Managing Edge Offloading for Stochastic Workloads with Deadlines

Agrim Bari
The University of Texas at Austin
Austin, TX, USA
agrim.bari@utexas.edu

Gustavo de Veciana
The University of Texas at Austin
Austin, TX, USA
deveciana@utexas.edu

Kerstin Johnsson
Intel Corporation, California
Santa-Clara, CA, USA
kerstin.johnsson@intel.com

Alexander Pyattaev
YL-Verkot OY
Tampere, FI
alexander.pyattaev@yl-verkot.com

ABSTRACT

Increasing demand for computationally intensive jobs on mobile devices is driving interest in computation offloading to the edge/cloud servers. This paper presents a comprehensive framework for managing offloading of stochastic and heterogeneous user(s)-generated jobs while considering job deadlines and congestion on wireless channels and edge/cloud servers. The goal of offloading is to maximize either the net computational work offloaded or power savings. We propose a class of policies called Predictive Abandonment (PA), where users opportunistically cut and offload jobs but abandon offloading if they predict that communication and computation delays will preclude on-time completion. Although these user-driven policies are desirable from an implementation perspective and achieve relatively good performance, they cannot coordinate tradeoffs amongst users with heterogeneous job types. To address this, we propose a complementary approach to coordinate offloading based on Probabilistic Admission Control and Cut Assignment (PACCA). When combined with PA, it delivers significant offloading benefits. We also develop an upper bound on the benefits of offloading, which can serve as a baseline for evaluating the additional gains of more complex offloading policies. We evaluate these policies via simulation for a range of loads and job profiles, demonstrating robust gains over a naive greedy offloading policy and near-optimal performance in some settings. Furthermore, we assess the robustness of PACCA + PA to imperfect knowledge of offered job rates.

CCS CONCEPTS

• **Networks → Network performance analysis**.

ACM Reference Format:
Agrim Bari, Gustavo de Veciana, Kerstin Johnsson, and Alexander Pyattaev. 2023. Managing Edge Offloading for Stochastic Workloads with Deadlines. In *Proceedings of the Int'l ACM Conference on Modeling Analysis and Simulation of Wireless and Mobile Systems (MSWiM '23), October 30-November 3, 2023, Montreal, QC, Canada.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3616388.3617515

MSWiM '23, October 30-November 3, 2023, Montreal, QC, Canada

ACM ISBN 979-8-4007-0366-9/23/10.
https://doi.org/10.1145/3616388.3617515

1 INTRODUCTION

Next-generation applications and the MEC fabric. A new generation of applications powered by machine learning, e.g., AR/VR/XR, autonomous navigation, and photo editors, is pushing the computational and energy limits of mobile devices. One way to overcome these limitations while addressing low latency and privacy requirements is for users to (partially) offload computationally intensive jobs to shared Mobile Edge Computing (MEC) resources. By combining mobile devices' sensing, communication, and computation capabilities with computation at nearby edge servers and/or more distant cloud servers, one can envision a computation-communication fabric that can cost-effectively address the most demanding mobile users' compute jobs.

Benefits of compute job offloading. There are several benefits mobile devices can reap from offloading. First, devices with insufficient computation resources may only be able to complete a job through offloading. Second, even if a device can complete a job, it may opt to offload to save energy and/or reduce its computational work to allow for computation of other jobs. Third, offloading a job might enable a mobile device to leverage powerful MEC/cloud computation resources to speed up job completion. In this paper, we introduce policies that maximize the amount of computational work mobile devices offload or the amount of energy devices can save while completing jobs within their deadlines.

Managing compute job offloading. To realize these benefits, one must orchestrate offloading across various resources and account for the possible costs of doing so. In general, offloading a compute job may include the following steps: (i) (partially) computing the job on the device; (ii) transferring data to an edge/cloud server via a shared wireless link for performing the remaining computational work; (iii) performing further computation on the edge/cloud server; and (iv) transferring the results back to the device. These steps may involve shared computation resources on the mobile device, shared wireless channels, and shared edge/cloud computation resources, which may become congested under stochastic loads. Such systems also face significant heterogeneity in terms of devices' computation and/or communication capabilities as well as running heterogeneous compute jobs with different Quality-of-Service (QoS) requirements, e.g., constraints on completion time.

