Cooperative-Competitive Task Allocation in Edge Computing for Delay-Sensitive Social Sensing

Daniel (Yue) Zhang, Yue Ma, Chao Zheng, Yang Zhang, X. Sharon Hu, Dong Wang
Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN, USA
{*yzhang40, yma1, czheng2, yzhang42, shu, dwang5*}*@nd.edu*

***Abstract*—With the ever-increasing data processing capabilities of edge computing devices and the growing acceptance of running social sensing applications on such cloud-edge systems, effectively allocating processing tasks between the server and the edge devices has emerged as a critical undertaking for maximizing the performance of such systems. Task allocation in such an environment faces several unique challenges: (i) the objectives of applications and edge devices may be inconsistent or even conflicting with each other, and (ii) edge devices may only be partially collaborative in finishing the computation tasks due to the "rational actor" nature and trust constraints of these devices, and (iii) an edge device's availability to participate in computation can change over time and the application is often unaware of such availability dynamics. Many social sensing applications are also delay-sensitive, which further exacerbates the problem. To overcome these challenges, this paper introduces a novel game-theoretic task allocation framework. The framework includes a dynamic feedback incentive mechanism, a decentralized fictitious play with a new negotiation scheme, and a judiciously-designed private payoff function. The proposed framework was implemented on a testbed that consists of heterogeneous edge devices (Jetson TX1, TK1, Raspberry Pi3) and Amazon elastic cloud. Evaluations based on two real-world social sensing applications show that the new framework can well satisfy real-time Quality-of-Service requirements of the applications and provide much higher payoffs to edge devices compared to the state-of-the-arts.**

I. Introduction

This paper targets at task allocation for delay-sensitive social sensing applications in heterogeneous edge computing systems. In recent years, social sensing has emerged as a new networked sensing application paradigm where humans and devices on their behalf are used as "sensors" to collect real-time measurements about the physical world [1], [2]. This paradigm is motivated by the proliferation of portable devices, ubiquitous network connectivity, and the advance in the Internet of Things (IoT). Typical social sensing applications include real-time traffic monitoring using mobile apps [3], obtaining real-time situation awareness during disaster events via online social media [4], and object tracking using portable video devices [5]. Many of these applications are delay-sensitive which require a timely response to the requests from the application users (e.g., querying real-time traffic condition). One key limitation of current social sensing solutions is that data processing and analytics are often done in the "back-end" (e.g., commercial cloud platforms, dedicated servers) [6]. Such design largely ignores the fact that moving data to the "back-end" may induce significant delay to the applications and bandwidth requirement to the communication infrastructure.

Edge computing has become a new computing paradigm that pushes the frontier of computation, data, and services away from centralized servers to the edge [7]. In this paper, we propose a Social Sensing based Edge Computing (SSEC) paradigm where the privately owned sensing devices (e.g., smartphones) are used as the computational resources at the edge of the network to process the data collected from social sensing applications. The advantages of the SSEC paradigm are multi-fold: (i) social sensing applications can process the sensing data right at the edge devices where the data has been collected, which could significantly reduce the communication costs (e.g., bandwidth) and improve the Quality of Service (QoS) (e.g., delay) of the applications; (ii) social sensors (e.g., owner of the edge devices) can obtain payoffs/rewards by leveraging the idle resources of their devices to execute the computing tasks for the application; (iii) the edge computing architecture does not suffer from the single point of failure and alleviates the performance bottleneck of the "back-end" solutions. However, there exist a few important challenges in supporting delay-sensitive social sensing applications in edge computing systems.

In this paper, we focus on the critical problem of allocating delay-sensitive social sensing tasks to heterogeneous edge computing devices. We consider edge devices (often owned by end users) to be rational actors in general (e.g., they are not interested in executing the sensing tasks or sharing their private device status unless incentives/payoffs are provided) [8]. This feature is unique in the edge computing system and in sharp contrast to the often-used "back-end" based solutions where all computational nodes are fully cooperative and the information is shared among all nodes [9], [10]. In particular, there exist a number of challenges in allocating delay sensitive social sensing tasks in edge computing systems.

Competing Objectives: An application and edge-device users may have inconsistent and even conflicting objectives. From the application's perspective, it is important to ensure that the edge devices finish the allocated social sensing tasks in a timely fashion to meet the Quality of Service (QoS) requirements. In contrast, privately owned edge devices are often less concerned about the QoS of applications and more concerned about their own costs (e.g., device's current utilization, energy consumption, memory usage) and payoff.

Incomplete Information: In the edge computing setting, the server often does not have full information about the edge devices. This is because of: i) the privacy concerns of end users; ii) the excessive overhead of synchronizing each devices' status to the server.

Constrained Cooperativeness: previous studies showed that collaboration among computation nodes can significantly improve the efficiency of resource utilization in distributed systems [11], [12]. However, collaboration among edge devices in SSEC is especially challenging because the edge devices are often privately owned by individuals who may not always be willing to share the resource of their devices through collaboration without any incentives [8]. In this work, we assume edge devices are rational actors who are unwilling to collaborate with others unless potential incentives are provided.

Task and Trust Constraints: Various constraints also significantly increase the difficulty of performing task allocation for social sensing applications. First, social sensing applications often consist of complex work-flow of interdependent tasks where it is crucial to explicitly consider the dependencies among tasks [13]. There also exist trust constraints that prohibit the collaboration and communication among arbitrary edge devices (e.g., the users of edge devices often configure the devices to only collaborate with the ones they trust.).

Dynamic Availability: the cost of an edge device and its willingness to execute social sensing tasks can change over time. For example, a user might be less willing to run an edge computing task if she/he uses the phone for video games, which causes a high background utilization on the phone. Failure to capture such dynamics in availability may lead to significantly inferior task allocations where the cost of edge devices to complete a task is too high or against its will.

Existing task allocation schemes in social sensing applications are often "top-down" by assuming that a centralized decision maker (e.g., a project manager or a server-level task allocation coordinator) makes the decision on when and where a task should be executed based on the computation power and real-time status of the devices [13]. Many of these approaches address the challenges of "task dependencies" [14] and "trust constraints" [15]. The underlying assumption of these approaches in incorporating the constraints is that the server has full control over and full information of the computation resources. However, such "top-down" approaches do not fit well with the edge computing paradigm where the server has little or no information about the private status of the edge devices and the edge devices may not always be willing to execute the tasks allocated by the server [16].

An initial effort has been made to address the rational nature of edge devices by developing a Bottom-up Game-theoretic Task Allocation (BGTA) solution [8]. This "bottom-up" approach addresses the challenges of "incomplete information" and "competing objectives". However, BGTA assumes edge devices are all selfish and no collaboration between them is allowed. Therefore, BGTA fails to address the "constrained cooperativeness" challenge and provide suboptimal performance (discussed in Section VI). Moreover, BGTA does not consider the *task and trust constraints* challenge by assuming no task dependency exists and edge devices can communicate arbitrarily without trust constraints. Such assumption is impractical in real-world social sensing scenarios [15]. We also note that simple extensions of BGTA cannot solve the problem discussed in this paper due to the fundamental change of the task models and cooperative-competitive assumption of the edge devices (detailed in Section III).

Furthermore, the proposed framework shares some similarities with existing systems such as HTCondor [17] and Femtocloud [18] that also leverage the spare sources of devices owned by individual users. For example, the HTCondor harnesses the idle computational cycles from distributed workstations to accomplish computation tasks. The more recent FemtoCloud system is a dynamic and self-configuring system architecture that enables privately owned mobile devices to be configured into a coordinated computing cluster. However, both HTCondor and FemtoCloud assume a centralized decision maker in the system to perform the task allocations. In contrast, the task allocations are performed on the distributed edge devices in our system to i) address the unique incomplete information challenge discussed above in SSEC, and ii) allow the edge devices to maximize their own payoffs.

In this paper, we develop a cooperative-competitive (i.e., co-opetitive) game-theoretic task allocation framework, referred to as CoGTA, for delay-sensitive social sensing applications in edge computing systems. CoGTA addresses a few critical challenges that have not been addressed in BGTA. In particular, to address the constrained cooperativeness challenge, we develop a Decentralized Fictitious Play with Negotiation scheme that models the collaboration between devices as "producer-consumer" and allows edge devices to naturally form collaboration while pursuing their own benefits. To address the task and trust constraint challenge, we develop a judicially designed payoff function and negotiation protocol that ensures the various constraints are satisfied in the co-opetitive task allocation process. Finally, to address the dynamic availability challenge, we propose a Dynamic Incentive Adjustment scheme that provides incentives for edge devices to comply with the server's requirement. We implemented a system prototype of CoGTA using RaspberryPi3, Nvidia Jetson TX1, and Jetson TK1 boards as edge devices and Kubernetes and AWS as the elastic cloud servers. The proposed framework was evaluated using two real-world delay-sensitive social sensing applications: *Abnormal Event Detection* [19] and *Real-time Traffic Monitoring* [20]. We compared CoGTA with the state-of-the-art task allocation schemes used in edge computing systems. The results show that our scheme achieves a significant performance gain in terms of meeting the objectives of both applications and edge devices (e.g., our scheme achieved up to 36% decrease in end-to-end delay for the application and 77% more payoffs for edge nodes compared to the baselines).

II. Motivation and Related Work

In this section, we discuss the motivation and related work.

A. Edge Computing for Delay-Sensitive Social Sensing

Social sensing has received a significant amount of attention due to the proliferation of low-cost mobile sensors and the ubiquitous Internet connectivity [1], [2]. A large set of social sensing applications are sensitive to delay, i.e., have real-time requirements. Examples of such applications include intelligent transportation systems [21], video crowdsourcing from mobile devices [22], urban sensing [23], and disaster and emergency response [24]. For instance, one case study in this paper considers *Abnormal Event Detection*, an application whose goal is to identify anomalies from video footages collected by IoT and personal devices and provide alarms for potentially malicious events (e.g., speeding, trespassing unauthorized area) [19]. Clearly, the processing of the video footages and detection of malicious events should be completed under certain real-time constraints. Otherwise, the resultant information would significantly lose its value.

