EFFECT: Energy-efficient Fog Computing Framework for Real-time Video Processing

Xiaojie Zhang, Amitangshu Pal, Saptarshi Debroy
City University of New York, Temple University
Email: *xzhang6@gradcenter.cuny.edu, amitangshu.pal@temple.edu, saptarshi.debroy@hunter.cuny.edu*

***Abstract*—Energy efficient task offloading within a fog computing environment comprising of end-devices and edge servers remains a challenging problem to solve, especially for real-time video processing applications due to such tasks' strict latency deadline demands. In this paper we propose an Energy-efficient Fog Computing framework (EFFECT) for real-time applications within mission-critical use cases. The proposed framework runs a Unified Resource Broker (URB) that implements: a) centralized sub-channel and transmission power allocation as well as end-device/edge server computation speed allocation algorithms, along with b) distributed multi-device, multi-server task offloading game based Directed Acyclic Graph (DAG) partition and edge server selection algorithms. The framework is designed, developed, implemented, and evaluated on an Amazon EC2 virtual testbed built using Apache Storm, which is a distributed computing platform. The results from the testbed experiments along with realistic simulations validate the utility of EFFECT task offloading strategy in minimizing energy consumption yet satisfying latency deadlines.**

I. Introduction

In order to provide rapid situational-awareness to mission-critical use cases (e.g., emergency response and tactical situations), reconnaissance missions employ *fog* computing environments. Such fog environments host real-time video applications where: i) raw video data with complex and real-time processing needs are captured by speciality end-devices (e.g., drones, robots); ii) speciality on-premise (e.g., hosted on vehicles) edge nodes/units equipped with wireless base stations/access points (AP) and computation servers (CPU/GPU) that are deployed to process the raw video data on-demand, iii) ground consumers of the data (e.g., tactical or first responder units) visualize the processed video on their hand-held devices as shown in Fig. 1. Thus in recent times, the adoption of such siloed and independent fog environment based real-time video processing applications for reconnaissance missions over traditional cloud-based solutions is motivated by: a) often unreliable network connectivity between the mission site and cloud data center during emergency situations and b) potential long end-to-end delays in enterprise network when supporting data-intensive video processing application workflows.

Challenges of video processing at fog: However, resource management in such fog environments in terms of network, compute, and energy resource allocation for real-time video applications is non-trivial and offers the following unique challenges. **Firstly**, traditional fog/edge computing resource allocation techniques [1], [2] recommend offloading all compute-intensive tasks to edge servers for end-devices' energy preservation. However, inherent fluctuations in wireless channel quality caused by phenomena such as, multi-path propagation, shadowing, and fading result in varying end-to-end latency. This in turn adds to the transmission cost of task offloading as well as end-devices' energy expenditure - thus outweighing offloading energy preservation benefits. **Secondly** reconnaissance missions often involve multiple agencies/stakeholders (different tactical units or first resoponder agencies) demanding resources from the common resource pool with little to no cooperation among them. This results in video applications selfishly trying to preserve latency deadline and involved devices seeking to minimize their own energy consumption. This in turn results in limited fog resources (unlike unlimited cloud resources) to be used inefficiently. **Finally,** most complex video applications require multi-stage computation where joint optimization of energy-efficiency and deadline satisfaction is non-trivial especially for multi-resource environments due to their diverging nature. Although there are significant strides made in energy-efficiency within fog/edge environments, few efforts have addressed the joint optimizing.

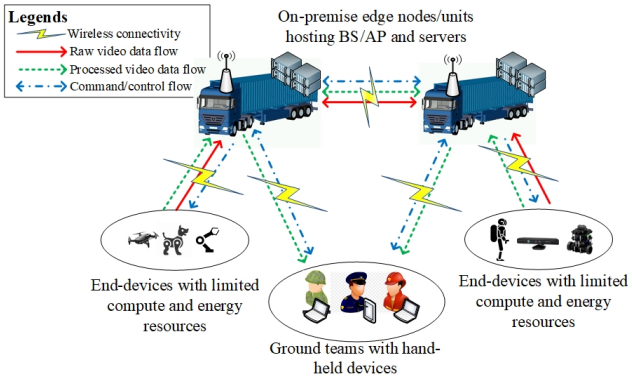

Fig. 1: An exemplary fog computing based real-time video processing application

Related work: Video analytics is one of the emerging use cases for deploying and utilizing fog/edge resources. In VideoStrom [3] and VideoEdge [4], the trade-off between query accuracy and resource demand is extensively studied. The authors in [5] consider bandwidth-efficiency in real-time drone video analysis. In [6], the authors propose a visual fog-cloud computing architecture for 3D visualization for incidence support. Works, such as [7], [8] propose learning based framework for video processing at the network edge, while [9] develops an online algorithm for joint configuration adaptation and bandwidth allocation. *However, the above works use pre-configured resource requirement as a problem evaluation metric which does not always capture the heterogeneous geo-distributed network resources.*

Based on task offloading model in fog/edge systems, most of the related literature can be grouped into following three categories. First, works [1], [2] that consider task offloading

as a deterministic problem, i.e., offload or do not offload. Reference [10] proposes a framework that determines whether to execute the task on the edge device or in the cloud. Second group of papers [11]–[14] formulates their problems in "computing while transmitting" paradigm by enabling local computation on devices. In addition to the above, the third group of works [15]–[17] expresses the tasks as Directed Acyclic Graphs (DAG) consisting of multiple computation components where unlike previous models, task components are placed across the edge servers. *However, such works do not aim at solving resource allocation problems for multi-stage computations in multi-server environments which is precisely our problem environment*.

Game theory as a powerful tool for distributed resource allocation has been adopted in recent times in works, such as [18]–[22]. Authors in [18], [19] propose decentralized computation offloading games based on network resource sharing. Others, such as [20]–[22] consider both CPU and network resource limitations within edge environments for their problem formulation. *However, to the best of our knowledge, none of the aforementioned works address both the multi-resource (e.g., sub-channel, CPU frequency and transmission power) allocation problem and multi-component task placement problem across multiple servers at the same time.*

Our contribution: In this paper, we propose EFFECT, an Energy-efficient Fog Computing framework to support real-time video processing applications. The proposed EFFECT framework runs a centralized Unified Resource Broker (URB) within the fog environment. This URB solves the aforementioned challenges by decoupling the energy-efficiency problem into two interconnected sub-problems. For the first sub-problem, EFFECT employs a *centralized* resource provisioning algorithm to optimize: i) sub-channel and transmission power allocation at the end-devices and ii) CPU speed allocation at both end-device (local) and servers (remote) hosted at the edge nodes/units. The algorithm runs on individual edge servers using 'Helper' modules and the optimized results are passed to the second sub-problem. Here, EFFECT implements a *distributed* multi-device and multi-server task offloading game aimed at tackling multi-stage computation partition and server selection problems. For this, end-devices running competing video applications obtain their favorite task offloading strategy from the first sub-problem and then propose a strategy update request to the URB. In EFFECT framework, an energy-efficiency based priority mechanism is adopted to select and accept the proposed update requests. The framework only accepts the request with highest priority and modifies the global strategy profile upon that request. Finally, the URB broadcasts the updated global strategy profile to edge servers when the EFFECT framework runs the first sub-problem algorithm again. This inter-exchange terminates when the system reaches a Nash Equilibrium (NE) - thus achieving optimal energy-efficiency.

We design, develop, implement, and evaluate the EFFECT framework virtual testbed on Amazon EC2 using Apache Storm [23] distributed computing platform. We run competing video processing applications on the testbed in order to evaluate the performance of EFFECT framework's resource allocation. The results show the EFFECT framework's success in terms of optimal task offloading decision-making. Compared to fair allocation, the EFFECT algorithm achieves considerable energy consumption by jointly considering the natures of applications and their real-time requirements. We also perform extensive simulations in order to verify the schedulability, the benefits of partial task offloading, and system convergence under a large number of applications and varying edge resources. These results demonstrate the high energy efficiency of the EFFECT framework in handling unpredictable system environment.