To address these complex challenges, we present an offloading framework that combines an offload admission control policy with a lightweight user-driven offload abandonment policy.

DAG job cutting and offloading. In this work, we focus on offloading compute jobs that can be roughly modeled as linear Directed Acyclic Graphs (DAGs), where the nodes represent computational *sub-tasks*, e.g., *Deep Neural Network (DNN) layers*, and edges represent the *data dependencies* and potential cut locations between sub-tasks, e.g., see Figure 1. In our work, we use the findings of [19] to only select a single cut location for time-sensitive jobs. The authors show that, under a block fading/Markovian stochastic channel, the optimal policy for offloading computational work from a local device to an edge/cloud server would at most offload once. This result assumes a congestion-free system, a device expends more power on processing a job than sending/receiving data, edge server processes faster, and flexibility to execute sub-tasks on either device or edge server. In the shared MEC fabric, a job's optimal cut location depends on its computation-communication requirements per cut location, completion deadline, current wireless network conditions, and the computational resources of the device it is generated on relative to available networked edge compute servers. It also depends on the operator's preferences [9] (e.g., rewards and/or fairness). Thus, some form of coordination of offloading decisions is necessary.

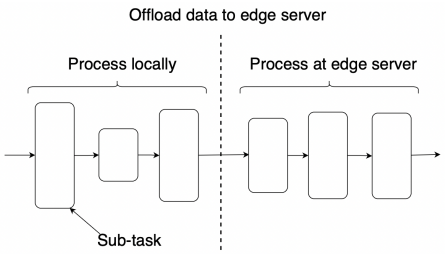


Figure 1: Cutting and offloading of a linear DAG.

Applicability of DAG model. There are several works [5–7, 11, 15] in the literature that embrace the linear DAG model as an effective abstraction/approximation of jobs that might particularly benefit from offloading. The underlying driver is the layered structure of DNNs, currently used in several applications ranging from image classification, facial, digit, and speech recognition to many others. We believe that a substantial volume of future workloads will have structures like linear DAGs, where there is flexibility to cut and offload jobs. For a general DAG, the researchers [18] have extended their offloading policy for a linear DAG to a general DAG by exploiting the notion of a DAG's critical path.

1.1 Related work

Offloading problem. Offloading of compute jobs to MEC has been widely studied in the literature, which can be divided into two main categories: binary offloading and partial offloading. In binary offloading, a job is either executed on the device or offloaded to one or more edge servers for execution, intending to optimize performance metrics such as average computation delay or energy consumption. In most cases, the binary offloading problem is NP-hard, and various heuristics, approximation, and stochastic approaches have been proposed, see e.g., [1, 3, 4, 10, 12, 16, 17, 20]. Researchers in [1] explore the behavior of users when making decisions about offloading compute jobs in a multi-MEC server environment. They propose a Prospect Theory-based solution, considering users' risk-seeking or loss-aversion behavior. However, [1] has limitations, focusing on jobs consisting of independent sub-tasks and lacking a strict deadline constraint. Similarly, authors in [17] propose a game theoretic solution.

In [4], authors address fairness and maximum delay tolerance in hybrid fog/cloud systems by jointly optimizing computation migration and resource allocation (including computing and bandwidth). They propose a suboptimal algorithm to solve the formulated mixed integer non-linear programming problem. Another paper, [20], focuses on joint computation offloading decisions, resource allocation, and content caching strategy. The authors transform the problem into a convex form and solve it in a distributed and efficient manner using optimization theory tools. Given a set of jobs and multiple edge servers, [3] proposes an approximate solution considering dynamic voltage frequency scaling for mobile devices. Their heuristic algorithm optimizes job offloading and frequency scaling decisions. However, all of the aforementioned works focus on static regimes where all jobs are assumed to be present at the beginning and ignore congestion on wireless channels and edge servers.