Edge computing systems (encompassing sensors, edge devices and servers in the cloud) are a natural platform for social sensing applications. A comprehensive survey of edge computing is given by Shi, Cao, Zhang, Li, and Xu [25]. For today's social sensing applications, sensors are typically assumed to have limited memory, battery, and computation power. Hence heavy data analytic tasks are transferred, i.e., *offloaded*, to external servers or devices [26]. Pushing all the computation tasks to the remote servers can be rather ineffective particularly for delay sensitive applications due to the limited network bandwidth and high communication latency. Various works have focused on offloading computation tasks to the edge devices, which are often closely connected to sensors, to reduce communication costs and application latency [27]. For example, Satyanarayanan, Bahl, Caceres, and Davies proposed an elegant solution by introducing an intermediate layer (i.e., "Cloudlet") located between the cloud and mobile devices to address the high latency issue between edge devices and servers [7]. Gao proposed a probabilistic computational offloading framework that offloads the computation tasks to the mobile devices [28]. Kosta, Aucinas, Hui, Mortier, and Zhang proposed an energy-aware code offloading framework to dynamically switch between edge devices and cloud servers to improve the energy efficiency of the system [29]. Recently, Saurez, Hong, Lillethun, Ramachandran, and Ottenwälder proposed "Foglet", which is a programming infrastructure for Geo-Distributed situation awareness applications in the fog. Foglet jointly addresses resource discovery, incremental deployment, and live task migration commensurate with application dynamism and resource availability [30]. Our proposed CoGTA framework is complementary to the above related work in that we consider the partial cooperativeness and availability issues in the landscape of edge computing systems.

B. Task Allocation in Real-Time Systems

Computational task offloading for delay-sensitive social sensing applications bears similarity with task allocation in real-time systems. Task allocation is a fundamental problem in real-time systems and both centralized and distributed solutions have been developed to address this problem [9], [10], [31]. For example, Zhu *et al.* proposed a Mixed Integer Linear Programming based approach to meet the deadlines and minimize the end-to-end latency in hard real-time systems [14]. Su and Zhu developed a mixed criticality task allocation model to maximize the number of low-criticality tasks being executed without influencing the timeliness of high-criticality tasks [9]. A set of task allocation schemes have been developed to optimize hardware reliability and energy efficiency in real-time systems [10], [32]. Most of the above schemes adopt a centralized approach that employs a central decision maker to allocate tasks in the system. Such an approach fails in the edge computing systems where the devices might refrain from providing necessary information to accomplish the centralized task allocation [8].

Decentralized task allocation schemes have been developed to address the above limitation. For example, Ahmad, Ranka, and Khan proposed a game theoretic approach for scheduling tasks on multi-core processors to jointly optimize performance and energy [33]. Bertuccelli, Choi, Cho, and How proposed a decentralized task allocation protocol in a dynamic and uncertain environment [31]. However, these decentralized schemes cannot be directly applied to our task allocation problem because they assume cooperative and controlled resources and ignore the "rational actor" nature and the availability issues of end users in edge computing. In contrast, our work proposes a co-opetitive game theoretic approach to overcome the challenges introduced by the competitiveness and availability issues in the edge devices.

C. Game Theory for Resource Allocation

Game theoretic approaches have been widely adopted in resource allocation problems in cloud computing and distributed systems [11], [34], [35]. For example, a set of works have been proposed to adopt game theory to determine the optimal pricing for cloud resources [36], [37]. Pathania, Venkataramani, Shafique, Mitra, and Henkel proposed a decentralized game-theoretic task scheduling scheme for many-core systems to improve scalability and alleviate the computation bottleneck of centralized scheduling algorithms [35]. Zhang *et al.* developed a game-theory based approach to assign computation tasks to *non-cooperative* edge devices with dynamic incentives by considering the conflicting objectives between edge devices and applications [8]. Liu, Khoukhi, and Hafid proposed a decentralized data offloading scheme using the multi-item auction and congestion game approach to allow edge devices to decide the optimal strategy for offloading tasks to the cloud [38]. Our CoGTA framework differs from the above solutions in that i) it explicitly considers the co-opetitive nature of edge devices as well as the task dependencies and trust constraints in the task allocation model; ii) it explicitly considers the dynamic availability of edge devices which has not been fully addressed by the above solutions.

III. Problem Formulation

In this section, we elaborate the co-opetitive task allocation problem for delay-sensitive social sensing applications in edge computing systems. We first present the system models and assumptions for our problem formulation. We then formally define the objectives of our problem.

A. System Models and Assumptions

Figure 1 depicts a high-level view of the CoGTA framework. In this framework, a project manager (resides on the server) launches a delay-sensitive social sensing application that constantly collects sensor data about the physical world via mobile devices (e.g., laptops, drones, smartphones, automobiles). We refer to these devices as *edge devices*. The edge devices may not be constantly connected to the network and often have a limited battery, bandwidth, memory and computation power [25]. Let $EN = \{E_1, E_2 ... E_X\}$ denote the set of all edge devices in the application. These edge devices are not only able to collect sensor data but also perform some computation tasks to reduce the burden of the back-end servers. The *back-end server* is a remote processing unit responsible for task distribution and data processing/analytics. A back-end server is often a powerful and controlled system (e.g., dedicated virtual machines, cloud platforms) that is directly managed by the project manager. We assume the edge devices have constantly changing device status and do not share the dynamic status of the server due to the privacy concerns and synchronization overhead.

We now introduce a few terms used in our model.

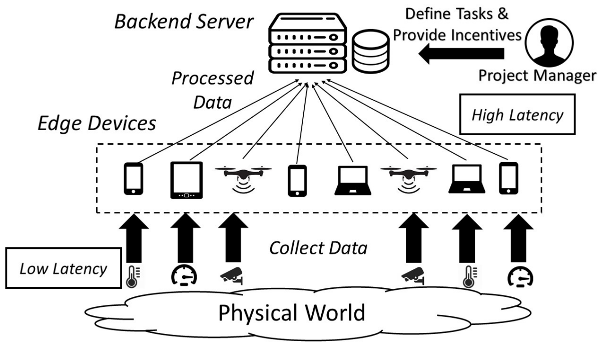


Figure 1: Social Sensing in Edge Computing

DEFINITION 1. Edge Device Status: An edge device status defines the operating state of the edge device at a given time. It consists of an edge device's hardware specification, battery status, location, CPU utilization, and available memory.

We further assume that communication among edge devices is constrained by the trust and privacy concerns of their end users. This assumption is motivated by the observation that edge devices owned by end users without trust relationship may not be able to directly communicate with each other (e.g., due to the privacy settings of the users) [39]. We describe such constraints as a *Trust Graph*:

DEFINITION 2. Trust Graph $\mathbf{G}_{dev}$: an undirected graph $\mathbf{G}_{dev} = (\mathbf{V}_{dev}, \mathbf{L}_{dev})$. If there is a link $(x, y) \in \mathbf{L}_{dev}$ between E_x and E_y, it means that E_x and E_y trust each other. For example, edge devices that belong to users who are friends or from the same institution may trust each other. We assume untrusted edge devices cannot communicate with each other.

In a practical setting, we can also allow the end users to configure their trust constraints. For example, an end user can make his/her device as "public", or by setting it to be visible to a specific set of devices. With the help of device authentication schemes [40], [41], end users can authenticate the devices they trust and work collaboratively in CoGTA.

Next, we discuss the task model in our framework. A social sensing application is assumed to have a set of Z jobs, $Job = \{J_1, J_2, ...J_Z\}$, which are initialized by the server at the beginning of each sensing cycle (i.e., sampling period). Each job converts the raw sensor input data to the final analysis results. We adopt a frame-based task model [42] commonly used in the real-time system community where jobs are periodically initialized and have the same period and deadline. We use Δ to denote the common deadline of all the jobs in an application. Δ captures the user desired QoS in terms of when the jobs should be finished.

To accomplish the data processing function, each job consists of one or more tasks that have dependencies among one another [43]. Each task is associated with a 3-tuple:

$\tau_i = \{VI_i, VO_i, c_{i,x}\}$ where VI_i is the data volume to be processed by task τ_i and VO_i is the size of the output. $c_{i,x}$ is the estimated worst-case execution time (WCET) if τ_i is assigned to edge device $E_x, 1 \leq x \leq X$. We adopt the method in [44] where the execution time of a task can be derived as a function of the CPU cycle of a task, the CPU frequency, and the background utilization of the device. In particular, we have $WCET = CC_i/(Freq_x \times (1 - BU_x)) + C$, where CC_i is the total CPU cycles of a task, $Freq_x$ is the CPU frequency of E_x, BU_x describes the background CPU utilization of E_x that is caused by other applications running on the same system. C is a constant to provide a conservative (worst case) estimation of the execution time. The dependencies among the tasks in an application are modeled by a *task dependency graph* defined below.

DEFINITION 3. Task Dependency Graph ($\mathbf{G}_{task}$): a directed graph $\mathbf{G}_{task} = (\mathbf{V}_{task}, \mathbf{L}_{task})$ where vertex $V_i \in \mathbf{V}_{task}$ represents task τ_i; link $(\tau_i \rightarrow \tau_j) \in \mathbf{L}_{task}$ signifies that the input of task τ_j depends on the output of task τ_i (Figure 2).

For a given application, we assume that a total of N tasks (from all Z jobs) are to be processed in each sensing cycle, i.e., $T = \{\tau_1, \tau_2, ..., \tau_N\}$.

Energy consumption is a major concern for most edge devices. In this paper, we adopt a relatively simple energy model since it is sufficient for demonstrating how CoGTA can take energy into consideration during the task allocation process[1]. Given the WCET of task τ_i, $c_{i,x}$, the energy consumed by executing τ_i on edge device E_x is computed as

$$e_{i,x} = Power_x * c_{i,x}, \tag{1}$$

where $Power_x$ is the average power consumption of edge device E_x and is calculated by

$$Power_x = Power_{comp,x} + Power_{trans,x} \tag{2}$$

where $Power_{comp}$ is the power consumption for computation and P_{trans} is power for data transmission via wireless network and is proportional to the data size been transferred.

Finally, we summarize a few additional assumptions we made in our model:

- We assume edge devices are not malicious (e.g., give fake task outputs) or lazy (i.e., intentionally postpone task executions).
- We assume edge devices do not quit or join the system within a sensing cycle.
- We assume end users are willing to provide part of their computation resources and energy by receiving incentives. Similar to the prior work on incentive mechanisms in crowdsourcing [45]–[47], we assume the incentive mechanism is critical in aligning the competing objectives of end users and the social sensing application in our model.
- We assume the sensor data is collected and processed in a streaming fashion and no latency is expected if both sensing and processing are done on the same edge device. However, the transmission delay is considered if the sensor data is sent to other edge devices for further processing over the network. We assume the proposed task allocation algorithm is running as a client app on each participant's edge device and it decides whether the sensor data should be processed locally or offloaded to other devices.

