Data-Driven Edge Computing Resource Scheduling for Protest Crowds Incident Management

Jon Patman*, Peter Lovett†, Andrew Banning‡, Annie Barnett§, Dmitrii Chemodanov*, Prasad Calyam*
*Department of Electrical Engineering & Computer Science, University of Missouri, USA
†Computer & Information Science Department, University of Oregon, USA
‡Department of Computer Science, Southeast Missouri State University, USA
§Department of Computer Science, Cornell University, USA
{dycbt4, jpxrc}@mail.missouri.edu, plovett@uoregon.edu, arbanning1s@semo.edu,
akb222@cornell.edu, calyamp@missouri.edu

***Abstract*—Computation offloading has been shown to be a viable solution for addressing the challenges of processing compute-intensive workloads between low-power devices and nearby servers known as cloudlets. However, factors such as dynamic network conditions, concurrent user access, and limited resource availability often result in offloading decisions negatively impacting end users in terms of delay and energy consumption. To address these shortcomings, we investigate the benefits of using Machine Learning for predicting offloading costs for a facial recognition service in a series of realistic wireless experiments. We also perform a set of trace-driven simulations to emulate a multi-edge protest crowd incident case study and formulate an optimization model that minimizes the time taken for all service tasks to be completed. Because optimizing offloading schedules for such a system is a well-known NP-complete problem, we use mixed integer programming and show that our scheduling solution scales efficiently for a moderate number of user devices (10-100) with a correspondingly small number of cloudlets (1-10), a scale commonly sufficient for public safety officials in crowd incident management. Moreover, our results indicate that using Machine Learning for predicting offloading costs leads to near-optimal scheduling in 70% of the cases we investigated and offers a 40% gain in performance over baseline estimation techniques.**

I. Introduction

The recent advances in cloud computing technologies and smart mobile devices has given rise to new systems that bring cloud-like applications and services to users on mobile devices. The maturing Internet of Things (IoT) paradigm has also provided an avenue for offering new services to users via low-power, and often wireless, embedded devices. The applications driven by IoT-based computing have emerged in a variety of domains such as smart city infrastructure, industrial remote sensing, disaster response, and protest crowd management. One of the biggest challenges hindering the full realization of IoT-based applications is how best to orchestrate user devices and resources that have limited processing capabilities while operating in potentially unreliable networks [1].

An emerging paradigm known as edge computing seeks to augment low-power devices with access to more responsive cloud-like services by migrating computational workloads from devices to nearby servers known as cloudlets [2]. For mobile devices in particular, Mobile Edge Computing (MEC) can be used to prolong the device's battery life and can perform computationally intensive tasks with less delay for the user [3]. 'Computation offloading' is a popular strategy for mitigating the issues associated with resource management in MEC networks. Offloading takes place by partitioning the device's application or workload (e.g. requests or data streams) in order to distribute the partitions to nearby servers with the aim of reducing latency or energy consumption for the device or the overall system.

Recent approaches involving energy-aware offloading policies offer flexibility to users who require energy conservation over low-latency or vice versa in visual IoT-based data processing [3]. These approaches are limited in that offloading policy selection is based on heuristics and domain-knowledge experience, which may not fully capture the heterogeneity of edge resources. The challenges of managing edge networks are exacerbated by the fact that a lot of these services are deployed on wireless networks, which are more susceptible to noise and disruptions than traditional cloud computing systems. Moreover, available edge resources are finite and need to be efficiently managed in order to provide users with the most cost-efficient policies for their device. To produce more accurate and flexible estimates about networking and computing behavior, machine learning models can be trained to predict both the transmission and processing delays for data being offloaded between a mobile device and a cloudlet.

Facial recognition and tracking applications which rely on deep learning are rapidly growing in demand but have traditionally been too computationally expensive to deploy on low-power consumer devices [4]. Facial recognition services, therefore, are an ideal use-case for studying the complexities of edge computing systems. Commercial services such as Amazon's Rekognition API and AWS Lambda functions are streamlined in order to offer advanced visual processing capabilities but also assume that users accessing those services have reliable Internet connectivity. In urban areas experiencing crowd protests or emergency situations, authorities may need to quickly assess the location of "bad actors" and act accordingly. Transferring imagery collected by mobile devices to nearby cloudlets allows for processing that provides incident-related situational awareness for emergency personnel.

978-1-5386-7659-2/18/$31.00 ©2018 IEEE

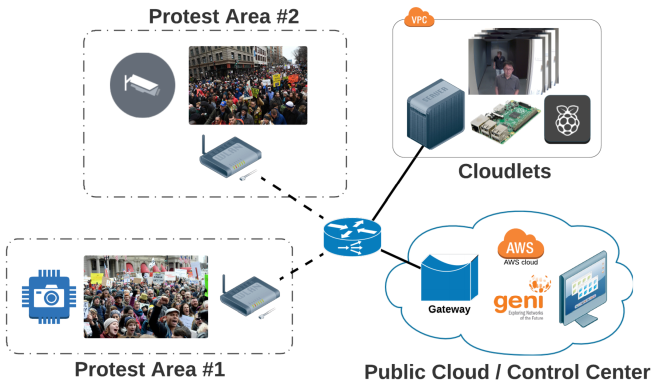


Fig. 1. Protest crowd management use case involving IoT and other remote devices that can be augmented via cloudlets; aim of the visual data processing services is to assist public safety personnel with gaining timely situational awareness in potentially hostile environments, where network and computing resources are limited.