Partial offloading. In partial offloading, a job represented as a DAG, see Figure 1, can be offloaded at several cut locations. Several research studies such as [2, 8, 13, 18, 19] have been conducted on partial offloading in the context of edge computing. In [19], the focus is on minimizing energy consumption while meeting latency constraints in a collaborative mobile device and edge server environment with stochastic channels. They propose a polynomial time algorithm for efficient job execution. Building on this work, [18] extends the approach to encompass general DAG frameworks beyond linear ones. In [2], the authors employ Reinforcement Learning (RL) to explore offloading multiple users' jobs to multiple servers. The users offload heterogeneous jobs over time-varying wireless channels. However, the RL-based policy requires re-learning for each new environment (number of users, job profiles, channel capacity, etc.). To overcome this limitation, [13] introduces the use of Meta Reinforcement Learning, enabling the RL agent to quickly adapt to new environments without re-learning. [8] investigates an online offloading framework similar to ours, where heterogeneous job types with deadline constraints arrive in the network according to a stochastic process and are executed dynamically over time. The authors propose a heuristic approach by relaxing the deadline constraint. The objective is to minimize the average makespan time.

1.2 Contributions and organization

The main contributions of this paper are summarized as follows:

- To the best of our knowledge, this is the first work to tackle the design of offloading policies for stochastic and heterogeneous job requests (distinct deadlines, cut locations, and computation-communication requirements per cut locations) under strict deadline constraints.
- We introduce and evaluate two "revenue" models to capture the possible benefits of offloading. The first model, termed net timely offloaded workload, aims to maximize the amount

of computational work offloaded while accounting for offloading overheads. The second model, referred to as power savings with wastage, aims to maximize power savings while considering power wastage associated with unsuccessful offloads. As a comparative baseline, we derive an upper bound on the revenue under *any* offloading and scheduling policy.
- We propose several classes of offloading policies. They differ in terms of (1) their requirements of knowledge of the system state, including adapting to the long-term offloading offered load and job types; (2) leveraging measurement-based abandonment policy, which reacts dynamically to congestion resulting from an excessive number of active users or poor wireless channel capacity relative to the job deadlines; and (3) their ability to adapt decisions regarding the choice of cut location and the fraction of jobs to admit based on changes in long term loads. We evaluate and compare these policies using representative job profiles from [6]. We do this for a range of offered job rates resulting in varying levels of congestion on the wireless network and edge server and study how close their performance is to the performance upper bound. Since some policies, such as PACCA + PA, our best policy, require prior knowledge of loads and thus possibly re-optimization when loads change, we also evaluate it under imperfect estimates of offered loads showing the approach is robust to such errors.

The paper is organized as follows: In Section 2, we introduce our basic system model. In Section 3, we develop an upper bound on the achievable revenue rate and explore different offloading management policies. We end the section with some simulation results. We discuss offloading management policies for heterogeneous compute job types in Section 4. Finally, Section 5 concludes the paper.

2 SYSTEM MODEL

We begin by introducing our system model for a set of users, generating homogeneous jobs (identical deadlines, cut locations, and computation-communication requirements per cut location) with limited local computation resources and a limited amount of shared wireless network and edge/cloud computation resources. We later consider heterogeneous jobs.

2.1 Model for load

We let $\mathcal{U}$ denote a set of users sharing a wireless access point – the set has cardinality $N = |\mathcal{U}|$. Each user u generates homogeneous jobs according to a stationary process. Users can have different channel qualities/classes. We let C denote the set of possible channel qualities with associated capacities. We let c_u denote the channel quality of user u and use λ_{u,c_u} to denote the arrival rate of jobs from user u. The total arrival rate of jobs from users with channel quality c is given by $\lambda_c = \sum_{u\in\mathcal{U}} \lambda_{u,c_u} \mathbb{1}(c_u = c)$. We denote the vector of total arrival rates for each channel quality as $\boldsymbol{\lambda} = (\lambda_c, c \in C)$. Finally, we define the total job arrival rate to the system as $\lambda = \sum_{c\in C} \lambda_c$.

