y. Azar, A. Epstein, V. S. Mirrokni, and A. Skopalik, "Fast convergence to nearly optimal solutions in potential games," in *Proceedings of the 9th ACM conference on Electronic commerce*, 2008, pp. 264–273.

[30] Y. H. Chew, B.-H. Soong *et al.*, *Potential game theory*. Springer, 2016.

[31] J. Emeras, S. Varrette, V. Plugaru, and P. Bouvry, "Amazon elastic compute cloud (ec2) versus in-house hpc platform: A cost analysis," *IEEE Transactions on Cloud Computing*, vol. 7, no. 2, pp. 456–468, 2019.

[32] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 6848–6856.

[33] Virtex Developers, "Virtex-5 fpga feature summary," [Online]. Available: https://inst.eecs.berkeley.edu/~cs150/sp09/Lecture/lec03-fpga.pdf.

[34] S. R. Chowdhury and H. Saha, "Development of a fpga based fuzzy neural network system for early diagnosis of critical health condition of a patient," *Computers in biology and medicine*, vol. 40, no. 2, pp. 190–200, 2010.

[35] G. Dantzig, *26. Discrete-Variable Extremum Problems*. Princeton University Press, 2016.

Authorized licensed use limited to: SUNY AT STONY BROOK. Downloaded on August 06,2024 at 18:45:00 UTC from IEEE Xplore. Restrictions apply.