We discuss how to deal with situations where such assumptions are not satisfied in Section VII.

B. Objectives

Based on the definitions, assumptions and system models, we formally define the objectives of CoGTA. Our goal is to develop a co-opetitive task allocation scheme that can best meet the objectives of both sides. To model the QoS requirement, we define end-to-end delay (E2E delay) of a job as:

DEFINITION 4. End-to-end delay of a job ($\mathcal{L}_z$): the total amount of time taken for a unit of sensor measurement data (e.g., a video frame) to be processed by all tasks in J_z. It includes the total computation time of J_z and the total communication overhead.

Our task allocation problem can be formulated as a multi-objective constrained optimization problem that targets at finding a task allocation scheme to

$$\begin{aligned}
&\text{maximize:} \quad u_x, \forall 1 \leq x \leq X \text{ (edge device's objective)}\\
&\text{minimize:} \quad \sum_{z=1}^{Z} \delta_z, \forall 1 \leq z \leq Z \text{ (application's objective)}\\
&\text{s.t.:} \quad \mathbf{G}_{task}, \mathbf{G}_{dev} \text{ are satisfied}\\
&\qquad\qquad \text{(dependency and trust constraints)}
\end{aligned} \tag{3}$$

Here u_x is the payoff that edge device E_x receives by executing tasks. It defines the individual gain of an edge device and more details on u_x are given in Section IV. δ_z is a binary variable, $\delta_z = 1$ if job J_z misses the deadline and $\delta_z = 0$ otherwise. It is well known that the task allocation problem for heterogeneous distributed systems is in general NP-hard [48], [49]. The problem becomes more challenging in this multi-objective formulation where the objectives of the application and edge devices are potentially conflicting and additional constraints from tasks and devices are imposed. In the next section, we introduce our CoGTA solution to this problem.

[1]The CoGTA framework can be readily extended to more complicated energy models, e.g., supporting multiple voltage/frequency levels. The details are omitted due to page limit.

IV. Co-opetitive Game-theoretic Task Allocation

An overview of the CoGTA framework is given in Figure 2. The framework consists of three major components: i) a novel Cooperative-Competitive Game (CCG) that models the competing objectives between edge devices and applications; ii) a Decentralized Fictitious Play with Negotiation (DFPN) scheme followed by edge devices to make local decisions and autonomously form collaborations to maximize individual payoffs while obeying the task dependency and trust constraints; iii) a Dynamic Incentive Adjustment (DIA) scheme that dynamically tunes the incentives to address the dynamic availability of end users while ensuring QoS of the application. We present these components in detail below.

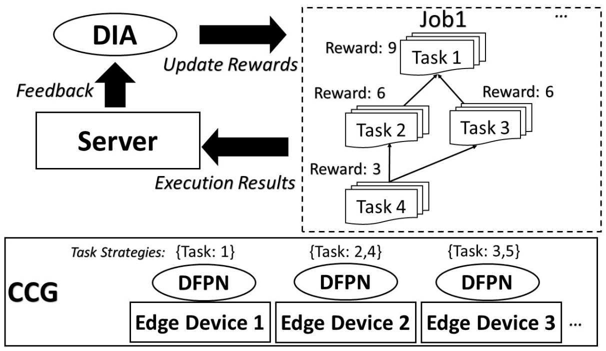


Figure 2: Overview of CoGTA

A. Cooperative-Competitive Task Allocation Game

The Cooperative-Competitive Game (CCG) is designed to specify the roles and protocols of the server and the edge devices in performing the task allocation. The high-level setup of the game adopts the bottom-up task allocation framework proposed in [8]. In particular, the protocol of the game is as follows:

1) At the beginning of a sensing cycle, the server defines Z jobs, the task dependencies and the reward for each task.
2) Each edge device picks a strategy that has the best payoff for itself via a fictitious play with negotiation process (refer to the next subsection for details).
3) If multiple devices choose the same task, for fairness, we randomly assign this task to one of the competing devices (*tie breaking*). We refer to one round of this task allocation process as an "iteration".
4) Within each iteration, we assume each edge device is myopic and only picks one task at a time.
5) Keep iterating until all tasks are picked. Then each device starts to process the tasks it picked.
6) Devices send outputs of executed tasks to the server to claim rewards, the server observes QoS performance loss, and then updates rewards for the next sensing cycle.

Though the game set up in [8] provides an effective approach for bottom-up task allocation, it cannot handle task dependencies nor trust constraints. It also assumes that edge devices are selfish and no collaboration is allowed between them. In contrast, our new CCG framework significantly extends the previous game model by explicitly incorporating the cooperation among devices under task dependency and trust constraints. This is achieved by a complete redesign of payoff functions and the negotiation process. Next, we first introduce a few key terms and notations for the CCG game and then discuss our new payoff functions and negotiation process.

A CCG is described by a tuple $(H, EN, J, T, S, \Psi, \pi, R)$ where H is the game host (i.e., the server) and EN is a set of X players (i.e., edge devices). J is a set of Z jobs and T is a set of N tasks as discussed in Section III. Ψ represents the task allocation strategy space for all players: $\Psi = \Psi_1 \times \Psi_2 \times ... \times \Psi_X$ where Ψ_x represents the strategy space of player E_x. We define a Strategy Profile S as a set of individual task allocation strategies, i.e., $S = \{s_1, s_2, ..., s_X\} \in \Psi$, where s_x denotes the strategy (i.e., which tasks to execute) on edge device E_x. Intuitively, each edge device incurs a cost (such as energy) to execute tasks. Unless there is some incentive, no devices would be willing to execute tasks. This obviously conflicts with the objective of the application. Thus, our CCG model provides rewards to encourage edge devices to execute tasks. Such rewards can be monetary rewards (e.g., cash or virtual currency) or non-monetary (e.g., contribution/experience scores, competitive rankings, virtual badges), which are often adopted in social sensing applications [50].

In the CCG model, the server assigns a reward to each task. Specifically, $R = \{R_1, R_2 ..., R_N\}$ is the set of rewards provided by the application and R_i is the reward for task τ_i. The cost function is associated with the edge devices, where π is a cost function vector $\pi = \{\pi_1, \pi_2, .., \pi_X\}$ and element π_x represents the cost of edge device E_x to execute the allocated tasks based on strategy s_x. To capture the dynamic availability of the edge devices, we model the cost π_x to explicitly consider the costs of edge device E_x as:

$$\pi_x = \begin{cases} e_x = c_{i,x} * (Power_{comp,x} + Power_{trans,x}), \\ \infty, \eta_{b,x} \leq thres_b \end{cases} \tag{4}$$

where e_x is the energy consumed by running task τ_i on E_x,and $\eta_{b,x}$ is the remaining battery of E_x. The intuition here is that an edge device would be less willing to execute a task if i) the energy cost if too high; or ii) the device is running low on battery.

After defining the cost and reward of the CCG, we now introduce the set of payoff functions for a strategy profile S and denote it as $U(S) = \{u_1, u_2, ..., u_X\}$, where u_x is the payoff function of edge device E_x. Specifically, we have:

$$u_x = g(R_x, \pi_x, s_x, s_{-x}) \tag{5}$$

Here s_{-x} represents the task allocation strategies for other edge devices. The payoff function models how much benefit edge device E_x can gain if strategy s_x is taken by the device given other devices' strategies. The goal of each edge device is to maximize its own payoff, namely:

$$\underset{s_x}{\operatorname{argmax}} \ u_x \tag{6}$$

B. *Payoff Function with Supply Chain Model*

A key contribution of CoGTA is to allow cooperation between rational edge devices while considering the task dependencies described by $\mathbf{G}_{task}$ and the trust constraints between devices given by $\mathbf{G}_{dev}$. The main idea of the designed solution is to define three distinct payoff functions: i) a payoff function that captures the benefit of executing the task and sending the execution result to the server; ii) a payoff function that captures the benefit of transferring a task to another edge device; and iii) a payoff function that captures the benefit of accepting a task from another edge device. In contrast to the BGTA framework which only considers the first type of payoff, CoGTA provides edge devices the option to trade tasks that give them better payoffs. This provides the basis for CoGTA to address the constrained cooperativeness challenge in the edge. Below, we first define a key term and then present the payoff functions.

DEFINITION 5. Task Delegation and Collaboration: task delegation refers to the process where an edge device transfers the output of a task to another edge device for further processing. Collaboration between two devices happens when a device (referred to as "producer") is delegating its task to another device (referred to as "consumer").

We solve the co-opetitive task allocation problem with task dependency and trust constraints using a modified "supply chain" model. Specifically, the output of a task is considered as a piece of "raw material" and the subsequent tasks in the job continue to process this "raw material" until the job finishes computation to generate the "final product" (i.e., end results). An edge device can choose to process all materials by itself or, at a certain stage, "sell" (i.e., by performing task delegation) the output of a task to other edge devices or the server to get immediate rewards. To support this execution model, the server provides a reward for each task based on the "difficulty" to process it and how "valuable" the task output is. We discuss reward assignment in details in Subsection IV-D.

To achieve the application objective defined in Equation (3), i.e., minimizing the number of job deadline misses, we define a *penalty function* $l_{i,x,y}$. In particular, after the edge device finishes a task, the server discount the absolute reward with the actual time it takes for the edge device to finish it. The penalty function is derived as:

$$l_{i,x,H} = \begin{cases} \dfrac{\Delta}{\Delta - (td(x,H) + c_{i,x})}, & td(x,H) + c_{i,x} < \Delta \\ \infty, & td(x,H) + c_{i,x} \geq \Delta \end{cases} \tag{7}$$

where $td(x,H)$ denotes the transmission delay from E_x to the server (H) and $c_{i,x}$ is the WCET of τ_i defined in Section III. The intuition of this penalty function is to penalize "lazy but greedy" edge devices who want to get the high rewards for picking many tasks but fail to process them efficiently. Specifically, the closer the total delay (execution time+transmission delay) of a task is to the deadline, the higher the penalty is.

The edge devices can calculate $td(x,H) + c_{i,x}$ by 1) first deriving $c_{i,x}$ by performing schedulability analysis and 2) estimating $td(x,H)$ based on the latency, bandwidth and the data volume to be transmitted to the server. We assume that edge devices acquire the location of the server (to estimate latency) and the bandwidth when it first joins the application.