2.2 Job model

The execution of a job may involve computation on a user's device, offloading of data to the edge/cloud server, processing on the edge/cloud server, and then transmitting the result back to the device. Initially, we focus on a single job type with a fixed time budget, τ, for job execution – that does not include the time required to transmit the result back to the user device[1]. We model the job as a DAG, where the possible cut locations are denoted by a set $\mathcal{S} = \{1, 2, ..., n\}$, with n representing the last cut location. For a cut location $k \in \mathcal{S}$, we let β_k denote the cumulative device processing measured in *floating point operations (FLOPs)* including overhead related to cutting itself, d_k denotes the offload data volume (in bits), and γ_k models the cumulative edge server processing (in FLOPs). Here, $k = 1$ corresponds to processing everything on the edge server, and $k = n$ corresponds to processing everything on the user's device.

2.3 Model for user's device, wireless channels, and edge server resources

A user's device has an effective processing speed denoted by δ, measured in *floating point operations per second (FLOPs/sec)*. A user with channel quality c has an uplink capacity to the base station of r_c Mbps. However, the transmission rate throughout the offloading process, as explained later, may be reduced by congestion, e.g., by competing job offloads from other users.

We consider multiple processors with multiple cores at the edge server. The total processing capacity, ω (FLOPs/sec), is modeled as the sum of all cores' processing rates across all processors, assuming jobs can be parallelized on all processors and cores. Thus, all active jobs get an "equal" time of edge server. Note that modern computing systems would allow the parallelization of jobs across only a limited number of cores of a given processor. Hence this is a simplification.

We let $\mathcal{S}_c \subseteq \mathcal{S}$ denote the set of cut locations for a user with channel quality c that guarantee the job will meet its delay deadline, τ when one optimistically assumes there is no competition for communication or computation resources in the system. Thus k is in $\mathcal{S}_c$ if

$$\underbrace{\frac{\beta_k}{\delta}}_{\text{local processing}} + \underbrace{\frac{d_k}{r_c}}_{\text{data offloading}} + \underbrace{\frac{\gamma_k}{\omega}}_{\text{edge processing}} \leq \tau \qquad (1)$$

where the left-hand side is the best possible end-to-end time to complete the offload when a job is cut at location k.

2.4 Sharing base station uplink resources

At any time, t, multiple users may be offloading data. $\mathcal{U}(t)$ represents the set of active users, and $N(t) = |\mathcal{U}(t)|$ is its cardinality. We shall assume that all users in $\mathcal{U}(t)$ share the BS's uplink resources in a Proportional Fair manner, with each ongoing offloading over channel c served at rate $r_c \frac{1}{N(t)}$.

2.5 Model for computation on the device and data offloading

When a job with local computational requirements is generated on a user's device, it undergoes computation based on a non-preemptive

[1]We neglect this because for a variety of applications such as video analytics, data volume associated with the result is much smaller than the uplink data transfer and the downlink capacity is typically much higher than the uplink capacity.

priority scheme that prioritizes jobs by their generation time. As a result, a job may be queued before its execution. After it leaves the queue, it is processed if there is enough time to execute it; otherwise, it is dropped.

Once the local part of the execution is complete, a job with data to offload is served on a first-come, first-served basis, so there may be additional waiting before offloading to the edge server begins.

2.6 Stationary offloading policies

We consider a set Π of stationary offloading policies. A policy may consist of *any* combination of job admission control/cutting /offloading methods, wireless channel scheduling, and edge server resource sharing. For a given offered load $\boldsymbol{\lambda}$ and policy $\pi \in \Pi$, we define $\mathbf{q}(\boldsymbol{\lambda}, \pi) = (q_{c,k}(\boldsymbol{\lambda}, \pi) : c \in C, k \in \mathcal{S})$, where $q_{c,k}(\boldsymbol{\lambda}, \pi)$ denotes the long-term *fraction* of jobs that belong to users with channel quality c, are cut at location k, and complete. Naturally, it must be the case that the sum of these fractions is less than or equal to one, meaning $\sum_{k\in\mathcal{S}} q_{c,k}(\boldsymbol{\lambda}, \pi) \leq 1$ for all $c \in C$ (in case not all jobs complete on time). We define $\mathcal{F}(\boldsymbol{\lambda}) = \{\mathbf{q}(\boldsymbol{\lambda}, \pi) \mid \pi \in \Pi\}$, as the set of fractions that are feasible under some policy. This set is convex since if $\pi_1, \pi_2 \in \Pi$, then by alternating between policies over long periods, one can achieve any convex combination of their associated performance.