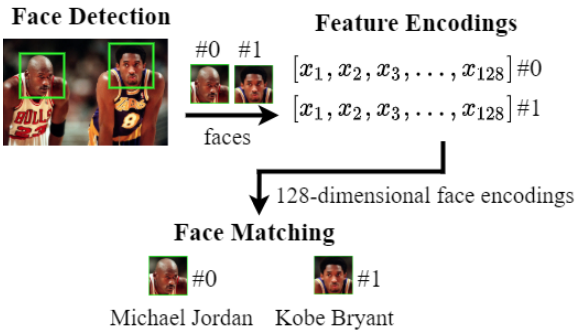

Fig. 2: An exemplar linear topology for face recognition used in this work

Paper organization: The rest of the paper is organized as follows. Section II proposes the system model and problem formulation. Section III presents the centralized resource allocation algorithm. Section IV discusses the distributed task offloading game. Section V discusses EFFECT evaluation. Section VI concludes the paper.

II. System Model and Problem Formulation

EFFECT is a fog computing framework which consists of a centralized URB, a set of N devices $N = \{1, 2, ..., N\}$, and a set of K edge nodes/units $K = \{1, 2, ..., K\}$. We assume that each edge node contains 1 computing server and 1 AP that connects to end-devices via wireless and has fibre connectivity with other APs in other edge nodes. In this paper, we describe the offloading decision (edge node, i.e., server selection) by end-device as $a_n \in A = \{0, 1, ..., K\}$ with the following definition

$$a_n = \begin{cases} 0 & \text{if device } n \text{ executes task locally} \\ k & \text{if device } n \text{ executes task on edge server k} \end{cases}$$

In order to simplify the expression, we define symbol $I_{\{a_n = x\}} \in \{0, 1\}$ as the event indicator $\forall x \in A$. $I_{\{a_n = x\}} = 1$ signifies decision $a_n = x$; otherwise $I_{\{a_n = x\}} = 0$.

A. Task Partition

In this paper, we only consider recurring tasks with linear topology as any concurrent or parallel task can be serialized by works such as [16]. The task topology is described by a DAG G_n. We assume that task execution constraint D_n (in seconds) and task period (recurring nature) are the same. We denote M_n as the number of jobs (vertices) in task G_n. One such exemplar linear topology is shown in Fig. 2 where the DAG of a face recognition [24] application workflow contains three jobs ($M_n = 3$) which are executed sequentially, viz., 1) face detection, 2) feature encoding, and 3) face matching.

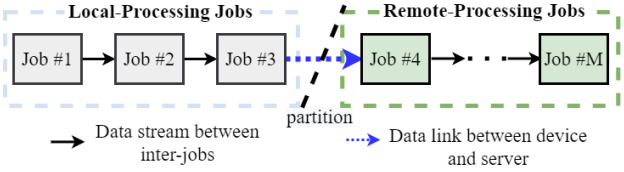

Fig. 3: An example of task partition. The first three jobs are executed locally and the remaining jobs (from job 4 to job M) are processed at the edge server

The task DAG can be partitioned into local-processing jobs and remote-processing jobs to achieve specific cost minimization. An example of task partition is shown in Fig. 3. Here, the device runs the first three jobs and sends the intermediate data (e.g., face encoding list from Fig. 2) to the edge server. After receiving the intermediate data, the edge server executes the rest of jobs. Obviously, a task with multiple jobs can be partitioned at different locations/points. A partition with m as the chosen partition location signifies that only the last m jobs are executed on the edge server with $m = M_n$ indicating full task offloading and $m = 0$ indicating local-only computation.

In this paper, we only focus on partition locations that can significantly reduce the energy spent on data transmission while the energy consumption of local computation is small. For example, in face recognition application from Fig. 2, we should let the device run the face detection job and the edge server should execute feature encoding and face matching after receiving the facial images sent by the device. Since the device only needs to transmit the detected facial images, this method greatly reduces the transmission cost. In terms of saving time and energy, the other option (i.e., running face detection and feature encoding on the device) can be considered useless.

B. Execution Profiles

In this work, we assume that the end-devices have perfect knowledge of their tasks and available partitions. With a given partition location m, we denote $X_{n,m}$ and $Y_{n,m}$ as the computation complexity of processing the remote and local jobs, which are measured by the number of CPU cycles. We also define $Z_{n,m}$ as the size of data for transmission (in bits). The devices store such information in a set called execution profiles which is defined as follows:

$$J_n = f\,(X_{n,m}, Y_{n,m}, Z_{n,m})\; j\; m \;2\; [0, M_n],\; Z_{n,m} \quad Z_{n,0} g$$

where the constraint $Z_{n,m} \quad Z_{n,0}$ is used to filter useless partition locations as we discussed before. In EFFECT framework, end-devices are first randomly connected to a nearby AP which is used upload their execution profiles along with the execution constraint. From this point the AP forwards the execution profiles to the URB. Once the URB receives all the profiles, it performs resource allocation to minimize the system energy consumption while satisfying the execution constraints.

C. Communication Model

Implemented within a stand-alone fog environment, EFFECT enforces Orthogonal Frequency-Division Multiple Access (OFDMA) communication system where the APs have certain amount of sub-channels to be used for application data transmission. Once a device decides to offload its task to an edge server (i.e. $I_{fa_n=kg} = 1$), the URB assigns several sub-channels through the AP for data transmission between the device and the AP. We denote C_k as the number of available sub-channels at AP k. We assume that the AP has perfect knowledge of sub-channel gain $h_{n,k}$. Therefore, the aggregated data rate for data transmission from device to AP is modeled as:

$$r_{n,k} = b_{n,k} B_0 \log_2 \left(1 + p_{n,k} h_{n,k}^2 / N_0\right) \tag{1}$$

where B_0 denotes the sub-channel bandwidth, $b_{n,k}$ is the number of sub-channels assigned to device n by AP k, $p_{n,k}$ is the transmission power used on a single sub-channel (i.e. the total transmission power is $b_{n,k} p_{n,k}$), and N_0 is the white noise power spectral density. Since one sub-channel can only be used by one device at any time instance, $\sum_{n=1}^{N} b_{n,k} \quad C_k$ holds for all APs with the fog environment. The energy spent on transmitting data from device to AP under given execution profile $J_{n,m}$ and sub-channel allocation $b_{n,k}$ is expressed as:

$$E^D_{n,k}(J_{n,m}) = b_{n,k} p_{n,k} Z_{n,m} / r_{n,k} \tag{2}$$

D. Computation and Energy Model

***Remark** 1:* With EFFECT, when a device decides to execute a task locally (i.e., $I_{fa_n=0g} = 1$ and $m = 0$), the minimal energy consumption $w.r.t.$ the task latency constraint can be computed by $E^L_n = \kappa\,(Y_{n,0})\,[Y_{n,0}/D_n]^2$ where $\kappa = 10^{-28}$ J/cycle is a constant related to the chip architecture [11], [25]. In EFFECT, a task can be executed locally if and only if the maximum CPU speed of device satisfies $Y_{n,0} \quad f^{\max}_n D_n$.