Given the above definitions, we formally derive the payoff functions, a key element in the CCG. The payoff functions are associated with each edge device's strategy and contain several different forms to capture the various execution scenarios. Assume that device E_x picks task τ_i in an iteration and $d(i)$ is the number of devices that pick τ_i. Based on the reward, penalty and cost, we define the *payoff function* $u_{i,x}^H$ of edge device E_x for finishing task τ_i and offloading the rest of the job to the server as:

$$u_{i,x}^H = \begin{cases} Exp(\dfrac{R_i}{\pi_x * l_{i,x,H}}) = \dfrac{R_i * (\Delta - (td(x,H) + c_{i,x}))}{\Delta * d(i) * e_x} \\ 0, \quad l_{i,x,H} = \infty \ or \ \pi_x = \infty \end{cases} \tag{8}$$

where $Exp(\cdot)$ finds the expected payoff, which is simply the original payoff divided by the number of devices that pick the tasks due to the tie breaking process. In the above equation, the payoff of executing a task depends on the expected reward (absolute reward discounted by the delay penalty) ($Exp(\frac{R_i}{l_{i,x,H}})$) per unit cost of energy. Intuitively, the higher energy cost or the lower expected reward of a task, the less likely an edge device will pick that task.

We further derive the payoff of performing collaboration between devices. For a task delegation, we assume E_x is the producer of task τ_i's output and $E_{x'}$ is the consumer of this output by picking a subsequent task $\tau_{i'}$. The *collaboration payoff function for producer* E_x with respect to τ_i is

$$u_{i,x}^p = \begin{cases} \dfrac{R_i * (\Delta - (0.5 * td(x,x') + c_{i,x}))}{\Delta * d(i) * \pi_x}, & x' \neq x \ or \ H \\ 0, \quad (x' \to x) \notin \mathbf{L}_{dev} \ or \ x' = x. \end{cases} \tag{9}$$

We allow the producer and the consumer to share the penalty due to the transmission delay between them (i.e., $0.5 * td(x, x')$). The above payoff function definition takes into consideration the trust constraints by setting the payoff to 0 if E_x and $E_{x'}$ are not connected in $\mathbf{G}_{dev}$.

Now assume an edge device E_x is receiving output from a set of tasks $\{\tau_{i'} | (i' \to i) \in \mathbf{L}_{task}\}$. Let $x(i')$ denote the device that picks a task $\tau_{i'}$. We design the *collaboration payoff function for consumer* E_x with respect to τ_i below.

$$u^c_{i,x} = \begin{cases} \dfrac{u^H_{i,x}}{d(i)} - \displaystyle\sum_{i'}^{(i' \to i) \in \mathbf{L}_{task}} \dfrac{u^p_{i',x(i')}}{d(i)}, \\ 0, \quad \exists i', x(i') = null \ or \ (x(i'), x) \notin \mathbf{L}_{dev} \end{cases} \tag{10}$$

To enforce task dependency constraint, we use the term $\exists i', x(i') = null \ or \ (x(i'), x) \notin \mathbf{L}_{dev}$ (i.e., the second case in Equation (10) to represent the case where some of the previous tasks of τ_i have not been picked or picked by untrusted devices. Hence, picking τ_i will yield no actual payoff for E_x. This design is to ensure that edge devices only consider tasks whose previous tasks have been picked.

With the CCG set up given here, we are now ready to introduce the scheme for "playing" the game in the following two subsections.

C. Decentralized Fictitious Play with Negotiation (DFPN)

To ensure that each edge device makes its best decision towards its objective, our goal is to find a Nash Equilibrium for the CCG. The Nash Equilibrium exists in a game where each player is assumed to know the equilibrium strategies of all other players and no player has anything to gain by only changing his/her own strategy [51]. A unique challenge in finding the Nash Equilibrium in our CCG model is that an edge device often has incomplete or no information on the individual status of other devices. This results in a scenario where an edge device cannot precisely derive the best strategies of other devices to make its own best response.

Here, we propose an efficient learning scheme to find the optimal task allocation based on Fictitious Play (FP) and negotiation heuristics to significantly reduce both the search space and convergence time. The key idea of FP is that each player is intended to "guess" the strategies other players might pick based on their historical strategies (without knowing their actual payoff functions) and make its own decision to yield the highest payoff for itself. The algorithm is described as:

1) *Initial Step:* Each edge device initializes a local empirical histogram ($histo$) [52] as a basis to "guess" the strategies other players might employ based on their historical strategies, where $histo_{i,x}$ denotes the frequency of E_x picking τ_i. We initialize $histo$ at the beginning of the game as: $histo_{i,x} = \frac{R_i}{\sum_{i=1}^{N} R_i}$. This follows the intuition that each edge device, without knowing other devices' payoff functions, guesses that every competitor tends to pick the tasks with higher rewards.
2) *Best Response:* Each edge device then picks the task with the highest payoff (defined in Equation ((8)) based on the guess of all other players' strategies.
3) *Negotiation:* After picking a task, each edge device decides whether to send the output of the task to the server or "sell" it to other devices. An edge device labels the task as "on sale" if it is willing to perform task delegation. Each edge device also decides whether to "buy" a task on sale. Each device performs this negotiation with the goal of maximizing $u^p_{i,x}$ and $u^c_{i,x}$ (defined in Equation (9)–(10)), respectively.
4) *Coordination:* After all the devices make decisions, a local edge server collects all the decisions and shares them with all the devices.
5) *Update Belief:* Each device observes the strategies of others and then updates its empirical histogram by increasing the corresponding strategy count by 1. Then the estimation on the future strategies is derived as the most frequent strategy of each device, recorded by $histo$.

The above steps are repeated until convergence. During each iteration, edge devices negotiate for tasks and the server breaks ties if multiple edge devices pick the same task. CoGTA adopts a similar weighted singleton congestion game protocol as BGTA [8] which allows the scheme to converge and find the Nash Equilibrium quickly. We evaluate the coordination overhead and convergence of DFPN in Section VI. The scheme differs from the fictitious play algorithm in BGTA by designing a new negotiation process that allows the edge devices to be both cooperative and competitive. In particular, we assume a device is always willing to invite collaboration if (i) it cannot consume the output of a task; and (ii) the increased reward of the collaboration is non-negative. Specifically, we derive the reward increase as $u^p_{i,x} - u^H_{i,x}$. If another device is willing to accept the task delegation (via $u^c_{i,x}$), the collaboration is formed.

D. Dynamic Incentive Adjustment

Our CCG and DFPN approaches discussed in the previous subsections ensure that the edge devices achieve maximum payoff through a co-opetitive game play, given the reward assignment of each task. As shown in the payoff functions (Equation (8)–(9)), the definition of the rewards plays a critical role with respect to whether a task is picked by edge devices for execution. Due to the dynamic availability issues, static reward functions are generally not optimal for the server to satisfy QoS requirements. A dynamic reward assignment scheme was proposed in [8], in which the reward for a job is dynamically assigned based on a feedback control mechanism. However, it cannot be directly applied to our model due to the introduction of task dependency. In this work, we introduce a new Dynamic Incentive Adjustment scheme for the server to dynamically update task rewards in order to meet the QoS objective of the application.

The design of the task reward function is based on the following two considerations. First, if a task must transfer a large amount of input data to the server for processing, it would be "more valuable" to execute the task on an edge device. Second, if a task is at later stages of a job (i.e., close to the final output), it would be more "valuable" in general so as not to waste the processing that has already been done for the job. Now, let VO_i and VI_i be the size of output and input of τ_i, respectively, and $drr_i = \frac{VO_i}{VI_i}$ be the data reduction ratio (DRR). Thus DRR represents how much reduction of data communication is achieved by accomplishing a task. Furthermore, let aggregated computation complexity (ACC) acc_i be recursively defined as $acc_i = \sum_{(i'->i)\in \mathbf{L}_{task}} acc_{i'} + c_{i,H}$. The ACC represents the total computational effort to finish a task. Finally, we define task reward R_i for τ_i as a linear combination of drr_i and acc_i:

$$R_i = \frac{\alpha_1}{\alpha_1 + \alpha_2} drr_i + \frac{\alpha_2}{\alpha_1 + \alpha_2} acc_i \tag{11}$$

where α_1 and α_2 are the weighting factors. To ensure that it is meaningful to add DRR and ACC, we normalize both DRR and ACC scores (Section VI). Therefore, the reward of each individual task is bounded.

The dynamic tuning of rewards is accomplished by adopting the exponential weights algorithm [53] as a feedback control mechanism. The intuition of this tuning process is that the server assumes there exist two "experts" who vote for the importance of the two factors ("DRR" and "ACC") in the reward function. Each expert's vote is associated with the corresponding weight (α_1 or α_2). Based on the experts' votes, the server calculates the reward according to Equation (11). After the jobs are executed, the server can observe which jobs missed the deadlines and re-attribute the performance feedback to each expert as a loss function and adjust their weights accordingly. The detailed tuning process is presented in the Appendix.

V. System Design and Implementation

This section presents the system design, hardware setup used in our experiments, and our implementation of the CoGTA framework.

A. System Architecture

An overview of the system architecture is illustrated in Figure 3. The system is composed of an elastic cloud server to host the social sensing application and provide additional computation resources. An edge server is used to provide near-edge coordination among local edge devices. The edge devices are composed of heterogeneous embedded boards.

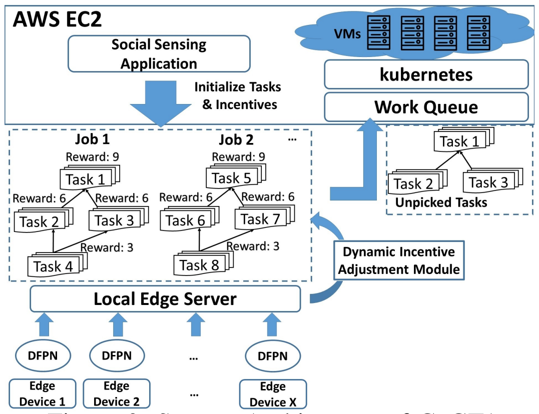


Figure 3: System Architecture of CoGTA

At the beginning of each sensing cycle, the application defines the number of data sources, the definition of each task, and the reward for each task. The edge devices then run DFPN to pick their most beneficial tasks and execute them. The edge server takes care of the synchronization of DFPN so any edge device is aware of the decisions of other devices. The tasks that are not picked by edge devices are then sent back to the server for further processing. We describe these system components in details below.