2.7 Reward model and revenue metric

We introduce two reward models to guide the design and evaluation of offloading policies. We use α_k to represent the reward associated with the timely completion of a job cut at location k.

Net timely offloaded workload. The first reward model captures the total amount of work offloaded to the edge server for jobs that complete on time. It indirectly captures the freeing up of users' computation resources. The reward for offloading a job at cut location k is modeled as

$$\alpha_k = \gamma_k - g \cdot d_k. \tag{2}$$

Here γ_k denotes computation work offloaded to the edge and $g \cdot d_k$ the overhead of doing so, where d_k represents the volume of data offloaded, and g is a factor that "converts" bits to FLOPs.

The *net timely offloaded workload* measured in FLOPs/sec for a given offered load $\boldsymbol{\lambda}$ under policy π, is defined as

$$O(\mathbf{q}(\boldsymbol{\lambda}, \pi), \boldsymbol{\lambda}) = R_{\text{ow}}(\mathbf{q}(\boldsymbol{\lambda}, \pi), \boldsymbol{\lambda}) - L_{\text{ow}}(\boldsymbol{\lambda}, \pi) \tag{3}$$

where

$$R_{\text{ow}}(\mathbf{q}(\boldsymbol{\lambda}, \pi), \boldsymbol{\lambda}) = \sum_{c\in C} \lambda_c \sum_{k\in\mathcal{S}} \alpha_k q_{c,k}(\boldsymbol{\lambda}, \pi) \tag{4}$$

denotes the rate at which net work is offloaded, and $L_{\text{ow}}(\boldsymbol{\lambda}, \pi)$ is the rate of computational work on users' devices associated with jobs that do not complete on time and with jobs that a user attempts to offload but ends up completing locally[2].

Power savings with wastage. The second reward model quantifies the energy savings on a user device resulting from offloading. The energy expended by a device when offloading a job at cut k is modelled as $a \cdot \beta_k + b \cdot d_k$ joules, where a and b represent the energy expended per FLOP for local computation and per bit for data offloading, respectively. The energy savings from offloading at cut k vs. not offloading at all, i.e., cut at n (the last cut location), is given by

$$\alpha_k = a \cdot (\beta_n - \beta_k) + b \cdot (d_n - d_k) \tag{5}$$

in joules. This captures the energy saved from decreased local computation while considering the energy overhead of data offloading.

We define the net *power savings with wastage* measured in Watts for a given load $\boldsymbol{\lambda}$ under policy π, as

$$P(\mathbf{q}(\boldsymbol{\lambda}, \pi), \boldsymbol{\lambda}) = R_{\text{ps}}(\mathbf{q}(\boldsymbol{\lambda}, \pi), \boldsymbol{\lambda}) - L_{\text{ps}}(\boldsymbol{\lambda}, \pi) \tag{6}$$

where $R_{\text{ps}}(\cdot)$ is defined in the same way as $R_{\text{ow}}(\cdot)$ but with the energy savings reward α_k defined above for each timely job completion. $L_{\text{ps}}(\boldsymbol{\lambda}, \pi)$ represents the power expended at devices associated with jobs that miss their deadlines and with jobs that a user attempts to offload but ends up completing locally.

3 HOMOGENEOUS JOBS AND OFFLOADING POLICIES

In this section, we propose and evaluate several offloading policies for users with heterogeneous channel qualities but a homogeneous job type. In the next section, we extend the analysis to the case with heterogeneous job types.