Figure 1 shows an example of an edge computing use-case for protest crowds incident management where data distributed data in the form of images or video streams is transmitted wirelessly to nearby cloudlets for processing and storage. The challenges in such use cases typically include: noisy and congested networks, high device-to-server ratios, and heterogeneous computing resources, all of which pose unique challenges for accurately estimating offloading costs. Furthermore, static network profiling techniques may fail to capture the complexities of such systems, which in turn makes: (a) efficient scheduling of tasks problematic, and (b) offloading performance to degrade over time.

In this paper, we propose a multi-edge resource scheduling scheme that provides the following contributions that can be organized under two major research thrusts:

Predicting offloading cost in wireless edge networks. We conduct several real-world wireless experiments (see Subsection IV-A) for a range of computing devices and networking conditions. In particular, we are interested in measuring the corresponding transmission and processing delays for each experiment. Observations of the collected data indicate that there is wide variability in estimating transmission delay using standard network measurement tools. Furthermore, we discovered that the task of estimating processing speed is non-trivial as it often relies on installing application profiling software which may not be compatible with virtual or heterogeneous resources available at the edge.

To address these shortcomings, we leverage the statistical capabilities of data-driven techniques, namely Machine Learning (ML) algorithms to estimate the offloading costs for different workloads under various experimental conditions. Using historical data from our wireless experiments, we train several state-of-the-art ML models on a total of 4000 data points for predicting transmission and processing delays. Based on our evaluation results (see Section IV-B), we evaluate the extent to which the ML-based models outperform traditional estimation techniques in terms of both accuracy and precision.

Optimal scheduling of offloading tasks for data processing. We formulate the challenge of offloading tasks from various mobile devices to nearby cloudlets as the popular job shop scheduling problem where we seek to minimize the 'maximum schedule time' when all the tasks have finished processing (also known as the *makespan*) across all edge servers. Such an objective minimization allows us to better balance the available physical resources. This in turn increases the acceptance ratio for future offloading requests [5] (i.e., the online optimization), which further improves the overall application throughput.

To evaluate our approach, we conduct a series of trace-driven simulations consisting of multiple devices offloading computational tasks (e.g. recognizing faces in images) to nearby cloudlets in a multi-edge environment. We vary the number of user devices from 10-100 while also scaling the number of available cloudlets from 1-10 in order to investigate the scalability of our approach for a commonly experienced scale of protest crowd incident management.

Using the (NP-hard) Mixed Integer Programming (MIP), we formulate an optimization problem of minimizing the maximum scheduling makespan. To address the scheduling overhead for a large system of devices (up to 100), we use a branch-and-bound-based solution and show that by allowing a 1% optimality gap, a scheduling solution becomes feasible as it can be produced faster than the resulting minimum or maximum makespan by an entire order of magnitude.

The rest of the paper is organized as follows: Section II discusses related work. In Section III-A and Section III-E, we motivate the need for accurate multi-edge resource scheduling. Sections III-B, IV-A, and IV-B describe our proposed offloading system approach. Subsection III-C and Section III-D formulate the job scheduling problem and optimization steps. Section IV details the results from our trace-driven simulations. Lastly, Section V concludes the paper.

II. Related Work

Several computation offloading frameworks have been proposed in recent years including heuristic-based algorithms that offer negligible overhead for handling requests. These algorithms typically perform poorly in dynamic environments [6], or are only evaluated for a limited number of devices [7]. Alternatively, the recent popular approaches tend to be *data-driven*, meaning they use historical information to build predictive models in order to forecast offloading behavior [8]. MALMOS is an exemplar machine learning-based runtime scheduler that outperforms static scheduling policies [9].

The main limitations of many data-driven approaches are that they are only evaluated for a small number of devices, use a limited number of features for training, and are often comprised of simple offloading decisions (i.e. deciding between offloading or executing locally) [9]. Furthermore, related methods focus on code partitioning rather than data migration which requires developers to re-factor sections of their code in order to enable offloading in their applications. Conversely, other approaches focus instead on data migration which is device agnostic and is only concerned with processing the user's raw data [10][11]. Our recent prior work provides researchers with publicly available datasets and testbed configurations for reproducing experimental edge computing methodologies [12].

Algorithms that schedule offloading decisions based on the device profile and network information perform significantly better than those that assume a static network model [13]. Researchers have previously addressed the problem of server choice for offloading tasks when multiple servers are available [3][14]. Other works have approximated the best offloading node using heuristics and statistical estimation methods for wireless network performance [15]. Several approaches have re-purposed the popular job shop scheduling problem to optimize their objective of choice (e.g. latency, energy consumption, throughput, etc.) [16][17].

To the best of our knowledge, none of the prior approaches consider the use of data-driven estimation techniques for improving optimization in a multi-edge resource scheduling context for a public safety application context.

III. Multi-Edge Resource Scheduling

A. Protest Crowds Incident Management

As mentioned in Section I, an exemplar use case for a distributed edge computing service involves offloading facial recognition tasks from mobile devices to nearby cloudlets in order to provide public safety officials with increased situational awareness. Visual data processing in a timely manner with limited networking and computing resources is essential in decision making of incident commanders e.g., in "force escalation" decisions that rely on data processing related to long time periods of crowd status monitoring.