In EFFECT, the computation follows the model described in [25] that allows end-devices to adjust their CPU speed for energy saving based on Dynamic Voltage Scaling (DVS) technique. When a device select task offloading $I_{fa_n=kg} = 1$ to server k and set $f^N_{n,k}$ as the device CPU speed, the energy consumption for local computation is $E^C_{n,k}(J_{n,m}) = \kappa Y_{n,m} [f^N_{n,k}]^2$. The total energy consumption is the sum of transmission cost and computation cost, which can be expressed as:

$$E_{n,k}(J_{n,m}) = E^D_{n,k}(J_{n,m}) + E^C_{n,k}(J_{n,m}) \tag{3}$$

E. Problem Formulation

The objective of EFFECT framework is to find the best execution profile and the optimal offloading decision. We also seek the optimal resource allocation strategy for devices and edge servers. Therefore, EFFECT's energy-aware multi-device and multi-server task offloading optimization problem can be represented as:

$$\min_{R_{n,a_n},\, J_{n,m}} \sum_{n=1}^{N} \sum_{k=1}^{K} E_{n,k}(J_{n,m}) I_{fa_n=kg}$$

$$\text{s.t. } \mathbf{C1:} \sum_{n=1}^{N} b_{n,k} I_{fa_n=kg} \quad C_k,\; 8k \;2\; K$$

$$\mathbf{C2:} \sum_{n=1}^{N} f^K_{n,k} I_{fa_n=kg} \quad F_k,\; 8k \;2\; K$$

$$\textbf{C3:}\sum_{k=1}^{K}\left(\frac{Y_{n,m}}{f^{N}_{n,k}}+\frac{X_{n,m}}{f^{K}_{n,k}}+\frac{Z_{n,m}}{r_{n,k}}\right)I_{fa_n=kg}\quad D_n, 8n\ 2\ N$$

$$\textbf{C4:}\sum_{k=1}^{K}E_{n,k}(J_{n,m})I_{fa_n=kg}\quad E^{L}_{n},\ 8n\ 2\ N$$

$$\textbf{C5:}\frac{Y_{n,0}}{D_n}I_{fa_n=0g}\quad f^{\max}_{n},\ 8n\ 2\ N$$

$$\textbf{C6:}a_n\ 2\ f0,1,2,...,Kg,\ J_{n,m}\ 2\ J_n \qquad \textbf{(P1)}$$

where $R_n = f f^{N}_{n,k}, p_{n,k}, f^{K}_{n,k}, b_{n,k}g$ denotes the resource allocation profile and $f^{K}_{n,k}$ is the CPU speed allocated to device n from server k. In (**P1**), constraints **C1** and **C2** limit the number of sub-channels and the CPU capacity used by the edge servers, constraints **C3** and **C4** specify the execution deadline and the energy saving constraints for an offloading decision and the constraint **C5** represents ***Remark 1***.

F. Problem Decoupling: *CRA* and *DSM*

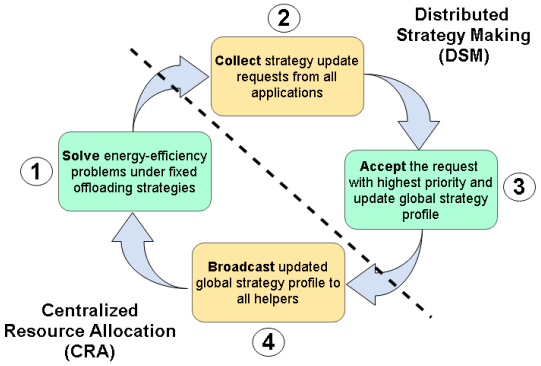

Fig. 4: Interrelationship between **CRA** and **DSM** in EFFECT

EFFECT's multi-device and multi-server task offloading optimization problem (**P1**) is non-trivial to solve because of its Mixed-Integer Nonlinear Programming (MINLP) nature - the number of sub-channels, the execution profile selection, and the task offloading decision making are all discrete integers. It is well known that solving such NP hard problems with closed-form expressions is very challenging. Thus, in EFFECT, we divide the optimization problem into two sub-problems:

$$\overbrace{\underset{a_n,J_{n,m}}{\operatorname{argmin}}\left\{\underbrace{\underset{R_n}{\operatorname{argmin}}\sum_{n=1}^{N}E_{n,k}(J_{n,m})I_{fa_n=kg},\ 8k\ 2\ K}_{\textbf{CRA}:\text{Centralized}}\right\}}^{\textbf{DSM}:\text{Distributed}} \qquad (4)$$

The outer sub-problem of Distributed Strategy Making (**DSM**) is a distributed game where devices are modeled as selfish in nature and always tending to select the best offloading strategy to minimize their energy consumption. Whereas the inner sub-problem of Centralized Resource Allocation (**CRA**) consists of K independent but identical sub-sub-problems that are centralized (at each edge node) to minimize the overall energy consumption of offloaded tasks. The interrelationship between **DSM** and **CRA** is shown in Fig. 4. The solution to **P1** is obtained by alternatively performing **DSM** and **CRA** until convergence. Next, we discuss sub-problems CRA and DSM individually.

III. CRA: Centralized Resource Allocation With Fixed Offloading Strategy

The sub-problem **CRA** represents a group of offloaded tasks denoted by $N_k = fn\ j\ I_{fa_n=kg} = 1, n\ 2\ Ng$ - the objective being the minimization of total energy consumption by devices in N_k. In this section, we will first remove the integer constraints **C1** and **C6** from (**P1**) in order to convert sub-problem **CRA** into a convex optimization problem. To this end, EFFECT first pre-allocates several sub-channels to each offloaded task and generates an initial resource allocation profile R^0_n. After that, EFFECT performs a heuristic sub-channel allocation that dynamically assigns sub-channels to the most desirable devices and continuously updates $R^0_n\ !\ R^t_n$ until convergence.

A. Problem Transformation: *CRA*

We denote auxiliary variables $\tau^{D}_{n,k}$ and $\tau^{C}_{n,k}$ as the data transmission and edge computation times and define function $g(r) = N_0\left(2^{r_{n,k}/b_{n,k}B_0}\quad 1\right)$ that is a monotonically decreasing function in terms of achievable data rate $r_{n,k}$. Based on Eq. (1), the transmission power can be expressed as:

$$p_{n,k} = \textbf{min}\left\{g(Z_{n,m}/\tau^{D}_{n,k})/h^2_{n,k}, p^{\max}_{n}/b_{n,k}\right\} \qquad (5)$$

where the maximum transmission power for each device is $p^{\max}_n$. Now the sub-problem **CRA** transforms to:

$$\min_{R_n}\sum_{n2N_k}\left(\kappa Y_{n,m}\left[f^{N}_{n,k}\right]^2+\frac{b_{n,k}}{h^2_{n,k}}g\left(\frac{Z_{n,m}}{\tau^{D}_{n,k}}\right)\tau^{D}_{n,k}\right)$$

$$\text{s.t. }\textbf{C1:}\ \frac{b_{n,k}}{h^2_{n,k}}g\left(Z_{n,m}/\tau^{D}_{n,k}\right)\quad p^{\max}_{n},\ 8n\ 2\ N_k$$

$$\textbf{C2:}\ \sum_{n2N_k}X_{n,m}/\tau^{C}_{n,k}\quad F_k$$

$$\textbf{C3:}\ \tau^{D}_{n,k}+\tau^{C}_{n,k}+Y_{n,m}/f^{N}_{n,k}\quad D_n,\ 8n\ 2\ N_k \qquad \textbf{(P2)}$$

The Hessian matrix of (**P2**) being positive semi-definite makes it a convex problem $w.r.t.$ resource allocation profile R_n. Therefore, applying Karush-Kuhn-Tucker (KKT) conditions yields the optimal resource allocation profile R_n. Introducing Lagrangian multipliers $\boldsymbol{\lambda}$, $\boldsymbol{\nu}$ and $\boldsymbol{\mu}$, the Lagrange function of problem **CRA** can be expressed as:

$$L(N_k) = \sum_{n2N_k}\left(\kappa Y_{n,m}[f^{N}_{n,k}]^2+\frac{b_{n,k}}{h^2_{n,k}}g\left(\frac{Z_{n,m}}{\tau^{D}_{n,k}}\right)\tau^{D}_{n,k}\right)$$
$$+\lambda_{n,k}\left(\frac{b_{n,k}}{h^2_{n,k}}g\left(\frac{Z_{n,m}}{\tau^{D}_{n,k}}\right)\quad p^{max}_{n}\right)+\nu_k\left(\sum_{n2N_k}\frac{X_{n,m}}{\tau^{C}_{n,k}}\quad F_k\right)$$
$$+\mu_{n,k}\left(\tau^{D}_{n,k}+\tau^{C}_{n,k}+\frac{Y_{n,m}}{f^{N}_{n,k}}\quad D_n\right) \qquad (6)$$