3.1 Upper bound

We begin by developing a simple upper bound for the net timely offloaded workload or power savings with wastage achievable by any stationary offloading policy. Let $\mathbf{q} = (q_{c,k} : c \in C, k \in \mathcal{S})$, where $q_{c,k}$ denotes the fraction of jobs that belong to users with channel quality c, are cut at location k, and complete. We define the set of all possible vectors $\mathbf{q}$ as

$$\mathcal{Q} = \{\mathbf{q} \mid \mathbf{q} \geq \mathbf{0}, \sum_{k\in\mathcal{S}} q_{c,k} \leq 1 \,\forall\, c \in C \text{ and } q_{c,k} = 0\, c \in C,\ k \in \mathcal{S}\backslash\mathcal{S}_c\} \tag{7}$$

where in some settings, a fraction of jobs may not complete, hence they need not sum up to 1, and fractions for infeasible cut locations are zero. Given $\mathbf{q}$, we define the channel and edge server utilization as

$$\rho_{\text{ch}}(\mathbf{q}) = \sum_{c\in C} \lambda_c \sum_{k\in\mathcal{S}} q_{c,k} \frac{d_k}{r_c} \mathbb{1}(k \in \mathcal{S}_c), \tag{8}$$

$$\rho_{\text{ed}}(\mathbf{q}) = \sum_{c\in C} \lambda_c \sum_{k\in\mathcal{S}} q_{c,k} \frac{\gamma_k}{\omega} \mathbb{1}(k \in \mathcal{S}_c), \tag{9}$$

respectively. Recall that we defined $\mathcal{F}(\boldsymbol{\lambda})$ to be the set of *feasible* long-term fractions of successful job completions under a set of stationary offloading policies when the system load is $\boldsymbol{\lambda}$. We define the set of all *possible* successful long term fractions

$$\overline{\mathcal{F}}(\boldsymbol{\lambda}) = \{\mathbf{q} \mid \mathbf{q} \in \mathcal{Q},\ \rho_{\text{ch}}(\mathbf{q}) \leq 1 \text{ and } \rho_{\text{ed}}(\mathbf{q}) \leq 1\} \tag{10}$$

as a natural outer bound for $\mathcal{F}(\boldsymbol{\lambda})$ which leads to the following simple performance bounds.

Theorem 1. Given an offered load $\boldsymbol{\lambda}$ we have that $\mathcal{F}(\boldsymbol{\lambda}) \subseteq \overline{\mathcal{F}}(\boldsymbol{\lambda})$. Then the maximum net timely offloaded workload achievable by any stationary offloading policy is defined as

$$O^*(\boldsymbol{\lambda}) := \max_{\mathbf{q}\in\mathcal{F}(\boldsymbol{\lambda})} O(\mathbf{q}, \boldsymbol{\lambda}) \leq \max_{\mathbf{q}\in\overline{\mathcal{F}}(\boldsymbol{\lambda})} R_{\text{ow}}(\mathbf{q}, \boldsymbol{\lambda}) \tag{11}$$

[2]Note this occurs when a user attempts to offload a job but due to congestion on wireless channels and/or edge server abandons and reverts to local execution.

Similarly, the maximum power savings with wastage achievable by any stationary offloading policy is defined as

$$P^*(\boldsymbol{\lambda}) := \max_{\mathbf{q}\in\mathcal{F}(\boldsymbol{\lambda})} P(\mathbf{q},\boldsymbol{\lambda}) \leq \max_{\mathbf{q}\in\overline{\mathcal{F}}(\boldsymbol{\lambda})} R_{\text{ps}}(\mathbf{q},\boldsymbol{\lambda}) \tag{12}$$