In order to provide a generalized solution for multi-edge scheduling in a protest crowds incident management scenarios, we begin by discussing the inherent challenges present in such a visual data processing system:

- Environmental conditions and large numbers of cloudlets and devices may cause noisy and congested network conditions
- Estimates about offloading costs prove challenging due to the nonlinear behavior of wireless networks
- Remote mobile devices are usually low-power and have limited processing and storage capabilities
- Scarce computing resources are available for performing offloading functionality (high device-to-server ratio)

For the purposes of scheduling offloading tasks in such a visual data processing system, we make the following assumptions:

- Our focus is on theater-scale edge computing where a maximum of 100 nodes is adequate for our use-case
- Computation should always be offloaded instead of executed locally due to the global facial database accessible only by the edge servers
- Network and device states are observable during runtime
- Every cloudlet is capable of accepting new jobs

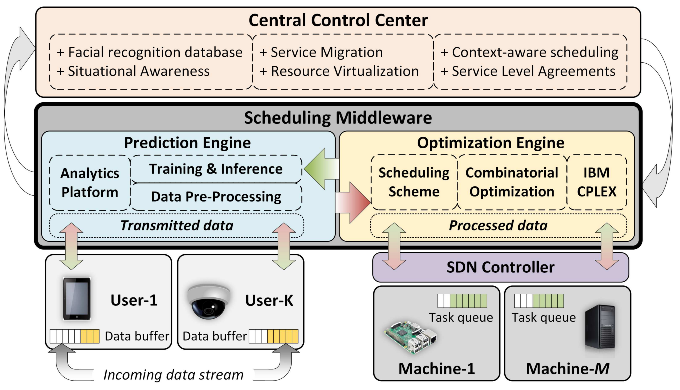


Fig. 2. Data-driven edge computing system that shows the interplay between the prediction and optimization engines for scheduling offloading tasks involving multiple users and their devices. User data is transmitted to a central node that acts as the scheduling middleware and contains the software stack for performing the prediction and optimization functions. A central control center acts as a remote authority for gathering processed data and implements context-specific policies to aid in incident commander decision making.

- Scheduling middleware receives the metadata for all jobs for prediction and offloading decision-making
- Only involved public safety management personnel have access to the edge network; no other traffic is present

B. Proposed Computation Offloading Approach

We propose a data-driven computation offloading system for use in multi-edge resource scheduling applications such as e.g., the protest crowds incident management use case. To overcome the inaccuracy and measurement overhead present in traditional static estimation techniques, our offloading system employs data-driven models for predicting the transmission and computation times based on historical network and device data. The use of ML models allows for more accurate network performance predictions without the added measurement overhead, which can be problematic for optimizing offloading schedules in multi-edge resource environments.

Figure 2 shows an overview of our proposed data-driven edge computation offloading system. An analytics platform can be used for profiling network and device parameters, which are then used as input features for designing the predictive models. The resulting cost predictions are then fed into an optimization engine in order to derive offloading schedules. Offloading decisions can then be facilitated by updating forwarding tables of routers using Software-Defined Networking or similar technologies. Additionally, scheduling policies can be implemented on the scheduling middleware to allow for different edge computing scenarios (e.g. energy-aware scheduling, prioritizing traffic for emergency personnel, etc.). In this paper, our scope is limited to developing and evaluating the scheduling middleware, and higher-layer Control Center orchestration is beyond the scope of this paper.

C. Job Shop Scheduling Problem

The multi-edge computation offloading problem in our crowd protest incident management use case can be formulated as the popular online job shop scheduling problem [18]. Our scheduling problem objective is to minimize the maximum makespan. To this aim, we model computation offloading requests from remote devices as *tasks* for scheduling, while the cloudlets are modeled as *machines* that process these tasks.

We start our formulation of the job shop scheduling problem by introducing the following binary variable:

$$x_{jk}^{i} = \begin{cases} 1, \text{ if task } i \text{ is assigned at machine } j \text{ position } k, \\ 0, \text{ otherwise,} \end{cases} \quad (1)$$

with $i \in N$, $j \in M$, and $k \in \{1, ..., |N|\}$, where N is the set of all tasks and M is the set of all machines.

We then describe the assignment constraints which are essential for the job scheduling problem. The first constraint type ensures that each task is assigned to exactly one machine position as follows:

$$\sum_{j=1}^{M} \sum_{k=1}^{N} x_{jk}^{i} = 1, \forall i \in N. \quad (2)$$

The second constraint type ensures that at most m tasks are assigned to each machine position as follows:

$$\sum_{i=1}^{N} x_{jk}^{i} \leq m, \forall j \in M, k \in N. \quad (3)$$

Note that in general when the edge cloud provider policy $m > 1$, multiple tasks can be processed in parallel at the position k of the machine j.

The following (optional) constraint type can be used to ensure that each machine's positions are filled sequentially from the beginning:

$$\sum_{i=1}^{N} x_{jk}^{i} \leq \sum_{i=1}^{N} x_{j(k-1)}^{i}, \forall j \in M, k \in \{2...N\}. \quad (4)$$

Note that constraints in Equations 2 and 3 are sufficient for a feasible task assignment when no specific processing task order is required (also known as precedence constraints). Note also that we are processing batches of independent images, and thus, we do not require having precedence constraints in our problem. However, specific task order can be required, e.g. for tracking computer vision applications that use results of preceding video frames to track current frames [19].

Let us also introduce a positive continuous variable y_{jk} that denotes the time when m tasks at position k of machine j finish processing. This time should be greater than or equal to the sum of the time when machine j finishes processing position $k-1$ and the processing/communication time of the m tasks. Thus, we have to satisfy the following set of constraints:

$$y_{jk} \geq y_{j(k-1)} + (t_{ij}^{p} + t_{ij}^{c}) x_{jk}^{i}, \forall i \in N, j \in M, k \in N, \quad (5)$$

where t_{ij}^{p} and t_{ij}^{c} are the processing and communication times of task i at machine j. Note also that $y_{j0} \geq 0$ is a machine offset time — the time when machine j finishes processing of the previously scheduled tasks which come online — and $y_{j0} = 0$ for the offline job scheduling problem when all tasks are known in advance.