B. Server Design

Remote Cloud Server: We set up a Kubernetes [54] cluster as the back-end servers on Amazon AWS by using EC2. It contains four c4.xlarge EC2 instances with each instance having four cores and 7.5 GB RAM. The horizontal Pod autoscaler of Kubernetes provides elastic resource management to handle dynamic requests from edge devices in delay-sensitive social sensing applications.

Local Edge Server: In the edge computing paradigm, edge devices are commonly connected to the cloud through local edge servers. Such a design can i) provide a generic communication interface between heterogeneous edge devices and the cloud service and ii) buffer simultaneous requests from edge devices. In our implementation, we use a PC workstation with Intel E5-2600 V4 processor and 16GB of DDR4 memory as the local edge server. To enable the task submission process, we applied the Work Queue framework [55] which sets up a master-worker architecture between the local edge server and AWS. The edge server also coordinates edge devices to synchronize their strategies after each iteration of the DFPN algorithm. Note that the local edge server does not take on computation tasks which contrasts many existing architecture [7], [28], [56]. The reasons are two-fold: 1) edge devices are increasingly powerful and capable of performing complex computation tasks themselves [8]; 2) it is more economically viable to minimize the deployment of expensive local edge servers mainly for computational purpose [57]. The economic consideration is particularly important in ensuring the scalability of the system for large real-world social sensing applications.

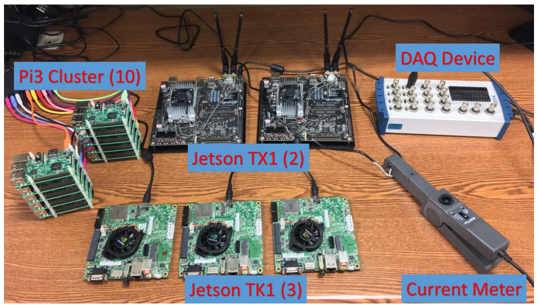


Figure 4: Heterogeneous Edge Computing Platform

C. Edge Design

Figure 4 shows the hardware platform for the edge devices. The platform consists of 15 edge devices: 2 Jetson TX1 [58] and 3 Jetson TK1 [59] boards from Nvidia, and 10 Raspberry Pi3 Model B boards [60]. These edge devices have heterogeneous energy profiles and computation capabilities and are commonly used in portable computers, UAVs, and autonomous vehicles.

D. Network Configuration

All devices and the edge server are connected via a local wireless router to emulate the proximity of the edge. The average latency between the edge server and edge devices is 12.6 ms, which is insignificant compared to the execution time of the CoGTA algorithm. The data communication is achieved using TCP sockets.

E. Energy Measurement

Energy data is needed in our system. We use FLUKE AC/DC current clamps to monitor real-time current signal of each edge device and capture the current values using a National Instruments USB-6216 Data Acquisition (DAQ) system. We then multiply the current values with the default voltage (12 V for TK1, 19 V for TX1 and 5 V for Pi3) to obtain the power consumption by each edge device.

VI. Evaluation

We now present an extensive evaluation of our CoGTA scheme using the hardware setup described above. We first discuss the baselines for comparison. We then present the evaluation results using two real-world social sensing applications: *Abnormal Event Detection* and *Real-time Traffic Monitoring*.

A. Baselines

We choose the following representative task allocation schemes from recent literature as baselines.

- **Congestion (COG)**: A congestion game based edge computing task allocation scheme where tasks are modeled as resources and the reward of a task is monotonically decreasing as more edge devices claiming that task [38].
- **Bottom-up Game-Theoretic Task Allocation (BGTA)**: A recent bottom-up task allocation scheme that does not allow cooperation among edge devices [8].
- **Greedy-Max Reward (GMXR)**: A greedy task allocation scheme where an edge device greedily picks the tasks with the highest reward [61].
- **Centralized Server-based Allocation (CSA)**: A centralized task allocation scheme where an edge device sends all its computation tasks to the cloud servers.
- **Mixed Integer Linear Programming (MILP)**: A top-down task allocation scheme with cooperative distributed computing resources using MILP to minimize the deadline miss rate [62].

Note that MILP is a top-down task allocation scheme that assumes the server has full control and information of the edge device. This is not practical in the edge computing paradigm and unfair to CoGTA and other baselines that assume rational edge devices and incomplete information of the edge devices at the server. However, we treat the performance of MILP as a performance upper-bound and investigate how close the performance of CoGTA and other schemes is compared to the upper bound.

B. Case Study 1: Abnormal Event Detection

The first case study is *Abnormal Event Detection (AED)* in social sensing where the goal is to generate alerts of abnormal events from video data contributed by camera-enabled sensors (mobile phones, drones, etc.). For example, in a mobile social sensing project, the participants are tasked to take videos/images of a location assigned by the application to help identify abnormal events such as trespassing, sudden movements, and appearance of unusual objects. Upon detection of the abnormal events, the application provides alerts to its subscribed users or the general public.

We use the UCSD Anomaly Detection Dataset [63] which consists of 98 (50 training, 48 testing) video footages collected from surveillance cameras that monitor pedestrian walkways around the UCSD campus. The dataset provides ground truth labels for two abnormal events: i) detection of non-pedestrian objects; ii) anomalous pedestrian motion patterns. We treat this dataset as the video data collected from the edge devices since our edge devices do not have cameras. We assume that each edge device generates one sensor data stream. Each run of the experiment contains a total of 100 sensing cycles. Within each sensing cycle, the edge devices are tasked to process a total of 8 seconds of video clips (with each video source sampled at 20 image frames per cycle).

Jobs for Abnormal Activity Detection: We adopt the abnormal activity detection framework proposed in [19] to define the set of tasks for a job in this application: i) data

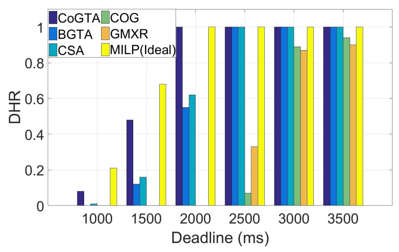


Figure 5: DHR in AED

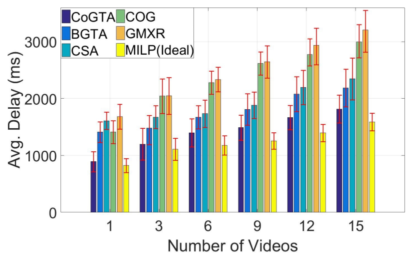


Figure 6: E2E Delay in AED (Δ = 3s)

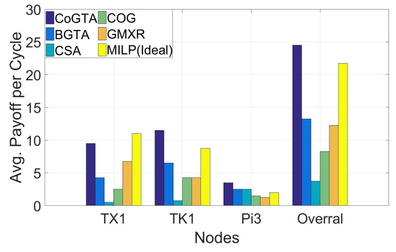


Figure 7: Avg. Payoff in AED

collection of video signals as image frames; ii) image pre-possessing (i.e., converting images to gray-scale); iii) motion feature extraction (i.e., optical flow); iv) object detection using YOLO framework [5]; v) feature classification using a sparse combination classifier [19]. All jobs have the same set of tasks but process different video clips. The results from different jobs are sent back to the server for the aggregation and final decision.

To impose trust constraints, we design two trust groups and each device is randomly assigned to one of the two groups in an experiment. We assume devices within each group can communicate with each other but inter-group communication is not allowed due to trust constraints. The evaluation results are averaged over 100 experiments.

1) Quality of Service (Application (Server) Side): In the first set of experiments, we focus on how the objective is achieved from the application side. In particular, we evaluate the deadline hit rate (DHR) and end-to-end (E2E) delay of all the compared task allocation schemes. The DHR is defined as the ratio of tasks that are completed within the deadline. The results are shown in Figure 5. Here we use all 15 edge devices and gradually increase the deadline constraints. We observe that CoGTA has significantly higher DHRs than all the baselines, except MILP (i.e., the upper bound of performance), and is the first one that reaches 100% DHR as the deadline increases. We attribute such performance gain to our deadline-driven dynamic incentive adjustment that guides the edge devices' decisions towards minimizing deadline miss rate by providing dynamic incentives. We also observe that CoGTA significantly outperforms the BGTA scheme because i) CoGTA develops a novel supply chain based negotiation scheme that allows task trading between edge devices while BGTA which assumes no cooperation between edge devices; ii) the penalty term in the payoff function of CoGTA discourages an edge device from picking the tasks that it cannot process efficiently, which was not considered in BGTA. We also observe the performance of CoGTA is very close to the performance upper bound (i.e., the MILP scheme that assumes full control and full information).

Figure 6 summarizes the E2E delays of all the schemes as the number of video sources varies. We show the average delays and the 90% confidence bounds. Clearly, our CoGTA scheme has the least E2E delay and tightest confidence bounds compared to the baselines while approaching the performance upper bound (i.e., MILP). The results further demonstrate the effectiveness of CoGTA for meeting real-time QoS requirements of the application. The performance gain of the CoGTA is achieved by explicitly modeling the dynamic status of the edge devices (e.g., computation capability and energy profile) and allocating tasks according to the current device status.

2) Payoff and Energy Consumption (Edge Device Side): In the second set of experiments, we focus on how the objectives of edge devices are achieved. In particular, we study the payoff and energy consumption of edge devices. Figure 7 shows the results of the payoff gained by the edge devices. We normalize DRR and ACC to a 0-100 scale and cost π_x to a 1-5 scale. We observe that CoGTA has the highest payoff compared to all the baselines. This is because CoGTA can most efficiently finish the jobs by pushing the computation to the edge, thus avoiding the communication latency of sending tasks to the server. This results in the least penalty for the rewards gained. Also, CoGTA allows edge devices to maximize their rewards via achieving the Nash Equilibrium. The negotiation process of DFPN allows CoGTA to significantly outperform other game-theoretic schemes (i.e., BGTA and COG) by allowing edge devices to collectively work out their best payoffs through task trading and cooperation. We note that CoGTA also outperforms MILP in terms of the payoff even though MILP represents the upper-bound in terms of QoS objectives (DHR and E2E delay). The reason is that MILP only focuses on the overall system performance without considering the benefits of edge devices. This is one the key advantages of CoGTA compared to the "top-down" approaches like MILP.