Proof. We first argue that $\mathcal{F}(\boldsymbol{\lambda}) \subseteq \overline{\mathcal{F}}(\boldsymbol{\lambda})$. Indeed suppose $\mathbf{q} \in \mathcal{F}(\boldsymbol{\lambda})$. Recall that $q_{c,k}$ represents the fraction of incoming jobs that belong to users with channel quality c, are cut at location k, and complete. Since each job is cut at only one location, these fractions always sum to less than or equal to 1 over all cut locations, thus q is clearly in $\mathcal{Q}$ – but suppose the load on the channel or edge server is greater than 1 – given these fraction of jobs, i.e., $\rho_{\text{ch}}(\mathbf{q}) > 1$ or $\rho_{\text{ed}}(\mathbf{q}) > 1$. If this is true, then not all jobs will complete on time, contradicting our earlier statement. Thus, the channel and edge server load must be less than or equal to 1, i.e., $\rho_{\text{ch}}(\mathbf{q}) \leq 1$ and $\rho_{\text{ed}}(\mathbf{q}) \leq 1$ if $\mathbf{q} \in \mathcal{F}(\boldsymbol{\lambda})$, which implies $\mathbf{q} \in \overline{\mathcal{F}}(\boldsymbol{\lambda})$. Thus $\mathcal{F}(\boldsymbol{\lambda}) \subseteq \overline{\mathcal{F}}(\boldsymbol{\lambda})$. This then implies $\max_{\mathbf{q}\in\mathcal{F}(\boldsymbol{\lambda})} R_{\text{ow}}(\mathbf{q},\boldsymbol{\lambda}) \leq \max_{\mathbf{q}\in\overline{\mathcal{F}}(\boldsymbol{\lambda})} R_{\text{ow}}(\mathbf{q},\boldsymbol{\lambda})$ which results in the Equation 11, since $L_{\text{ow}}(\boldsymbol{\lambda},\pi) \geq 0$. Similarly under the energy savings reward model and recognizing that $L_{\text{ps}}(\boldsymbol{\lambda},\pi) \geq 0$, we have Equation 12.

□

Remark 1. We let $\mathbf{q}^*_{\text{ow}}(\boldsymbol{\lambda}) = \text{argmax}_{\mathbf{q}\in\overline{\mathcal{F}}(\boldsymbol{\lambda})} R_{\text{ow}}(\mathbf{q},\boldsymbol{\lambda})$ and $\mathbf{q}^*_{\text{ps}}(\boldsymbol{\lambda}) = \text{argmax}_{\mathbf{q}\in\overline{\mathcal{F}}(\boldsymbol{\lambda})} R_{\text{ps}}(\mathbf{q},\boldsymbol{\lambda})$ denote the vector that maximizes the bounds for the two reward models. These indicate the fraction of load to admit across sets of channel qualities and cut locations to maximize revenue in the absence of resource contention during job offloading.

3.2 Offloading policies

Naive Greedy (NG). The NG offloading policy optimistically assigns the cut location, k_u, that yields the highest reward among all feasible cut locations given deadline τ, to all jobs generated by user u in $\mathcal{U}$ with channel quality c_u as follows:

$$k_u = \underset{k\in\mathcal{S}_{c_u}}{\text{argmax}}\, \alpha_k \tag{13}$$

where α_k either reflects net offloaded workload or energy savings, see Section 2.7. The policy then greedily tries to offload data and process on the edge server until success or time budget, τ, expires.

Predictive Abandonment (PA). PA is a real-time user-based offloading policy that adapts to channel and edge/cloud server congestion. Similar to the NG policy, a user u implementing PA selects the cut location k_u with the highest reward, see Equation 13. However, unlike NG, during the offloading process, a user implementing PA estimates the feasibility of meeting a job's completion deadline given current channel and/or edge server congestion. If the deadline is unlikely to be met, PA saves resources by abandoning the job's offload, thus increasing the likelihood of completing other jobs on time. Furthermore, a job whose offload is abandoned can still attempt to complete its residual processing on the user device if time permits. Thus, PA can be viewed as performing a type of state-dependent self-admission control or abandonment policy.

Under PA, a user u determines whether its ith job is likely to complete on time by considering/predicting the following factors: queuing time, local processing time, data offloading time, and edge processing time. The queuing time for a job i at time t, denoted $W_{u,i}(t)$, corresponds to the time the job has been waiting in the user's queue. The local processing time is calculated as the number of operations before the cut location k_u divided by the device's execution speed, i.e., $\frac{\beta_{k_u}}{\delta}$ secs. Since a user's compute speed is fixed, this value is the same for all t. Also, recall that all jobs from a user u are cut at the same location. The data offloading time at time t is calculated as the sum of time already spent offloading the job (if any) and the time required to offload any remaining data. The latter can be estimated by dividing the data yet to be offloaded at time t, $s_{u,i}(t)$, by an estimate for the future average transmission rate $h_{u,i}(t)$. Suppose user u initiates data offloading of job i at time $t_{u,i}$ over channel quality c_u. At any instance t' (for t' greater than or equal to $t_{u,i}$), the transmission rate under our model is given by $\frac{r_{c_u}}{N(t')}$, where recall r_{c_u} is the uplink channel capacity and $N(t')$ is the number of active users. We estimate the future average transmission rate based on the average throughput experienced by the user u since it began offloading job i, i.e.,