TABLE I
SYMBOLS AND NOTATIONS OF OPTIMIZATION PROBLEM

Sets:		
N	$\triangleq$	Set of tasks (offloading mobile devices)
M	$\triangleq$	Set of machines (edge servers)
Variables:		
x_{jk}^{i}	$\triangleq$	Binary variable equal to 1 if task i is assigned at machine j position k and 0 otherwise
y_{jk}	$\triangleq$	Positive continuous variable that denotes the time when m tasks at position k of machine j will finish being processed
α	$\triangleq$	Continuous variable that denotes the maximum makespan
Parameters:		
t_{ij}^{p}	$\triangleq$	Task i processing time at machine j
t_{ij}^{c}	$\triangleq$	Task i transmission time to machine j
Policies:		
m	$\triangleq$	The maximum number of tasks that can be processed in parallel at the same machine position

Let us finally introduce a continuous variable α that denotes the maximum makespan among all machines. Thus, α has to satisfy the following set of constraints:

$$y_{jN} \leq \alpha, \forall j \in M. \quad (6)$$

Having both variables and constraints discussed, the online job shop scheduling problem that minimizes the maximum makespan α can be formulated as following:

$$\begin{aligned} \text{minimize} \quad & \alpha \\ \text{subject to} \quad & (2) - (6) \\ & x_{jk}^{i} \in \{0,1\}, \quad \forall i \in N, j \in M, k \in N \\ & y_{jk} \geq 0, \quad \forall j \in M, k \in N, \end{aligned} \quad (7)$$

where all variables, parameters and sets are listed in Table I.

D. Reducing the overall scheduling makespan

In order to solve Equation 7, which is known to be an NP-hard problem, a method such as Mixed Integer Programming (MIP) is suitable. To simplify this problem, we first omit optional constraints in Equation 4 and ignore all idle intermediate positions of each machine. Secondly, we subsume constraints in Equation 5 with a reduced set of following constraints:

$$y_{jk} \geq y_{j(k-1)} + \sum_{i=1}^{N} (t_{ij}^{p} + t_{ij}^{c}) x_{jk}^{i}, j \in M, k \in N. \quad (8)$$

Note that this substitution is valid for $m = 1$ policy, i.e. when a traditional job scheduling problem formulation is used where only one task can be processed at a time [20]. Finally, we fix all $x_{jk}^{i} = 0$ if the device i is more than 1 wireless hop away from the machine j. This is done to avoid the high bitrate reduction and loss rate common for multi-hop wireless ad-hoc mesh networking [21].

To solve Equation 7, we use the latest IBM ILOG CPLEX v.12.8 [22] and a High Performance Computing (HPC) Cloud server with *two 16-core Intel Xeon Gold 6142 CPUs at 2.6 GHz*, *384GB ECC DDR4-2666 RAM* running *Linux Ubuntu 18.04 STD* that is allocated in the CloudLab platform [23]. Our evaluation results in Section IV show that the 1% optimality gap solution scales an order of magnitude faster than the minimum makespan for moderate number of devices (up to 100). This occurs at the expense of a 2-3 second increase in maximum makespan, with the average maximum makespan value being 5-6 minutes.

E. Limitations of Current Estimation Techniques

Static transmission rate estimators can degrade the quality of calculated schedules over time, particularly when relying on measuring the link bitrate or using the *iperf* utility to calculate the transmission time of data [12]. These inaccuracies can prevent scheduling algorithms from making efficient offloading decisions, especially in situations where network and device states are dynamic. Another common issue with static measurement techniques is that they are not able to make predictions *a priori*, and therefore need to take measurements periodically. This can lead to increases in network overhead and subsequent delays in scheduling large number of tasks.

The plots in Figure 3 show the transmission and processing characteristics for a variety of machine types and network conditions collected in our experiments. The benchmark used in Figure 3a is from a study that used dlib for face detection [24] on the same image dataset [25] used in our facial recognition application based on the `dlib` library [26]. As expected, there is a positive correlation between processing speed and hardware type. Additionally, we observe small differences in processing speed between image resolutions as well as a small variance within each image resolution group which suggest that the appearance of faces isn't a compounding factor in resulting processing times. In Figure 3b, each method provides estimates that are much faster than the actual transmission time measured, which we believe is due to the simplicity of the underlying estimation techniques.

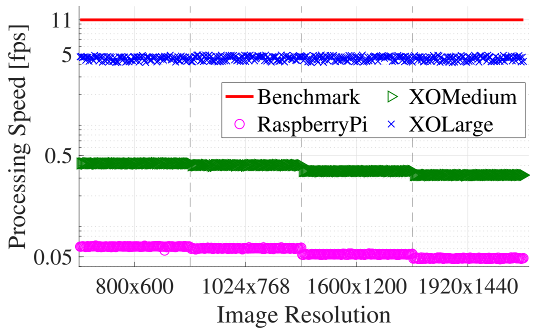
IV. Evaluation Results

A. Data Collection from Real-World Experiments

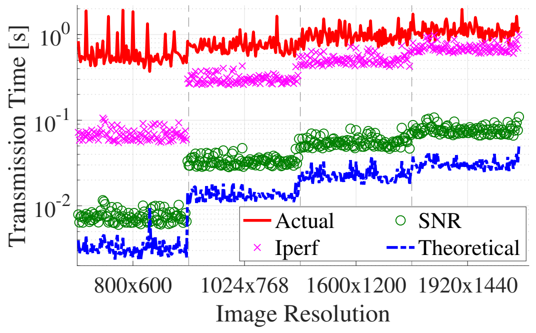
We conducted a series of realistic wireless network experiments to measure the transmission and processing delays for a facial recognition service. The image dataset was subsampled from the ChokePoint facial recognition dataset [25]. The dataset was then partitioned into four groups of differing image resolutions (800x600, 1024x768, 1600x1200, 1920x1440), with 100 images per group. We divided the data into transmission and processing datasets in order to more accurately model each independent factor.