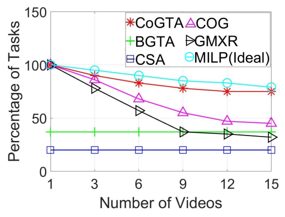
The results of energy consumption by edge devices are shown in Table I. Here we use all 15 edge devices and set the number of video sources to 10 and deadline to 3 seconds. The number of video sources being less than the number of edge devices emulates the scenarios that the number of devices can be much larger than the number of jobs required by the application in many real-world applications [64]. We can observe that the CSA and BGTA baselines consume the least energy due to the fact that they push most of the computation tasks to the back-end servers. However,

Table I: Average Energy (mJ) per Frame in AED

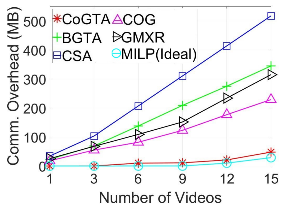
	CoGTA	BGTA	CSA	COG	GMXR	MILP
TX1	167.71	160.84	127.99	163.88	161.39	174.88
TK1	162.73	161.45	122.17	162.03	158.20	162.63
Pi3	64.89	65.72	64.03	67.10	69.64	63.05
All	1472.51	1463.23	1262.79	1484.85	1493.78	1468.15

Table II: Average Energy (mJ) per Frame in RTM

	CoGTA	BGTA	CSA	COG	GMXR	MILP
TX1	152.87	148.70	125.76	148.61	146.57	161.75
TK1	148.86	143.77	120.98	152.67	151.22	146.88
Pi3	62.52	62.88	60.48	67.52	68.78	61.21
All	1377.52	1353.91	1219.61	1430.43	1434.60	1376.24



(a) Percentage of Tasks Finished by Edge Devices per Cycle



(b) Edge to Server Communication Overhead per Cycle

Figure 8: Communication Costs in AED

CoGTA achieves much higher DHR and lower E2E delay since CoGTA adopts an energy-aware payoff function and thus uses energy more efficiently.

3) Communication Overhead (Edge to Server): We further investigate the communication overhead from edge devices to servers. Figure 8(a) shows the percentage of tasks (within a sensing cycle) that are accomplished by the edge devices. We observe that CoGTA and MILP finish the largest percentage of the tasks at the edge devices among all the schemes, which greatly alleviates the computation burden of the server. CoGTA offloads slightly more tasks to the server as compared to MILP. The results on the communication overhead (in terms of the amount of data sent from edge devices to the server) are shown in Figure 8(b). We observe that CoGTA incurs the least communication overhead compared to the baselines. The reduced communication overhead also explains the E2E delay reduction achieved by CoGTA shown in Figure 6.

C. Case Study 2: Real-time Traffic Monitoring

The second case study is *Real-time Traffic Monitoring (RTM)* where participants in a social sensing application use personal mobile devices (e.g., mobile phones, dash cams) to record and analyze the current traffic conditions. For example, a traffic monitoring application can task a set of drivers to use their dash cams to take videos of the traffic in front of their vehicles and then infer the congestion rate of the road.

We collected the video data using dash cams from two vehicles. The data contains a total of 30 video clips and 15 of them are used for training. We divided the application into 100 sensing cycles and each sensing cycle processes video clips of 6 seconds (with each video sampled at 15 frames per cycle).

Jobs for Traffic Monitoring: i) data collection of traffic video signal as image frames; ii) image prepossessing; iii) feature extraction (optical flow and HOG); iv) feature classification using trained SVM to identify vehicles and traffic counts. The final result of each job is further processed by the server to infer the overall traffic condition.

1) Quality of Service (Application (Server) Side): We perform similar experiments as those discussed in the previous subsection. In particular, we evaluate all the schemes in terms of DHR and E2E delay. The results are shown in Figure 9 and Figure 10 respectively. We observe similar results of CoGTA as the previous case study. We also observed that both DHR and E2E delay results are better than those in the previous section. The reason is that i) the YOLO framework used in the previous application is more computationally intensive; ii) fewer image frames need to be processed per sensing cycle.

2) Energy Consumption and Payoff (Edge Device Side): The results of payoffs gained at the edge devices are shown in Figure 11. We observe that our scheme continues to provide significantly more amount of payoffs to the edge devices than other baselines. CoGTA also outperforms MILP in terms of the payoff in this case study. This again demonstrates that CoGTA is more "user-friendly" by allowing edge devices to obtain more payoffs. The results of the energy consumption of the edge devices are reported in Table II. Similar performance gains of CoGTA are observed.

3) Communication Overhead (Edge to Server): The results of the percentage of tasks processed by the edge devices and the communication overhead from the edge to the server are reported in Figures 12(a) and 12(b) respectively. Similar to the previous application, we observe a significant reduction in the percentage of tasks and the amount of data sent to the server.

D. CoGTA Convergence and Scalability

Finally, we study the convergence and coordination overhead of the CoGTA scheme. The results are reported in Figure 13. In particular, Figure 14(a) shows the average number of iterations of CoGTA till convergence. We observe that CoGTA finds the Nash Equilibrium fairly quickly. Furthermore, Figure 14(b) shows the execution time of CoGTA. The execution time includes the running time of the DFPN as well as the communication delay between the edge servers and the edge devices. We observe the execution time of CoGTA grows almost linearly as the number of edge devices increases. The above results again demonstrate the suitability of using CoGTA for delay-sensitive social sensing applications.

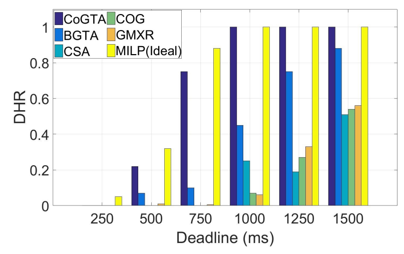


Figure 9: DHR in RTM

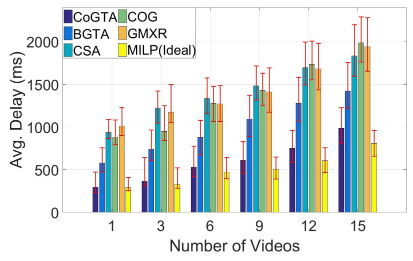


Figure 10: E2E Delay in RTM (Δ = 2s)

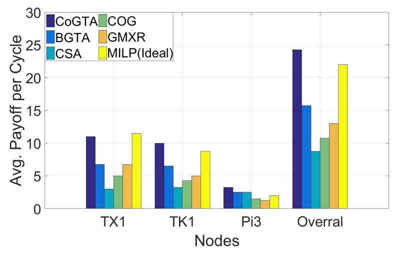
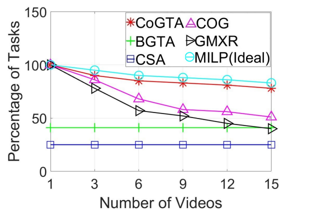


Figure 11: Avg. Payoff in RTM


(a) Percentage of Tasks Finished by Edge Devices per Cycle

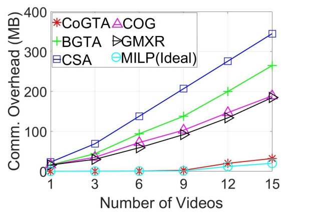
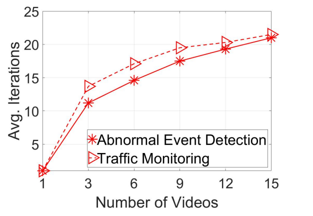

(b) Edge to Server Communication Overhead per Cycle

Figure 12: Communication Costs in RTM


(a) Number of Iterations of CoGTA till Convergence

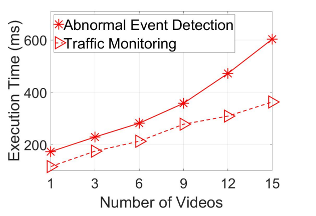
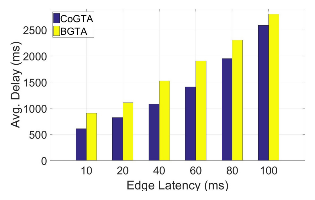

(b) Execution Time of CoGTA till Convergence

Figure 13: Convergence of CoGTA

We note that the execution time of the CoGTA scheme might become a non-trivial overhead when the number of edge devices in the system becomes very large. A possible solution to such scalability problem is to increase the number of local edge servers and run CoGTA in the cluster of edge devices coordinated by the same local edge server. In our implementation, we assume the edge server only performs coordination between the cloud and the edge and synchronizes the task allocation decisions among edge devices. Therefore, the local edge server can be inexpensive smart gateways or mobile cloudlets rather than expensive dedicated servers.

E. Evaluating Co-opetitive Negotiation

The above experiments showed that CoGTA outperforms BGTA in terms of both E2E delay and payoffs. A key factor that contributes to the performance gain is the negotiation process (i.e., DFPN algorithm). In this experiment, we further evaluate how this negotiation process could influence the performance gain of CoGTA compared to BGTA. We


(a) E2E Delay of CoGTA vs. BGTA

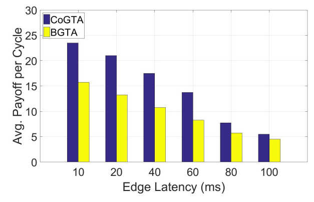

(b) Payoffs of COGTA vs. BGTA

Figure 14: Comparing CoGTA with BGTA

use the RTM application as an example and use the same settings as the previous experiments. The results are shown in Figure 14. The x-axis is the emulated latency between edge devices and the y-axis are the performance metrics of interest - E2E delay and average payoffs. We can observe that, as the edge latency increases, CoGTA's performance degrades on both metrics. This is because the negotiation process becomes less effective when trading tasks between edge devices become more expensive (i.e., high latency between edge devices). On the other hand, we also observe that the CoGTA still significantly outperforms BGTA when the negotiation process is carried out effectively (i.e., when the latency between edge devices is low). The above result shows the significant impact of the new DFPN algorithm.

VII. Conclusion and Future Work

This paper presents the CoGTA framework to addresses three fundamental challenges in solving the task allocation problem for delay-sensitive social sensing applications in edge computing systems, namely *competing objectives*, *constrained cooperativeness* and *dynamic availability*. We have implemented our proposed framework on a hardware setup including Nvidia Jetson TK1, TX1, Raspberry Pi3 boards, and AWS EC2. The evaluation results from two real-world social sensing applications demonstrate that CoGTA achieves significant performance gains in terms of meeting both the objectives of applications and edge devices.