B. Transmission Time and Transmission Power

For transmission time and power optimization, we first define $f(x) = g(x)\quad xg'(x)$. Now, if $\tau^{D}_{n,k}{}^{(\ )}$ denotes the optimal data transmission time, then:

$$f\left(Z_{n,m}/\tau^{D}_{n,k}{}^{(\ )}\right) = \quad \mu_{n,k}h^2_{n,k}/\left(1+\lambda_{n,k}\right)b_{n,k}$$

Based on the Lambert function W_0 and similar solutions proposed in works, such as [11], [26], the relationship between data rate and transmission time can be stated as:

$$f^{-1}(y) = Z_{n,m}/\tau_{n,k}^{D}{}^{(t)} = B\left[W_0\left(y + N_0/\quad N_0 e\right) + 1\right]/\ln(2)$$

Since a device has to transmit at least $Z_{n,m}$ bits of data during $\tau_{n,k}^{D}{}^{(t)}$, it provides the lower bound of $\tau_{n,k}^{D}{}^{(t)}$ when the device uses its maximum transmission power p_n^{max} resulting in maximum data rate $r_{n,k}^{max}$. Thus, the optimal transmission time can be stated as:

$$\tau_{n,k}^{D}{}^{(t)} = \mathbf{max}\left\{\frac{Z_{n,m}\quad \ln(2)}{B\left[W_0\left(\frac{\mu_{n,k}h_{n,k}^2}{(1+\lambda_{n,k})b_{n,k}N_0 e}\quad \frac{1}{e}\right)+1\right]}, \frac{Z_{n,m}}{r_{n,k}^{max}}\right\} \tag{7}$$

Upon obtaining this optimal transmission time, the transmission power can be computed from Eq. (5).

C. CPU Speed Allocation

Similar to the analysis in III-B, the optimal CPU speed at an end-device can be expressed as:

$$f_{n,k}^{N}{}^{(t)} = \mathbf{min}\left\{\left[\mu_{n,k}/2\kappa\right]^{\frac{1}{3}}, f_n^{\max}\right\} \tag{8}$$

Whereas, the optimal CPU time at an edge server is:

$$\tau_{n,k}^{C}{}^{(t)} = \left[\nu_k X_{n,m}/\mu_{n,k}\right]^{\frac{1}{2}} = X_{n,m}/f_{n,k}^{K}{}^{(t)} \tag{9}$$

It is evident that the edge server should allocate all its CPU to the offloaded tasks resulting in tasks having higher computation requirement $X_{n,m}$ getting more resources. Therefore, in Eq. (9), we can substitute multiplier ν_k with $\sum_{n \in N_k} f_{n,k}^{K}{}^{(t)} = F_k$. Thus, the optimal CPU speed allocated to each offloaded task in an edge server can be calculated as $f_{n,k}^{K}{}^{(t)} = \frac{\sqrt{X_{n,m}\mu_{n,k}}}{\sum_{n \in N_k}\sqrt{X_{n,m}\mu_{n,k}}} F_k$.

D. Heuristic Sub-channel Allocation

The EFFECT framework provides a low complexity sub-channel allocation by periodically running a heuristic with an interval of T_{ch}. To do that each device n calculates the gain function for getting the next sub-channel from AP k, which is defined as:

$$G_{n,k} = P(b_{n,k}) \quad P(b_{n,k}+1) \tag{10}$$

where

$$P(b_{n,k}) = \frac{b_{n,k}}{h_{n,k}^2} g\left(\frac{Z_{n,m}}{\tau_{n,k}^{D}}\right)\tau_{n,k}^{D}$$

The algorithm runs through all the sub-channel and the device which has the highest $G_{n,k}$, ends up receiving a sub-channel. The sub-channel allocation strategy can thus be stated as:

$$b_{n,k}^{(t)} = \begin{cases} b_{n,k}^{(t)} + 1 & \text{if } n = \underset{n}{\text{argmax}}(G_{n,k}) \\ b_{n,k}^{(t)} & \text{otherwise} \end{cases} \tag{11}$$

E. Joint Resource Energy Efficient Allocation

Once R_n is obtained, we can use a sub-gradient method to update $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$. The update rules are:

1) For $\lambda_{n,k}$, when $b_{n,k}p_{n,k} \neq p_n^{max}$, we set $\lambda_{n,k} = 0$; otherwise

$$\lambda_{n,k}^{t+1} = \left[\lambda_{n,k}^{t} + \quad_{\lambda}(t)\left(\frac{b_{n,k}}{h_{n,k}^2} g\left(\frac{Z_{n,m}}{\tau_{n,k}^{D}}\right) \quad p_n^{max}\right)\right]^{+} \tag{12}$$

2) For $\mu_{n,k}$, it follows

$$\mu_{n,k}^{t+1} = \left[\mu_{n,k}^{t} + \quad_{\mu}(t)\left(\tau_{n,k}^{D} + \tau_{n,k}^{C} + \frac{Y_{n,m}}{f_{n,k}^{N}} \quad D_n\right)\right]^{+} \tag{13}$$

As shown in Eq. (12) and (13) (t) is the diminishing step size. The joint resource and energy allocation algorithm is described in **Algo. 1** and has a complexity of $O(1/\epsilon^2)$.

Algorithm 1: Joint Resource and Energy Allocation

```
Given a list of offloaded tasks N_k, stop point ε, step control t = 0
Initialize multipliers {λ_n | n ∈ N_k} and {μ_n | n ∈ N_k}
Initialize resource allocation profiles {R_n | n ∈ N_k}
while True do
    stop = True
    for each task n ∈ N_k do
        Update λ^t_{n,k} and μ^t_{n,k} based on (12) and (13)
        Update R^t_n
        if completion time of task n − D_n > ε then
            stop = False
    if stop then
        break
    if t % T_ch == 0 then
        Run Heuristic Sub-channel Allocation from (11)
    t = t + 1
return {R^t_n | n ∈ N_k}
```

IV. DSM: Multi-device and Multi-server Distributed Task Offloading Game

The EFFECT framework solves the sub-problem **DSM** by performing a multi-device and multi-server task offloading game. It considers the following characteristics of the task offloading game: 1) The edge servers' objective is to reduce the overall system energy consumption i.e., with sub-problem **CRA**, EFFECT minimizes the energy consumption of all end-devices in a centralized manner and 2) The devices are inherently selfish; thus they are only concerned about reducing their own energy consumption i.e., with sub-problem **DSM**, EFFECT minimizes the energy consumption of all devices in a distributed manner.

The proposed task offloading game is a two-sided matching game where devices always offload their tasks to the edge servers based on their current preferences. Thus, any unilateral strategy update may affect the preferences of all devices and may lead to continuous updating of personal strategies. In EFFECT, we aim to find a Nash equilibrium (NE) for the proposed matching game and analyze the convergence of such strategy update process.