To measure transmission time, we configured a Mikrotik router using the standard mesh routing protocol. A Raspberry Pi served as an IoT-based cloudlet while using a wireless laptop to vary the distance between the client and server. In total, 2000 transmission delay samples were recorded, some of which were removed due to being statistical outliers (i.e. having a transmission time outside of three standard deviations from the mean). This was done to reduce the impact of presumably unrepresentative outliers on the models.

Processing time was captured by running a facial recognition application that was implemented using the popular `dlib` machine learning library [26] on each group of images.



(a) Processing speed based on hardware type



(b) Transmission time estimates based on measurement type

Fig. 3. Experimental analysis for a set of image resolutions relating to: (a) processing characteristics for various hardware profiles, and (b) transmission estimates calculated using various methods and measurement tools.

Processing speed was measured in terms of frames per second, while profiling the processor clock speed, RAM, and cache size. This was done because they are most indicative of the processing delay characteristics of the server. We measured the processing characteristics of five different machine types: Raspberry Pi v3, three ExoGENI nodes with various processing capabilities, and one instaGENI XEN-based virtual node. In total, 2000 processing delay samples were recorded, with no outliers removed.

Table II shows the 'features of interest' that were collected during our experiments. We collected features related to the networking hardware, processing capabilities of the machines, and attributes of the data to being offloaded. Signal level and link quality are both measures of the strength of the connection, while noise, packet loss, and jitter are measures of connection interruptions. Bitrate and mean RTT are valuable measurements of the capacity and speed of the connection. Note, we have released this dataset for public use in order to foster more research into improving offloading systems [27].

B. Predicting Offloading Costs with Machine Learning

Model selection and evaluation. We evaluated a mixture of linear and nonlinear approaches in order to best survey the accuracy of ML models for predicting transmission and processing delays. All input feature values were normalized, and the models were implemented using RapidMiner [28]. We chose Linear Regression, Multi-Layer Perceptron, and Support Vector Regression as our linear models, and Decision Tree,

TABLE II
COLLECTED FEATURES FOR TRAINING MODELS

Transmission Dataset	*Processing* Dataset
Data Attributes:	
Image Height (pixels)	Image Height (pixels)
Image Width (pixels)	Image Width (pixels)
Data Size (bytes)	Data Size (bytes)
Hardware Parameters:	
Signal Level (dB)	Processor Speed (MHz)
Noise Level (dB)	RAM (Kb)
Link Quality (%)	L1d Cache (Kb)
Bitrate (Mb/s)	L1i Cache (Kb)
Packet Loss (%)	L2 Cache (Kb)
Mean RTT (sec)	-
Jitter (sec)	-
Mobility:	
Distance (m) $\pm$ 15	N/A
Prediction Parameters:	
Transmission Time (sec)	Processing Speed (fps)

Random Forest, and k-Nearest Neighbors as our nonlinear models. We anticipated that the nonlinear models would better capture the nonlinearities inherent in wireless networks.

Our performance results in Table III show the benefits of using data-driven models for predicting transmission and processing speed over traditional estimation techniques that use *iperf* utility and the facial recognition benchmark as a baseline. Not only do the ML models on average have a much lower RMSE than the baseline estimates, they also have much better precision in terms of the variance observed for predictions. These findings suggest that using data-driven models for predicting offloading costs may lead to improved and more reliable scheduling performance.

TABLE III
MACHINE LEARNING PREDICTOR RESULTS

Model Type	Transmission Time RMSE	Processing Speed RMSE
Baseline Estimate	2.700 $\pm$ 2.223	8.285 $\pm$ 6.520
Decision Tree	0.877 $\pm$ 0.227	**0.231 $\pm$ 0.040**
k-Nearest Neighbors	0.826 $\pm$ 0.174	0.250 $\pm$ 0.041
Linear Regression	0.835 $\pm$ 0.139	0.748 $\pm$ 0.041
Multi-Layer Perceptron	0.814 $\pm$ 0.179	0.307 $\pm$ 0.055
Random Forest Regressor	**0.813 $\pm$ 0.263**	0.741 $\pm$ 0.147
Support Vector Machine	0.837 $\pm$ 0.269	0.878 $\pm$ 0.079

We hypothesize that the Random Forest Regressor (RFR) and Decision Tree (DT) models perform the best due to several underlying factors inherent in their design. For instance, decision trees have the ability to produce fine-grained decision branches which tend to perform well for modeling various complex relationships. For more complicated relationships such as those exhibited in wireless networks, RFR is ideal because it is composed of a collection of decision trees, where each tree's output is used in a voting system to determine the final output of the forest. This voting system, known as *bagging* reduces the bias and variance of the model.