Our work has some limitations and can be extended further. In this work, we propose a unique vision of using private edge devices owned by individuals to perform edge computation tasks, which is in sharp contrast to the existing model that assume edge devices are collaborative [7], [28].

Though our approach brings the benefit of exploring the massive amount of privately owned computational resources on the edge, it introduces many challenges to be solved as well. One particular challenge is user privacy. For example, the data collected from edge devices can potentially reveal the private information of end users. Also, it is possible for the application to monitor the task execution time of the edge devices to perform device profiling analysis [65]. A potential solution to this problem is to build a secure privacy-preserving crowdsourcing architecture such as AnonySense [66] that allows the edge devices to contribute anonymized data to an untrusted server.

Second, CoGTA entails security concerns. In particular, we assume that edge devices are not malicious and they follow the game protocol honestly (e.g., will not quit abruptly, or intentionally delay the executing). However, there may be malicious edge devices that intentionally generate wrong results for tasks or delay the execution of tasks to generate sub-optimal results. The first issue can be addressed by combining CoGTA with the verifiable computing techniques where the system maintains verifiable results by requiring a "proof" of the correct execution of tasks [67]. The second issue can be mitigated by adding an extra function in CoGTA to keep track of the normal behaviors of edge devices and actively block the identified lazy devices. These directions are left for future work.

Third, CoGTA does not consider the dynamic churn rate of end users. We assume the availability of devices does not change within each sensing cycle. In practice, such availability could change at any point in time, which may lead to unfinished or partially finished tasks. This issue has initially been discussed in [18], in which task re-assignment and adaptive workload management mechanisms have been developed to mask device churn. We can extend CoGTA to address this problem by allocating important tasks to more than one edge devices so the task will be finished unless all devices associated with the task decide to quit simultaneously [68]. Alternatively, CoGTA can also specify that the edge devices could only join and quit at the beginning of each sensing cycle by imposing a high penalty cost for the devices who quit without finishing their tasks. We leave these extensions for future work.

Fourth, CoGTA is a real-time task allocation scheme that maximizes deadline hit rate of the system instead of providing the hard real-time guarantee. This is due to several factors. First, the worst-case estimation of the task execution time is not precise due to the complicated computing and communication environment in edge computing systems [44]. In the future, we would explore more sophisticated execution time prediction schemes (e.g., static program analysis [69] and narrow spectrum benchmarking [70] in CoGTA. Second, the convergence time of the Nash Equilibrium solution is also highly dynamic and unpredictable. This can be addressed by setting the maximum number of iterations in DFPN or relaxing strict Nash Equilibrium by performing ϵ-Nash Equilibrium Solution [71].

Finally, the current CoGTA scheme has not fully explored the mobility and battery status of the edge devices due to the constraints of the current experiment platform. In real-world scenarios, the location of the edge devices and battery status may influence the dynamic cost and availability of the end users [8], [72]. To address this limitation, we plan to i) extend the current payoff function in CoGTA to explicitly consider the physical distance and the battery status of the device (e.g., higher payoffs will be offered to devices with lower battery and further distance from the task); ii) extend the experiment platform by adding edge devices with the mobility feature (e.g., UAVs and wearable devices).

Acknowledgement

This research is supported in part by the National Science Foundation under Grant No. CNS-1831669, CBET-1637251, CNS-1566465, IIS-1447795 and CNS-1319904, Army Research Office under Grant W911NF-17-1-0409, Google 2017 Faculty Research Award. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

We also thank our shepherd Professor Umakishore Ramachandran for providing valuable input to our paper.

VIII. Appendix

We update the weights (α_1 and α_2) in Equation 11 as:

$$\alpha_k^{new} = \alpha_k^{old} * e^{-\eta\lambda_k}, \quad 1 \leq k \leq 2.$$

η is a learning parameter, λ_k is a loss function for α_k, and

$$\lambda_1 = \frac{\sum_{z=1}^{Z} o_z^t * \delta_z}{\sum_{z=1}^{Z} \delta_z} - \frac{\sum_{z=1}^{Z} o_z^t * (1-\delta_z)}{\sum_{z=1}^{Z}(1-\delta_z)}, \sum_{z=1}^{Z} \delta_z \neq 0 \ or \ Z$$

$$\lambda_2 = \frac{\sum_{z=1}^{Z} o_z^c * \delta_z}{\sum_{z=1}^{Z} \delta_z} - \frac{\sum_{z=1}^{Z} o_z^c * (1-\delta_z)}{\sum_{z=1}^{Z}(1-\delta_z)}, \sum_{z=1}^{Z} \delta_z \neq 0 \ or \ Z,$$

where δ_z signifies whether the job J_z misses the deadline or not (see Section III). In the above equation, we use the average transmission overhead o_z^t (edge to server) of the tasks that meet the deadlines (i.e., $\frac{\sum_{z=1}^{Z} o_z^t*(1-\delta_z)}{\sum_{z=1}^{Z}(1-\delta_z)}$) as a set point and compare it with the average transmission overhead of the tasks that missed the deadlines ($\frac{\sum_{z=1}^{Z} o_z^t*\delta_z}{\sum_{z=1}^{Z} \delta_z}$). The same idea applies for the ACC factor (using computation overhead o_z^c). If all jobs meet their deadlines (i.e., $\sum_{z=1}^{Z} \delta_z = 0$), we keep the current reward function. In the rare case where all jobs miss their deadlines (i.e., $\sum_{z=1}^{Z} \delta_z = Z$), we resort to admission control and decrease the number of assigned jobs. Due to the page limit, we omit the detailed discussions.

References

[1] D. Wang, B. K. Szymanski, T. Abdelzaher, H. Ji, and L. Kaplan, "The age of social sensing," *IEEE Computer*, 2018.

[2] D. Wang, M. T. Amin, S. Li, T. Abdelzaher, L. Kaplan, S. Gu, C. Pan, H. Liu, C. C. Aggarwal, R. Ganti *et al.*, "Using humans as sensors: an estimation-theoretic perspective," in *Information Processing in Sensor Networks, IPSN-14 Proceedings of the 13th International Symposium on*. IEEE, 2014, pp. 35–46.

[3] P. Giridhar, M. T. Amin, T. Abdelzaher, D. Wang, L. Kaplan, J. George, and R. Ganti, "Clarisense+: An enhanced traffic anomaly explanation service using social network feeds," *Pervasive and Mobile Computing*, vol. 33, pp. 140–155, 2016.

[4] D. Wang, L. Kaplan, H. Le, and T. Abdelzaher, "On truth discovery in social sensing: A maximum likelihood estimation approach," in *Proc. ACM/IEEE 11th Int Information Processing in Sensor Networks (IPSN) Conf*, Apr. 2012, pp. 233–244.

[5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 779–788.

[6] D. Zhang, D. Wang, N. Vance, Y. Zhang, and S. Mike, "On scalable and robust truth discovery in big data social media sensing applications," *IEEE Transactions on Big Data*, 2018.

[7] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE pervasive Computing*, vol. 8, no. 4, 2009.

[8] D. Zhang, Y. Ma, Y. Zhang, S. Lin, X. S. Hu, and D. Wang, "A real-time and non-cooperative task allocation framework for social sensing applications in edge computing systems," in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2018, pp. 316–326.

[9] H. Su and D. Zhu, "An elastic mixed-criticality task model and its scheduling algorithm," in *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, 2013, pp. 147–152.

[10] J.-J. Chen and C.-F. Kuo, "Energy-efficient scheduling for real-time systems on dynamic voltage scaling (dvs) platforms," in *Embedded and Real-Time Computing Systems and Applications, 2007. RTCSA 2007. 13th IEEE International Conference on*. IEEE, 2007, pp. 28–38.

[11] G. Wei, A. V. Vasilakos, Y. Zheng, and N. Xiong, "A game-theoretic method of fair resource allocation for cloud computing services," *The journal of supercomputing*, vol. 54, no. 2, pp. 252–269, 2010.

[12] L. Chen, S. Zhou, and J. Xu, "Computation peer offloading for energy-constrained mobile edge computing in small-cell networks," *arXiv preprint arXiv:1703.06058*, 2017.

[13] D. Y. Zhang, C. Zheng, D. Wang, D. Thain, X. Mu, G. Madey, and C. Huang, "Towards scalable and dynamic social sensing using a distributed computing framework," in *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*. IEEE, 2017, pp. 966–976.

[14] Q. Zhu, H. Zeng, W. Zheng, M. D. Natale, and A. Sangiovanni-Vincentelli, "Optimization of task allocation and priority assignment in hard real-time distributed systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 11, no. 4, p. 85, 2012.

[15] S. Ponda, J. Redding, H.-L. Choi, J. P. How, M. Vavrina, and J. Vian, "Decentralized planning for complex missions with dynamic communication constraints," in *American Control Conference (ACC), 2010*. IEEE, 2010, pp. 3998–4003.

[16] M.-A. Messous, H. Sedjelmaci, N. Houari, and S.-M. Senouci, "Computation offloading game for an uav network in mobile edge computing," in *Communications (ICC), 2017 IEEE International Conference on*. IEEE, 2017, pp. 1–6.

[17] M. J. Litzkow, "Remote unix: Turning idle workstations into cycle servers," in *Proceedings of the Summer USENIX Conference*, 1987, pp. 381–384.

[18] K. Habak, M. Ammar, K. A. Harras, and E. Zegura, "Femto clouds: Leveraging mobile devices to provide cloud service at the edge," in *2015 IEEE 8th International Conference on Cloud Computing (CLOUD)*. IEEE, 2015, pp. 9–16.

[19] H. Tan, Y. Zhai, Y. Liu, and M. Zhang, "Fast anomaly detection in traffic surveillance video based on robust sparse optical flow," in *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1976–1980.

[20] G. Chatzimilioudis, A. Konstantinidis, C. Laoudias, and D. Zeinalipour-Yazti, "Crowdsourcing with smartphones," *IEEE Internet Computing*, vol. 16, no. 5, pp. 36–44, 2012.

[21] Z. Xu, H. Gupta, and U. Ramachandran, "Sttr: A system for tracking all vehicles all the time at the edge of the network," in *Proceedings of the 12th ACM International Conference on Distributed and Event-based Systems*. ACM, 2018, pp. 124–135.