A. Game Formulation

In order to formulate a game for task offloading sub-problem **DSM**, we use a 3-dimensional strategy space which is defined as $S = [A, J, R]$. The strategy at update iteration t by device n is denoted by $s_n(t) = [a_n(t), J_{n,m}(t), R_n(t)]$. It indicates the offloading decision $a_n(t)$, the execution profile $J_{n,m}(t)$, and the resource allocation profile $R_n(t)$ that are made by device n. In EFFECT, a global strategy profile is generated and maintained by the framework URB, which is defined as $S(t) = \{s_0(t), s_1(t), .., s_N(t)\}$. We define $s_{-n}(t)$ as the set of offloading strategies made by all other devices except for device n. Thus, the energy consumption function for device n can be formulated as:

$$\eta_n(s_n(t), s_{-n}(t)) = \begin{cases} E_n^L & \text{if } I_{\{a_n(t)=0\}} = 1 \\ E_{n,k}(J_{n,m}(t)) & \text{if } I_{\{a_n(t)=k\}} = 1 \end{cases} \tag{14}$$

The multi-device and multi-server task offloading problem is formulated as a strategic game with individual utilities calculated by Eq (14). Given $s_{-n}(t)$, each device chooses a suitable strategy $s_n(t)$ to minimize its own energy consumption (in a selfish manner) where $\forall n \in N$,

$$s_n(t) = \underset{s_n(t) \in S}{\operatorname{argmin}} \eta_n(s_n(t), s_{-n}(t)) \tag{P3}$$

It is to be noted that (**P3**) can be treated as N parallelized sub-sub-problems sharing the same global strategy profile $S(t)$, thereby significantly reducing the computation overhead.

***Remark** 2:* In any energy-aware framework, having devices run optimization algorithms can be counterproductive. Thus EFFECT employs **Helper** processes that can concurrently make task offloading decisions for all devices. Helpers (as shown in Fig. 5) reside within edge nodes and provide energy optimization service for a set of devices. To avoid redundant computation during game iteration, a central database is used for caching historical optimization outcomes (from **Algo. 1**) and is shared among all the Helpers.

In accordance to ***Remark** 2*, EFFECT framework URB assigns a Helper process to each end-device (randomly or from the nearest edge node) and applies the Best Response Strategy algorithm (in response to $s_{-n}(t)$) on Helpers in order to find the best strategy $s_n(t)$ for individual devices. At the same time, the URB broadcasts the current global strategy profile $S(t)$ to all the Helpers. The Best Response Strategy algorithm running on Helpers is described in **Algo. 2**.

Algorithm 2: Best Response Strategy For Device n

```
Receive strategies made by other devices s_{-n}(t) from URB;
  Initialize minimal energy consumption E = E_n^L
for each edge server k ∈ K do
    Get the list of offloaded tasks on server k: N_k
    for each J_{n,m} ∈ J_n do
        Run Algo. 1 to find the best R_n, then calculate E_{n,k}
        if E > E_{n,k} then
            E = E_{n,k}, s_n(t) = [k, J_{n,m}, R_n]
return s_n(t)
```

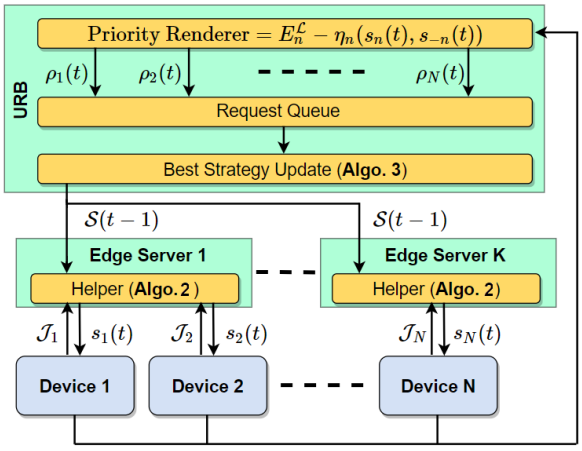

Fig. 5: The logical diagram of proposed distributed multi-device and multi-server task offloading game components within the EFFECT framework

B. Nash Equilibrium and Convergence

Here we explore the Nash Equilibrium (NE) characteristics for EFFECT's proposed multi-device and multi-server task offloading game. As mentioned earlier, EFFECT assumes that the end-devices belonging to individual teams/agencies are selfish in nature and are only concerned about their own energy consumption.

Theorem 1: The multi-device and multi-server task offloading game with a global cost function $\phi(S)$ defined in Eq. (15) is a potential game which always has a NE.

$$\phi(S) = \sum_{n=1}^{N} \left(\kappa Y_{n,m} [f_{n,k}^N]^2 + p_{n,k} \frac{Z_{n,m}}{B_0 \log_2 \left(1 + \frac{p_{n,k} h_{n,k}^2}{N_0}\right)} \right) \tag{15}$$

Proof: The details of the proof is trivial and so is skipped for the sake of brevity. For now, we will assume that the game will always have a NE S^* and the finite improvement property (FIP) [18], [19]. ■

C. Priority Based Request Update Policy

Algo. 3 describes the proposed multi-device and multi-server offloading strategy that the EFFECT framework applies to find the improvement path that leads to the NE (based on FIP) built upon **Algo. 2**. In **Algo. 3**, steps 6-9 are distributed on individual Helpers (i.e., **DSM**) that send update requests $s_n(t)$ to the URB. However, only one request is accepted by the URB at each iteration among all end-devices' requests that want to change their offloading decisions ($s_n(t-1) \neq s_n(t)$). Compared to the random request selection policy used in [18], [19], EFFECT rather adopts an energy-efficiency priority based request update policy. In order to satisfy the objective of **CRA**, the priority is maintained according to:

$$\rho_n(t) = E_n^L - \eta_n(s_n(t), s_{-n}(t))$$

It indicates the energy saving obtained from task offloading against local-only computation (i.e., $a_n(t) = 0$). In EFFECT, the request with highest $\rho_n(t)$ is accepted. The algorithm terminates when there are no more update requests or the algorithm has reached the maximum number of iterations t_{max}, whichever comes first.

Algorithm 3: Game-based Task Offloading Algorithm

Collect execution profiles from APs as described in section II-B.
Initialize a global strategy S with all devices selecting the local-only model.
Assign Helper process to each task.
while $t < t_{max}$ ***or not*** *all tasks are completed by the deadline* **do**
  $request$ = ;
  for *each device* n 2 N **do**
    Send $s_n(t$ 1) to Helper(**Algo. 2**) of device n and get $s_n(t)$
    if $s_n(t$ 1) $\neq$ $s_n(t)$ **then**
      $request.add(s_n(t))$
  if $request \neq$; **then**
    Accept $request$: s_n $s_n(t)$, where n = $\underset{n2N}{\text{argmax}}\ \rho_n(t)$
  else
    break
  t $t+1$
return S

V. Performance Evaluation

In this section, we present EFFECT framework prototype and experimental results; followed by simulation results.

A. Testbed Design, Implementation, and Experiment Results

In order to evaluate the performance of the EFFECT resource management on 'near real-world' edge computing platform, we design a small virtual testbed using Amazon EC2 service with 6 device instances (@2.3 GHz 1 vCPU), 2 edge server instances (the performance has been adjusted to @2.3 GHz 8 vCPUs and @2.3 GHz 6 vCPUs), and 1 URB instance (@2.3 GHz 1 vCPU) as shown in Fig 6. We use TCP/IP socket programming to simulate the message passing on the control channels for EFFECT framework described in Fig. 5 (sub-channel bandwidth is simulated as 125 KHz and the maximum transmission power is assumed to be 1 watt).

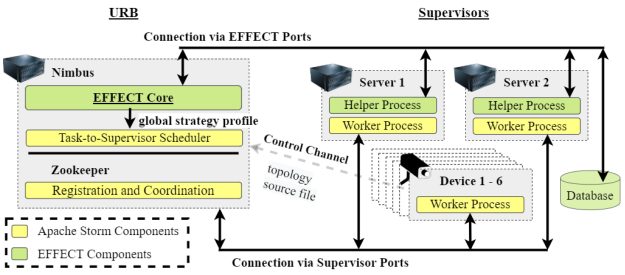

Fig. 6: EFFECT framework prototype implementation and testbed design

1) Testbed design and implementation: In order to implement a multi-device, multi-server task offloading game on a practical development platform, we choose Apache Storm [23], [27] as the distributed stream processing computation platform for the testbed design. In Apache Storm, the communication between jobs are described as data streams. Jobs can either be *spouts*, i.e., a source of streams or *bolts*, i.e., consume and process input streams with the major roles being *Nimbus*, *Zookeeper*, and *Supervisor*. The role-matching (shown in Fig. 6) between Apache Storm and EFFECT are described as follows:

1. The edge servers within the edge units and end-devices are registered as *Supervisor* nodes. However the end-device *Supervisors* only provide service to their own local-processing jobs. These nodes contain a list of "worker" processes, whereas each worker process executes a subset of task jobs.