C. Protest management case study results

Simulation settings. Our Java-based simulation environment is composed by the latest IBM ILOG CPLEX v.12.8 [22] and a High Performance Computing (HPC) Cloud server with two 16-core Intel Xeon Gold 6142 CPUs at 2.6 GHz, 384GB ECC DDR4-2666 RAM running Linux Ubuntu 18.04 STD allocated in the CloudLab platform [23].

Motivated by the challenges of protest crowds incident management described in Section III-A, we generate a 1 km^2 area similar to a typical university campus scale as shown in Figure 4. We then uniformly generate from 1 to 5 protest incident epicenters within this area. After that, we place both servers and devices following a normal sampling distribution $\mathcal{N}(\vec{epi}, \sigma^2)$, where the *mean* corresponds to some protest epicenter coordinates $\vec{epi}$, and the *standard deviation* $\sigma = \frac{1}{6}$ km. By default, we place a total of 50 devices, and use a device-to-server ratio of 10 unless stated otherwise.

Finally, each device offloads from 1 to 10 images of some resolution uniformly selected from an image resolution set $\{800p, 1024p, 1600p, 1920p\}$. We then select a corresponding trace for each device i and server j based on the image resolution, server and device proximity, server type, etc. We then use these traces to obtain either ground-truth communication t^c_{ij} and processing t^p_{ij} times or estimate/predict these times.

After setting up the environment, we attempt to offload each task to exactly one server in a way to minimize the maximum makespan among all servers. In this simulation, our main goal is to understand how the prediction/estimation techniques used for t^c_{ij} and t^p_{ij} affects the optimal scheduling when ground-truth data is used instead.

Comparison methods and metrics. We compare the solution to Equation 7 when ground-truth values of t^c_{ij} and t^p_{ij} are known in advance (intractable in practice) with its solutions when t^c_{ij} and t^p_{ij} are either estimated or predicted. We refer to the ground-truth-based Equation 7 solution as the optimal solution Opt.

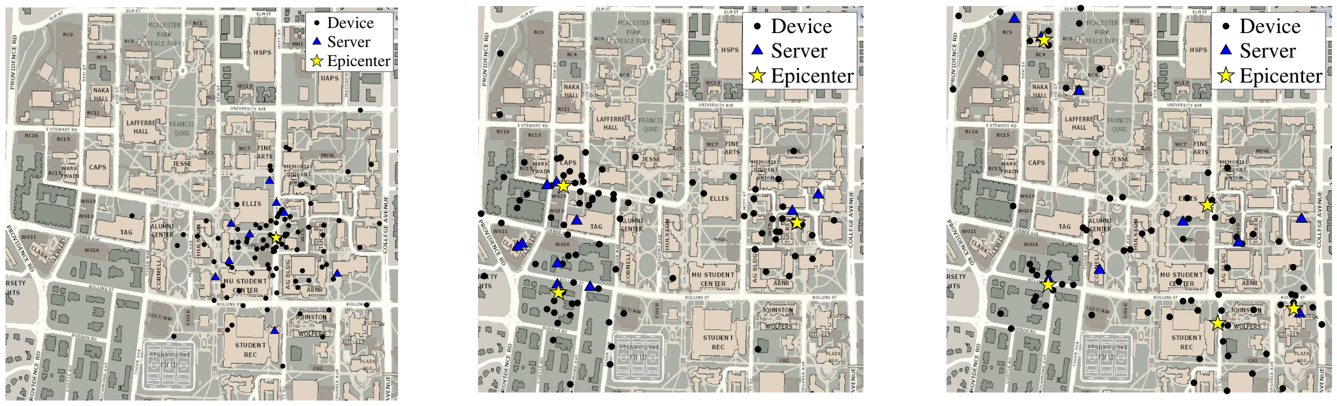
To estimate the communication time t^c_{ij}, we use the following formula:

$$t^c_{ij} = \frac{data\_size_i}{throughput_{ij}}, \tag{9}$$

where the average $throughput_{ij}$ value between device i and server j is estimated using *iperf* utility. We then estimate the processing time t^p_{ij} as follows:

$$t^p_{ij} = \frac{N}{processing\_speed}, \tag{10}$$

where N is a number of images, and $processing\_speed$ is an average face recognition speed (measured in frames per second) for a specific image resolution obtained from prior benchmark results [24]. We refer to the solution of Equation 7 that is estimated via Equations 9 and 10 values of t^c_{ij} and t^p_{ij} as the estimated solution Est.



(a) (b) (c)

Fig. 4. Geo-spatial maps showing an example of device and server placement during trace-driven simulations for the case of: 1(a), 3(b), and 5(c) Epicenters, respectively. In our crowd protest incident case study, we represent each protest site as an Epicenter, with a set of heterogeneous resources available for processing offloading requests from remote devices.

Finally, we use our ML models to predict values t_{ij}^c and t_{ij}^p, and represent the two best pairs of predictors as $(P1)$ and $(P2)$. For predicting t_{ij}^c and t_{ij}^p, we use the Random Forest and Decision Tree models for $(P1)$ and the Multi-Layer Perceptron and k-Nearest Neighbors models for $(P2)$, respectively.

The related solutions are compared using two metrics: the Equation 7 objective — the maximum makespan α (the lower the better) and its optimality gap measured as a percentage. We measure the optimality gap by substitution of ground-truth values t_{ij}^c and t_{ij}^p to the final schedule in order to retrieve the ground-truth α. We then estimate the increase in percentage for Est, $P1$ and $P2$ solutions with respect to the Opt. Additionally, we compare the time it takes to solve Equation 7 with its maximum/minimum makespan, and when 1% optimality gap is allowed.