[22] D. Y. Zhang, Q. Li, H. Tong, J. Badilla, Y. Zhang, and D. Wang, "Crowdsourcing-based copyright infringement detection in live video streams," in *Proceedings of the 2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2018*, 2018.

[23] Y. Zhang, N. Vance, D. Zhang, and D. Wang, "Optimizing online task allocation for multi-attribute social sensing," in *The 27th International Conference on Computer Communications and Networks (ICCCN 2018)*. IEEE, 2018.

[24] C. Huang and D. Wang, "Topic-aware social sensing with arbitrary source dependency graphs," in *Proceedings of the 15th International Conference on Information Processing in Sensor Networks*. IEEE Press, 2016, p. 7.

[25] 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.

[26] A. Mtibaa, K. A. Harras, and A. Fahim, "Towards computational offloading in mobile device clouds," in *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, vol. 1. IEEE, 2013, pp. 331–338.

[27] K. Kumar and Y.-H. Lu, “Cloud computing for mobile users: Can offloading computation save energy?” *Computer*, vol. 43, no. 4, pp. 51–56, 2010.

[28] W. Gao, “Opportunistic peer-to-peer mobile cloud computing at the tactical edge,” in *Military Communications Conference (MILCOM), 2014 IEEE*. IEEE, 2014, pp. 1614–1620.

[29] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, “Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading,” in *Infocom, 2012 Proceedings IEEE*. IEEE, 2012, pp. 945–953.

[30] E. Saurez, K. Hong, D. Lillethun, U. Ramachandran, and B. Ottenwälder, “Incremental deployment and migration of geo-distributed situation awareness applications in the fog,” in *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*. ACM, 2016, pp. 258–269.

[31] L. F. Bertuccelli, H.-L. Choi, P. Cho, and J. P. How, “Real-time multi-uav task assignment in dynamic and uncertain environments,” *American Institute of Aeronautics and Astronautics*, 2009.

[32] Y. Ma, T. Chantem, R. P. Dick, and X. S. Hu, “Improving system-level lifetime reliability of multicore soft real-time systems,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 6, pp. 1895–1905, 2017.

[33] I. Ahmad, S. Ranka, and S. U. Khan, “Using game theory for scheduling tasks on multi-core processors for simultaneous optimization of performance and energy,” in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*. IEEE, 2008, pp. 1–6.

[34] 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.

[35] A. Pathania, V. Venkataramani, M. Shafique, T. Mitra, and J. Henkel, “Distributed scheduling for many-cores using cooperative game theory,” in *Design Automation Conference (DAC), 2016 53nd ACM/EDAC/IEEE*. IEEE, 2016, pp. 1–6.

[36] F. Teng and F. Magoules, “Resource pricing and equilibrium allocation policy in cloud computing,” in *Computer and information technology (CIT), 2010 IEEE 10th international conference on*. IEEE, 2010, pp. 195–202.

[37] F. Teng and F. Magoulès, “A new game theoretical resource allocation algorithm for cloud computing,” in *International Conference on Grid and Pervasive Computing*. Springer, 2010, pp. 321–330.

[38] D. Liu, L. Khoukhi, and A. Hafid, “Decentralized data offloading for mobile cloud computing based on game theory,” in *Fog and Mobile Edge Computing (FMEC), 2017 Second International Conference on*. IEEE, 2017, pp. 20–24.

[39] J. L. Williams, J. W. Fisher, and A. S. Willsky, “Approximate dynamic programming for communication-constrained sensor network management,” *IEEE Transactions on signal Processing*, vol. 55, no. 8, pp. 4300–4311, 2007.

[40] M.-S. Hwang and L.-H. Li, “A new remote user authentication scheme using smart cards,” *IEEE Transactions on consumer Electronics*, vol. 46, no. 1, pp. 28–30, 2000.

[41] G. E. Suh and S. Devadas, “Physical unclonable functions for device authentication and secret key generation,” in *Design Automation Conference, 2007. DAC'07. 44th ACM/IEEE*. IEEE, 2007, pp. 9–14.

[42] A. Allavena and D. Mossé, “Scheduling of frame-based embedded systems with rechargeable batteries,” in *Workshop on Power Management for Real-time and Embedded systems (in conjunction with RTAS 2001)*, 2001.

[43] J. Dean and S. Ghemawat, “Mapreduce: simplified data processing on large clusters,” *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[44] C. You, K. Huang, H. Chae, and B.-H. Kim, “Energy-efficient resource allocation for mobile-edge computation offloading,” *IEEE Transactions on Wireless Communications*, vol. 16, no. 3, pp. 1397–1411, 2017.

[45] D. Peng, F. Wu, and G. Chen, “Pay as how well you do: A quality based incentive mechanism for crowdsensing,” in *Proceedings of the 16th ACM International Symposium on Mobile Ad Hoc Networking and Computing*. ACM, 2015, pp. 177–186.

[46] D. Yang, G. Xue, X. Fang, and J. Tang, “Crowdsourcing to smartphones: incentive mechanism design for mobile phone sensing,” in *Proceedings of the 18th annual international conference on Mobile computing and networking*. ACM, 2012, pp. 173–184.

[47] X. Zhang, Z. Yang, Z. Zhou, H. Cai, L. Chen, and X. Li, “Free market of crowdsourcing: Incentive mechanism design for mobile sensing,” *IEEE transactions on parallel and distributed systems*, vol. 25, no. 12, pp. 3190–3200, 2014.

[48] J. Chen and L. K. John, “Efficient program scheduling for heterogeneous multi-core processors,” in *Proceedings of the 46th Annual Design Automation Conference*. ACM, 2009, pp. 927–930.

[49] K. W. Tindell, A. Burns, and A. J. Wellings, “Allocating hard real-time tasks: an np-hard problem made easy,” *Real-Time Systems*, vol. 4, no. 2, pp. 145–165, 1992.

[50] L. G. Jaimes, I. Vergara-Laurens, and M. A. Labrador, “A location-based incentive mechanism for participatory sensing systems with budget constraints,” in *Pervasive Computing and Communications (PerCom), 2012 IEEE International Conference on*. IEEE, 2012, pp. 103–108.

[51] X. Vives, “Nash equilibrium with strategic complementarities,” *Journal of Mathematical Economics*, vol. 19, no. 3, pp. 305–321, 1990.

[52] E. Campos-Nañez, A. Garcia, and C. Li, “A game-theoretic approach to efficient power management in sensor networks,” *Operations Research*, vol. 56, no. 3, pp. 552–561, 2008.

[53] S. Sorin, “Exponential weight algorithm in continuous time,” *Mathematical Programming*, vol. 116, no. 1-2, pp. 513–528, 2009.

[54] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014.

[55] C. Zheng and D. Thain, "Integrating containers into workflows: A case study using makeflow, work queue, and docker," in *Proceedings of the 8th International Workshop on Virtualization Technologies in Distributed Computing*. ACM, 2015, pp. 31–38.

[56] M. Satyanarayanan, P. Simoens, Y. Xiao, P. Pillai, Z. Chen, K. Ha, W. Hu, and B. Amos, "Edge analytics in the internet of things," *IEEE Pervasive Computing*, vol. 14, no. 2, pp. 24–31, 2015.

[57] T. H. Luan, L. Gao, Z. Li, Y. Xiang, G. Wei, and L. Sun, "Fog computing: Focusing on mobile users at the edge," *arXiv preprint arXiv:1502.01815*, 2015.

[58] Nvidia, "Jetson Tegra X1." [Online]. Available: http://www.nvidia.com/object/embedded-systems-dev-kits-modules.html

[59] ——, "Jetson Tegra K1." [Online]. Available: http://www.nvidia.com/object/jetson-tk1-embedded-dev-kit.html

[60] Raspberry, "Raspberry Pi 3b." [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[61] D. Wicke, D. Freelan, and S. Luke, "Bounty hunters and multiagent task allocation," in *Proceedings of the 2015 international conference on autonomous agents and multiagent systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2015, pp. 387–394.

[62] A. Davare, J. Chong, Q. Zhu, D. M. Densmore, and A. L. Sangiovanni-Vincentelli, "Classification, customization, and characterization: Using milp for task allocation and scheduling," *Systems Research*, 2006.

[63] V. Mahadevan, W. Li, V. Bhalodia, and N. Vasconcelos, "Anomaly detection in crowded scenes," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE, 2010, pp. 1975–1981.

[64] A. Ghasemi and E. S. Sousa, "Opportunistic spectrum access in fading channels through collaborative sensing." *JCM*, vol. 2, no. 2, pp. 71–82, 2007.

[65] T. K. Buennemeyer, T. M. Nelson, L. M. Clagett, J. P. Dunning, R. C. Marchany, and J. G. Tront, "Mobile device profiling and intrusion detection using smart batteries," in *hicss*. IEEE, 2008, p. 296.

[66] C. Cornelius, A. Kapadia, D. Kotz, D. Peebles, M. Shin, and N. Triandopoulos, "Anonysense: privacy-aware people-centric sensing," in *Proceedings of the 6th international conference on Mobile systems, applications, and services*. ACM, 2008, pp. 211–224.

[67] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers," *Advances in Cryptology–CRYPTO 2010*, pp. 465–482, 2010.

[68] A. Kulkarni, M. Can, and B. Hartmann, "Collaboratively crowdsourcing workflows with turkomatic," in *Proceedings of the acm 2012 conference on computer supported cooperative work*. ACM, 2012, pp. 1003–1012.

[69] R. Heckmann and C. Ferdinand, "Worst-case execution time prediction by static program analysis," in *In 18th International Parallel and Distributed Processing Symposium (IPDPS 2004, pages 26–30. IEEE Computer Society*, 2004.

[70] R. H. Saavedra-Barrera, "Cpu performance evaluation and execution time prediction using narrow spectrum benchmarking," Ph.D. dissertation, University of California, Berkeley, 1992.

[71] R. Radner, "Collusive behavior in noncooperative epsilon-equilibria of oligopolies with long but finite lives," in *Noncooperative Approaches to the Theory of Perfect Competition*. Elsevier, 1982, pp. 17–35.

[72] H. To, L. Fan, L. Tran, and C. Shahabi, "Real-time task assignment in hyperlocal spatial crowdsourcing under budget constraints," in *Pervasive Computing and Communications (PerCom), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1–8.