2. The URB works as the *Nimbus* as well as the *Zookeeper* that distributes the tasks and coordinates the communication between the nodes. The end-devices submit their task topology source files alone with the estimated execution profile list (from subsection V-A1) to the *Nimbus*.

3. A custom scheduler that runs the EFFECT algorithms is deployed to perform server selection and job assignment. CPU allocation is performed using *cpulimit* [28] tool.

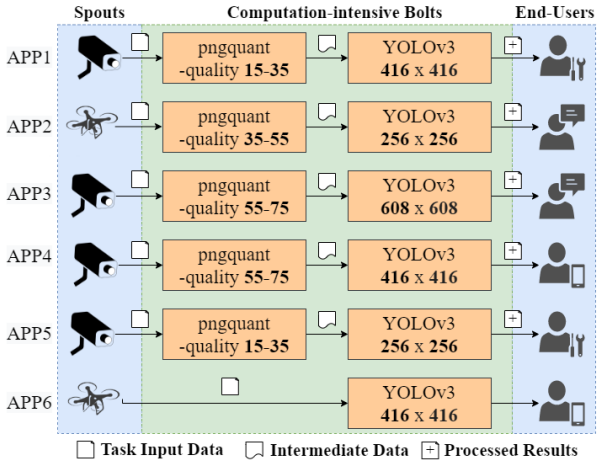

Fig. 7: Applications with different image quality and frame resolutions used for the testbed

In order to mimic computation-intensive and real-time video processing tasks, we implement 6 applications with 2 jobs, as shown in Fig 7. The first job of each application (except APP6) is a lossy PNG compression (often the first stage of video processing) with different image quality options in *pngquant* algorithm [29]. The second job is objection detection through YOLOv3 algorithms [30] using different frame resolutions.

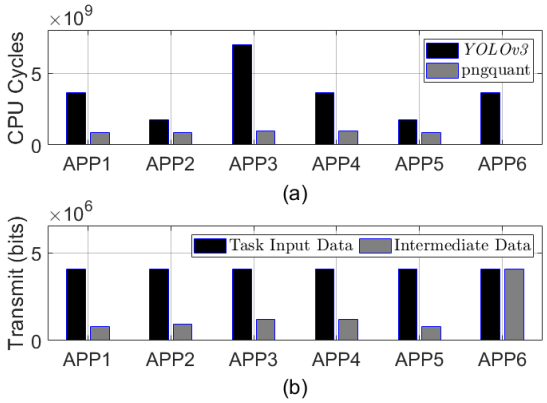

Fig. 8: Computation resource requirement estimation for different applications

For the experiments, we first have to generate the numeric values for execution profiles J_n. Thus, we run *pngquant*

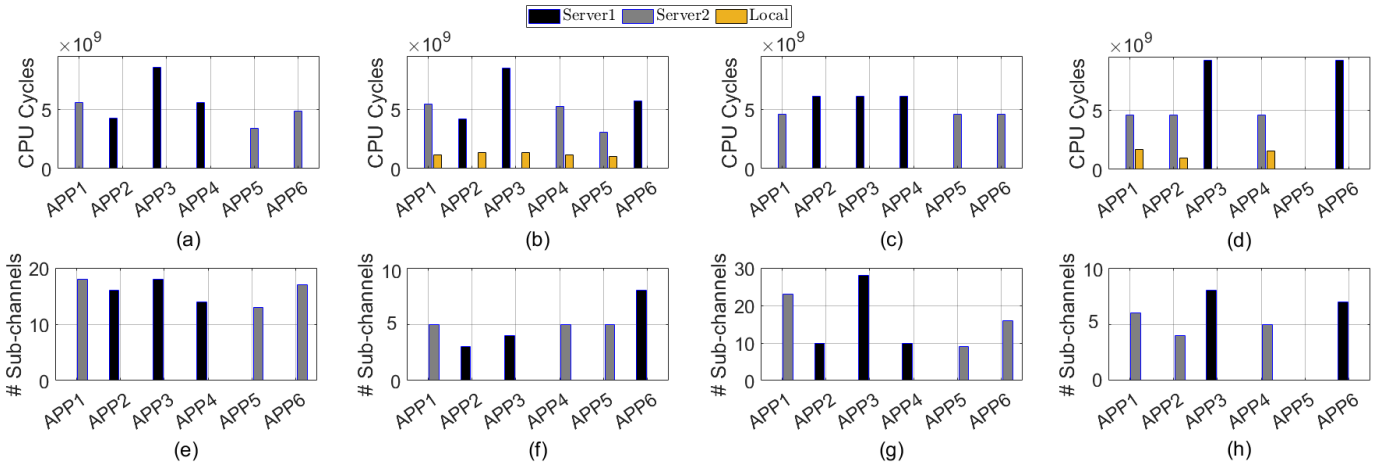

Fig. 9: Computation and network resource allocation for different applications; **Case 1**: (a), (e); **Case 2**: (b), (f), **Case 3**: (c), (g), **Case 4**: (d), (h))

and YOLOv3 with different configurations on server1 independently without running the background processes. With the help of *cpulimit* tool, we monitor the execution latency under different CPU utilization and estimate the computation complexity of each stage by taking the product of delay and the CPU usage. The computation needs of both jobs and data sizes for each application are given in Fig. 8. In Fig. 8 (a) we see that YOLOv3 consumes much more CPU resources than *pngquant* and the CPU demands of YOLOv3 alter greatly when the input frame uses different resolutions. Among all the applications, the APP3 has the heaviest computation tasks with intensive YOLOv3. On the other hand, in Fig. 8 (b) we see that if devices choose to conduct partial task offloading by running *pngquant* locally, the data-transmit requirement can be significantly reduced (70% 80%).

2) Experiment results: We study 4 cases where the availability of network resources is at different levels while maintaining the same amount of computation resources (CPU cycles). **Case 1** and **Case 2** use EFFECT for CPU and sub-channel allocation, while in **Case 3** and **Case 4**, we perform a competing fair CPU allocation, where the CPU resources are evenly allocated to the offloaded task without considering the task deadline.

Case 1: In this case (as shown in Fig. 9 (a) and (e)) we see the application task behavior with EFFECT when network resource availability is sufficient, e.g., 48 sub-channels. From Fig. 9 (a) we see that with EFFECT each tasks are allocated enough sub-channels (i.e., 15+). Thus they are more likely to select full task offloading (i.e., no local computation as shown in Fig. 9 (e)) as energy consumed by data transmission only accounts for a small part of the total energy consumption.

Case 2: As shown in Fig. 9 (b) and (f), in this case we instrument fewer sub-channels for data transmission (only 15 sub-channels in comparison to **Case 1**). Therefore in Fig. 9 (b) and (f), all applications barring APP6 (which only has one job) perform partial task offloading (i.e., some local computation) to reduce the data transmission requirements causing fewer sub-channel allocation.

Case 3: In order to make this case of fair allocation (as shown in Fig. 9 (c) and (g)) comparable to **Case 1**, we ensure sufficient network resources by keeping number of sub-channels to 48 with the sub-channel allocation process still following (11). We see that with fixed CPU allocation, APP1 and APP3 exhibit significant domination in terms of sub-channel occupancy as such applications must use more sub-channels to make up for the lack of computing resources. This in turn has a negative effect on channel efficiency.