Scheduling Performance. Our trace-driven simulations produced two significant results, described below:

(i) **The branch-and-bound-based solution to Equation 7 scales sufficiently for moderate wireless network sizes of up to 100 nodes when a 1% optimality gap is allowed.**

Our trace-driven simulations indicate that optimally solving Equation 7 is NP-hard and can be intractable for moderate scale networks of 50-100 nodes. To address this intractability, we can allow a 1% optimality gap that enables the branch-and-bound-based solution using IBM CPLEX to scale sufficiently for a large number of nodes. Figure 5a shows that for a 100-node edge network, a solution to Equation 7 with a 1% optimality gap can be produced in an order of magnitude less time than either the minimum or maximum makespan requires. Because the optimization problem can be solved in significantly less time, producing solutions will not increase the blocking probability of our online offloading schemes. The relatively shorter scheduling time also allows for the scheduling middleware to begin another batch of tasks while the previous batch is still being processed. Note however that for the case of more than 100 nodes, existing (polynomial) greedy or approximation algorithms can be used for solution.

(ii) **Data-driven resource scheduling matches the ground-truth optimal in 70% of cases and has an α value no more than 3 times higher in the worst case.**

From our simulations, we also found that estimation based edge offloading solutions produce significantly worse scheduling results with respect to both the optimal ground-truth solution and our data-driven approach. Figure 5c shows estimation-based offloading only matches the optimal scheduling in 50% of cases, and can have a maximum makespan over ten times higher in the worst case. In comparison, our data-driven scheduling system matches the optimal ground-truth offloading 70% of the time and produces 2-3 times higher maximum makespan in the worst case scenario.

V. Conclusion and Future Work

In this paper, we propose the use of data-driven techniques for predicting offloading costs in multi-edge resource scheduling that is used in conjunction with an optimization engine to provide near-optimal scheduling solutions. We benchmarked several state-of-the-art ML models in order to identify the trade-offs between each model, and evaluated the models to find the most appropriate ones for the data processing needs in a protest crowd incident management use case. We also investigated the scalability of our approach in a multi-edge system testbed setup by performing a set of trace-driven simulations for a large number of devices and machines. By using a standard optimization approach such as Mixed Integer Programming, we were able to derive near-optimal schedules when allowing a 1% optimality gap.

In the future, we plan to explore online and incremental machine learning models to investigate the robustness of ML-based predictions for highly dynamic networking and computing environments (e.g. user mobility, link failures. etc.). In addition, consideration of deadline constraints for real-time decision making scenarios in protest crowds incident management is also part of potential future work.

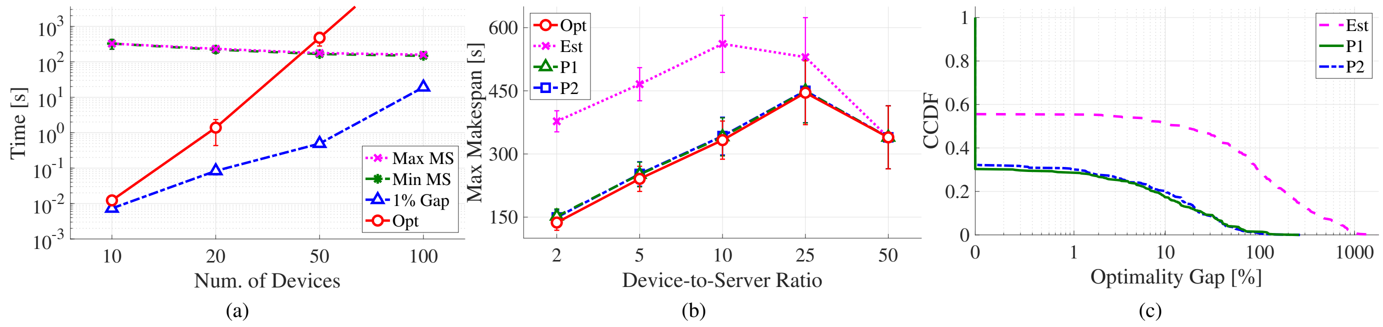


Fig. 5. Evaluation results from trace-driven simulations. (a) Equation 7 scalability results of its optimal versus 1% optimality gap solutions with respect to the min/max scheduling makespan (MS). (b) Resulting maximum makespan based on a server availability, i.e., device-to-server ratio. (c) Complement Cumulative Distribution Function (CCDF) of the ground-truth optimality gap for estimation/prediction-based offloading methods.

ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Award Numbers: CNS-1647182 and CNS-1359125. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] G. A. Akpakwu, B. J. Silva, G. P. Hancke, and A. M. Abu-Mahfouz, "A survey on 5G networks for the Internet of Things: Communication technologies and challenges," *IEEE Access*, vol. 6, pp. 3619–3647, 2018.

[2] M. Satyanarayanan, Z. Chen, K. Ha, W. Hu, W. Richter, and P. Pillai, "Cloudlets: at the leading edge of mobile-cloud convergence," in *2014 6th International Conference on Mobile Computing, Applications and Services (MobiCASE)*. IEEE, 2014, pp. 1–9.

[3] P. Mach and Z. Becvar, "Mobile Edge Computing: A Survey on Architecture and Computation Offloading," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1628–1656, thirdquarter 2017.

[4] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, "A Survey of Computation Offloading for Mobile Systems," *Mobile Networks and Applications*, vol. 18, no. 1, pp. 129–140, Feb 2013. [Online]. Available: https://doi.org/10.1007/s11036-012-0368-0

[5] D. Chemodanov, F. Esposito, P. Calyam, and A. Sukhov, "A constrained shortest path scheme for virtual network service management," *IEEE Transactions on Network and Service Management*, 2018.

[6] S. Sthapit, J. R. Hopgood, N. M. Robertson, and J. Thompson, "Offloading to neighbouring nodes in smart camera network," in *Signal Processing Conference (EUSIPCO), 2016 24th European*. IEEE, 2016, pp. 1823–1827.

[7] F. Samie, V. Tsoutsouras, L. Bauer, S. Xydis, D. Soudris, and J. Henkel, "Computation offloading and resource allocation for low-power iot edge devices," in *Internet of Things (WF-IoT), 2016 IEEE 3rd World Forum on*. IEEE, 2016, pp. 7–12.

[8] M. Kulin, C. Fortuna, E. De Poorter, D. Deschrijver, and I. Moerman, "Data-driven design of intelligent wireless networks: An overview and tutorial," vol. 16, no. 6. Multidisciplinary Digital Publishing Institute, 2016, p. 790.

[9] H. Eom, R. Figueiredo, H. Cai, Y. Zhang, and G. Huang, "MALMOS: Machine Learning-Based Mobile Offloading Scheduler with Online Training," in *3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, March 2015, pp. 51–60.

[10] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the sixth conference on Computer systems*. ACM, 2011, pp. 301–314.

[11] A. Crutcher, C. Koch, K. Coleman, J. Patman, F. Esposito, and P. Calyam, "Hyperprofile-Based Computation Offloading for Mobile Edge Networks," *2017 IEEE 14th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, pp. 525–529, 2017.

[12] J. Patman, M. Alfarhood, S. Islam, M. Lemus, P. Calyam, and K. Palaniappan, "Predictive Analytics for Fog Computing using Machine Learning and GENI," in *Computer Communications Workshops (INFOCOM WKSHPS), 2017 IEEE Conference on*. IEEE, 2018.

[13] Z. Liu, X. Zeng, W. Huang, J. Lin, X. Chen, and W. Guo, "Framework for Context-Aware Computation Offloading in Mobile Cloud Computing," in *2016 15th International Symposium on Parallel and Distributed Computing (ISPDC)*, July 2016, pp. 172–177.

[14] K. Sato and T. Fujii, "Radio Environment Aware Computation Offloading with Multiple Mobile Edge Computing Servers," in *2017 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, March 2017, pp. 1–5.

[15] M. A. Alsheikh, S. Lin, D. Niyato, and H. P. Tan, "Machine Learning in Wireless Sensor Networks: Algorithms, Strategies, and Applications," *IEEE Communications Surveys Tutorials*, vol. 16, no. 4, pp. 1996–2018, Fourthquarter 2014.

[16] P. Agrawal and S. Rao, "Energy-aware scheduling of distributed systems," *IEEE Transactions on Automation Science and Engineering*, vol. 11, no. 4, pp. 1163–1175, 2014.

[17] P. Lindberg, J. Leingang, D. Lysaker, K. Bilal, S. U. Khan, P. Bouvry, N. Ghani, N. Min-Allah, and J. Li, "Comparison and analysis of greedy energy-efficient scheduling algorithms for computational grids," *Energy Aware Distributed Computing Systems, John Wiley & Sons, Hoboken, NJ, USA*, 2012.

[18] M. L. Pinedo, *Scheduling: theory, algorithms, and systems*. Springer, 2016.

[19] R. Gargees, B. Morago, R. Pelapur, D. Chemodanov, P. Calyam, Z. Oraibi, Y. Duan, G. Seetharaman, and K. Palaniappan, "Incident-Supporting Visual Cloud Computing Utilizing Software-Defined Networking," vol. 27, no. 1. IEEE, 2017, pp. 182–197.

[20] D. Applegate and W. Cook, "A computational study of the job-shop scheduling problem," *ORSA Journal on computing*, vol. 3, no. 2, pp. 149–156, 1991.

[21] G. R. Hiertz, D. Denteneer, S. Max, R. Taori, J. Cardona, L. Berlemann, and B. Walke, "IEEE 802.11s: The WLAN Mesh Standard," *IEEE Wireless Communications*, vol. 17, no. 1, pp. 104–111, February 2010.

[22] IBM, "ILOG CPLEX v.12.8. User's Manual for CPLEX," https://www.ibm.com/products/ilog-cplex-optimization-studio.

[23] R. Ricci, E. Eide, and C. Team, "Introducing cloudlab: Scientific infrastructure for advancing cloud architectures and applications," *; login:: the magazine of USENIX & SAGE*, vol. 39, no. 6, pp. 36–38, 2014.

[24] Y. An, J. Wu, and C. Yue, "CNNs for Face Detection and Recognition," https://github.com/fusio-wu/CS231A_project, 2017.

[25] Y. Wong, S. Chen, S. Mau, C. Sanderson, and B. C. Lovell, "Patch-based Probabilistic Image Quality Assessment for Face Selection and Improved Video-based Face Recognition," in *IEEE Biometrics Workshop, Computer Vision and Pattern Recognition (CVPR) Workshops*. IEEE, June 2011, pp. 81–88.

[26] D. E. King, "Dlib-ml: A Machine Learning Toolkit," *Journal of Machine Learning Research*, vol. 10, pp. 1755–1758, 2009.

[27] "Data-Driven Scheduling Raw Experimental Data," https://missouri.box.com/v/DDS-RawExpData.

[28] RapidMiner, "RapidMiner," https://rapidminer.com/, 2001–2018.