Case 4: In this fair allocation scenario with 15 sub-channels (as shown in Fig. 9 (d) - (h)) comparable to **Case 2**, we see that APP1, APP2, and APP4 conduct partial task offloading by running *pngquant* locally, while APP3 and APP6 offload all computations to the edge servers resulting APP5 processing all the jobs locally. This signifies the inefficiency of fair CPU allocation caused by not considering the nature of applications.

The comparison of energy consumption between the four cases are shown in Fig. 10. As excepted, the availability of edge resources (specifically sub-channels in these cases) plays a significant role in task offloading. Between **Case 1** and **Case 2**, the energy consumption can be largely reduced when there are more sub-channels in the system. This is especially true for APP6 which is unable to reduce its data transmission requirement by task partition (from Fig. 10 (a)). In comparison, all applications need to spend more total energy and on data transmission under fair CPU allocation (**Case 1** vs. **Case 3** and **Case 2** vs. **Case 4**) as shown in Fig. 10 (b). Overall, compared to fair CPU allocation, EFFECT saves 40% (in sufficient network resource scenarios) to 60% (in limited network resource scenarios) on the total energy consumption.

B. Simulation Results

We also evaluate EFFECT performance against a larger set of applications and edge servers via simulation. In the simulation, the bandwidth of each sub-channel is set to 2 MHz. The channel gains are modeled by independent Rayleigh fading with average power loss set to 10^{-3} and the white Gaussian noise N_0 is configured at 10^{-9} W [26]. The computation capacity of devices are set between 1.75 GHz and 2.5 GHz that match with general processing speeds of commodity smartphones. The frame sizes and the deadlines are set to 500 - 1000 KB and 0.5 - 1 seconds respectively. Here, we measure the computation capacity of the edge server by counting the number of computational units (e.g., vCPUs). It is assumed

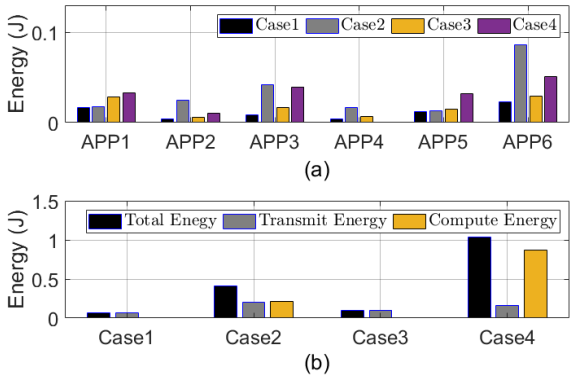

Fig. 10: a) Energy consumption on data transmission for individual applications. b) Energy consumption of all applications combined for different cases.

that the computation speed of each vCPU is tuned at 1.5 GHz. The accuracy value ϵ from **Algo. 1** is set to 1ms which signifies that all tasks should be finished within 1ms before their corresponding deadlines in order to ensure energy saving.

*1) Evaluation of sub-problem **CRA**:* The evaluation uses 10 device tasks with given execution profiles $J_{n,m}$ (pre-confirmed). The summation of computation complexity of processing the remote and local jobs are uniformly distributed between 0.2 10^9 cycles and 0.75 10^9 cycles (for $X_{n,m}$ and $Y_{n,m}$). We compare EFFECT to the following strategies:

1) Local: All tasks are executed on-device, the computation speeds are configured according to **Remark 1**.
2) Joint Resource with Fair Sub-channel Allocation (JR/FS): The resource allocation profile R_n is obtained by **Algo. 1**, however the sub-channels are evenly assigned to each offloaded task.
3) Joint Resource with Proportional Computation Allocation (JR/PC): The resource allocation profile R_n is obtained by **Algo. 1**, however the sub-channels are evenly assigned to each offloaded task and the amount of computation resources obtained by each task is directly proportional to their computation demands.

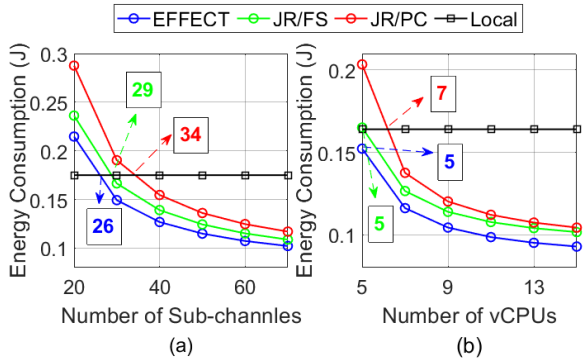

Fig. 11: The energy saving evaluation of **CRA**. a) The computation capacity is selected as 8 vCPUs. b) The number of sub-channels is fixed at 40.

Local vs. Edge: We first compare the energy consumption under different number of sub-channels and vCPUs. Fig. 11 (a) shows the impact of data transmission. With fewer sub-channels in the system, the energy spent on data transmission outweighs the energy preservation benefits of remote computation. To save energy compared to local computing, JR/PC requires at least 34 sub-channels, while JR/FS and EFFECT need 29 and 26 sub-channels, respectively. On the other hand, Fig. 11 (b) shows that the computation capacity of edge servers plays a great role in energy saving. In order to save energy, it shows that JR/FS and EFFECT require at least 5 vCPUs, while JR/PC needs at least 7 vCPUs. In all cases, EFFECT consistently outperforms the other two strategies.

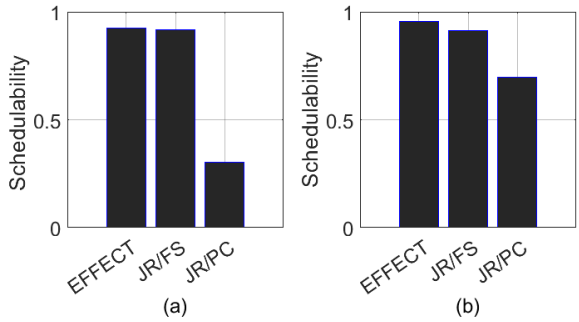

Fig. 12: The schedulability evaluation of sub-problem **CRA**. a) Results with 5 vCPUs and 40 sub-channels. b) Results with 8 vCPUs and 20 sub-channels.

Schedulability: The schedulability of execution profiles in terms of deadline satisfaction against different resource allocation strategies are shown in Fig. 12. The JR/PC strategy suffers from poor schedulability as it ignores the heterogeneity of execution profiles. Whereas, EFFECT jointly optimizes the sub-channel and transmission power allocation as well as the device and edge server computation speed allocation, thus guaranteeing very high schedulability. While JR/FS strategy performs a similar high schedulability, it consumes more energy cost compared to EFFECT as shown in Fig. 11.

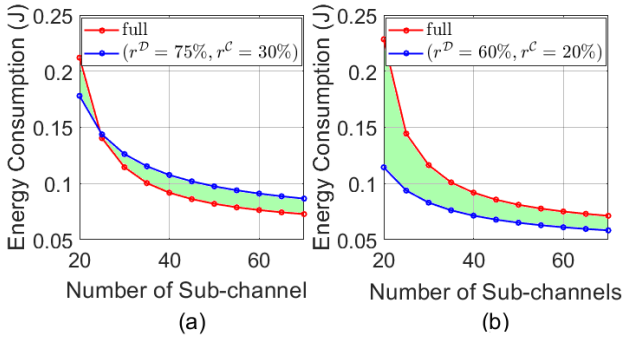

Fig. 13: The energy consumption comparisons between full and partial offloading with amount of saved energy.

Full vs. partial offloading: As it is non-trivial to measure and compare full and partial task offloading when execution profiles have arbitrary computation demands and data sizes, we define a tuple metric (r^D, r^C) to indicate the partial offloading features in order to perform meaningful evaluation. Here, r^D denotes the ratio of intermediate data size to raw data size with r^C implying the ratio of local-processing computation to total computation. We then compare the energy consumption between the two offloading strategies under different number of sub-channels as well as different (r^D, r^C) combinations. For this simulation, the edge server computation is fixed at 5 vCPUs. Fig. 13 (a) shows that in cases with execution profiles $(r^D = 75\%, r^C = 30\%)$ and with more than 25 sub-channels, full offloading provides a better energy saving solution. While

in Fig. 13 (b), partial offloading can save more energy (denoted by colored areas) with (r^D = 60%, r^C = 20%) . Evidently, the lower the ratio of (r^D, r^C), the higher the priority of adopting the partial task offload strategy.

*2) Nash equilibrium of sub-problem **DSM**:* Here, we evaluate the performance of EFFECT **Algo. 3** for sub-problem **DSM** by showing the system convergence. In this simulation, each edge server has 10 to 20 sub-channels and 3 to 5 vCPUs. The task execution profiles are generated by DAGs with 2 to 4 jobs and the job complexity varies between 0.2 10^9 cycles and 0.75 10^9 cycles.

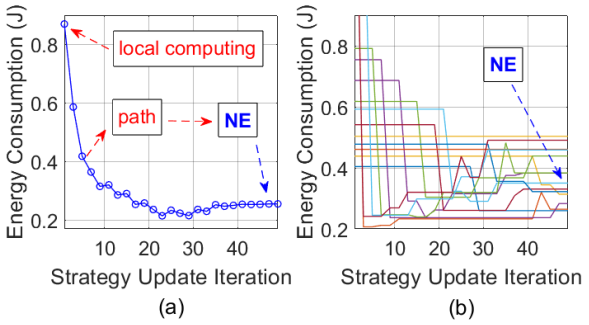

Fig. 14: Task offloading game convergence for 15 devices and 3 edge servers. a) Total energy consumption. b) Individual devices energy consumption.

System convergence: The objective of this evaluation is to show that the task offloading game algorithm terminates after few iterations and no device can further reduce its energy consumption by unilaterally changing its strategy, i.e., all devices reach NE. Fig. 14 (a) shows that all devices select local computation model (i.e., $a_n(0) = 0$) at the very beginning of the game. Based on FIP (in Section III), the improvement path shown in the figure demonstrates the strategy update sequence carried out by the best response strategy that eventually leads to a NE after 40 iterations. Fig. 14 (b) depicts the energy consumption of individual devices which represents how task offloading preferences are changed during the game. Upon further investigation, we observe that the number of iterations is almost linearly correlated to the number of tasks, e.g., a game with N devices requires at most αN iterations to reach NE where α relies on the heterogeneity of resources.

VI. Conclusions

In this paper, we explored how energy consumption can be reduced within fog computing environments without violating service latency constraints by intelligently performing partial task offloading. The theoretical methods and models are designed around proposed EFFECT framework and implemented as algorithms into a virtual testbed. Our testbed and simulation results showed that the EFFECT framework can: i) significantly reduce application task energy consumption, ii) satisfy the execution deadline without violating the selfish nature of individuals, and iii) be computationally lightweight. By jointly optimizing the multi-resource allocation across a multi-sever fog computing environment, the EFFECT framework creates a highly adaptive and intelligent resource provisioning and orchestration environment. The ideas, methods, models, and results presented in this paper can be critical towards a broader paradigm shift that dictates how fog/edge resources could be managed in various transient environments.

References

[1] M. Chen *et al.*, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 587–597, 2018.

[2] K. Zhang *et al.*, "Energy-efficient offloading for mobile edge computing in 5g heterogeneous networks," *IEEE Access*, vol. 4, pp. 5896–5907, 2016.

[3] H. Zhang *et al.*, "Live video analytics at scale with approximation and delay-tolerance," in *USENIX NSDI*, 2017, pp. 377—-392.

[4] C. Hung *et al.*, "Videoedge: Processing camera streams using hierarchical clusters," in *IEEE/ACM SEC*, 2018, pp. 115–131.

[5] J. Wang *et al.*, "Bandwidth-efficient live video analytics for drones via edge computing," in *IEEE/ACM SEC*, 2018, pp. 159–173.

[6] R. Gargees *et al.*, "Incident-supporting visual cloud computing utilizing software-defined networking," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, no. 1, pp. 182–197, 2016.

[7] X. Ran *et al.*, "Deepdecision: A mobile deep learning framework for edge video analytics," in *IEEE INFOCOM*, 2018, pp. 1421–1429.

[8] S. Wang *et al.*, "Surveiledge: Real-time video query based on collaborative cloud-edge deep learning," *arXiv preprint arXiv:2001.01043*, 2020.

[9] C. Wang *et al.*, "Joint configuration adaptation and bandwidth allocation for edge-based real-time video analytics," in *IEEE INFOCOM*, 2020, pp. 1–10.

[10] A. Das *et al.*, "Performance optimization for edge-cloud serverless platforms via dynamic task placement," in *IEEE/ACM CCGRID*, 2020, pp. 41–50.

[11] F. Wang *et al.*, "Joint offloading and computing optimization in wireless powered mobile-edge computing systems," *IEEE Transactions on Wireless Communications*, vol. 17, no. 3, pp. 1784–1797, 2017.

[12] Y. Mao *et al.*, "Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems," *IEEE Transactions on Wireless Communications*, vol. 16, no. 9, pp. 5994–6009, 2017.

[13] J. Ren *et al.*, "Latency optimization for resource allocation in mobile-edge computation offloading," *IEEE Transactions on Wireless Communications*, vol. 17, no. 8, pp. 5506–5519, 2018.

[14] X. Zhang *et al.*, "Migration-driven resilient disaster response edge-cloud deployments," in *IEEE NCA*, 2019, pp. 1–8.

[15] T. Bahreini *et al.*, "Efficient placement of multi-component applications in edge computing systems," in *ACM/IEEE SEC*, 2017, pp. 1–11.

[16] S. Khare *et al.*, "Linearize, predict and place: minimizing the makespan for edge-based stream processing of directed acyclic graphs," in *ACM/IEEE SEC*, 2019, pp. 1–14.

[17] S. Yi *et al.*, "Lavea: Latency-aware video analytics on edge computing platform," in *IEEE ICDCS*, 2017, pp. 2573–2574.

[18] X. Chen, "Decentralized computation offloading game for mobile cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 4, pp. 974–983, 2015.

[19] X. Chen *et al.*, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, 2016.

[20] H. Shah-Mansouri *et al.*, "Hierarchical fog-cloud computing for iot systems: A computation offloading game," *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 3246–3257, 2018.

[21] J. Zhang *et al.*, "Joint computation offloading and resource allocation optimization in heterogeneous networks with mobile edge computing," *IEEE Access*, vol. 6, pp. 19 324–19 337, 2018.

[22] X. Zhang *et al.*, "Energy efficient task offloading for compute-intensive mobile edge applications," in *IEEE ICC*, 2020.

[23] M. Senn, *Apache Storm - https://storm.apache.org/*.

[24] *Face recognition - https://github.com/ageitgey/face_recognition*.

[25] W. Zhang *et al.*, "Energy-optimal mobile cloud computing under stochastic wireless channel," *IEEE Transactions on Wireless Communications*, vol. 12, no. 9, pp. 4569–4581, 2013.

[26] C. You *et al.*, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Transactions on Wireless Communications*, vol. 16, no. 3, pp. 1397–1411, 2017.

[27] W. Zhang *et al.*, "Hetero-edge: Orchestration of real-time vision applications on heterogeneous edge clouds," in *IEEE INFOCOM*, 2019, pp. 1270–1278.

[28] *cpulimit - https://github.com/opsengine/cpulimit*.

[29] *pngquant - https://pngquant.org/*.

[30] J. Redmon *et al.*, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.