$$h_{u,i}(t) = \frac{1}{e_{u,i}(t)} \sum_{t'=t_{u,i}}^{e_{u,i}(t)+t_{u,i}} \frac{r_{c_u}}{N(t')} \tag{14}$$

where $e_{u,i}(t)$ is the time elapsed since offloading began. The last factor in the equation for job latency is the edge processing time. We assume that the edge server provides users with or users themselves maintain estimates of the job's processing time per cut location, $\overline{m}_{k_u}$. These estimates are periodically updated.

Putting together the elapsed time and estimated future transmission/processing latencies, the estimated total latency for user u's job i is given by

$$\underbrace{W_{u,i}(t)}_{\text{queuing}} + \underbrace{\frac{\beta_{k_u}}{\delta}}_{\text{local processing}} + \underbrace{e_{u,i}(t) + \frac{s_{u,i}(t)}{h_{u,i}(t)}}_{\text{data offloading}} + \underbrace{\overline{m}_{k_u}}_{\text{edge processing}} . \tag{15}$$

Under PA, a user may abandon offloading if it determines that its estimated total latency for job i is greater than its time budget, τ. Once abandoned, any remaining computation, γ_{k_u}, for job i can be completed on the user's device if there is enough time.

This prediction method is crude but roughly captures the impact of the user's channel capacity r_{c_u}, channel's uplink congestion $N(t)$, and congestion at the edge server using the actual estimate of processing latencies, $\overline{m}_{k_u}$. Note that these estimates will reflect changes resulting from the PA policy itself since PA impacts the load on the channel and edge server. For more accurate predictions, one can consider additional factors such as number of active users during each user's offload and/or remaining service requirements of currently offloading jobs. See, [14] for such a discussion in a processor-sharing scenario without abandonment.

PA effectively performs a sort of "real-time" admission control by abandoning jobs unlikely to meet their deadlines due to channel and/or edge server congestion. In congested scenarios, PA "prefers" users with better channels, since they are more likely to complete their offloads on time. Finally, note PA, like NG, is decentralized and does not require knowledge of the overall offered job rate, $\boldsymbol{\lambda}$. Next, we explore a policy that considers the overall rate of offered jobs for coordination.

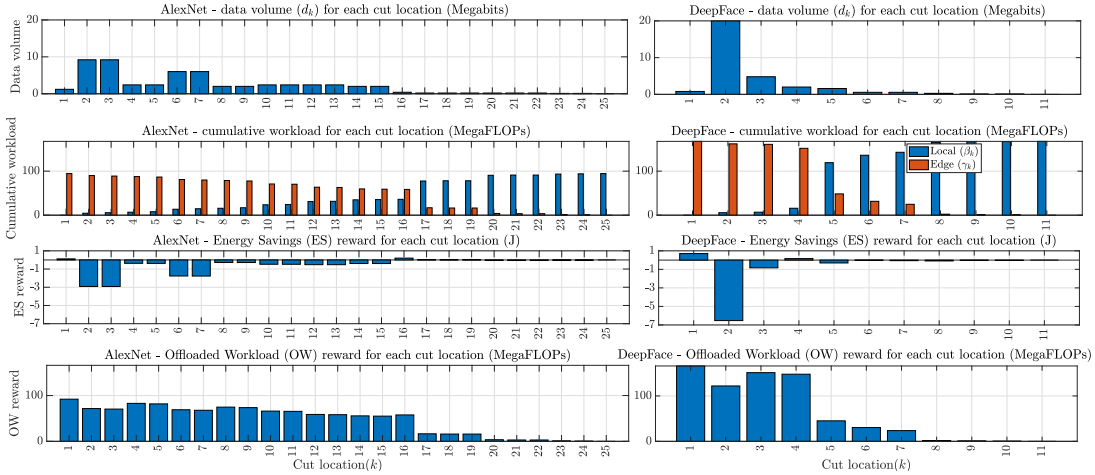


Figure 2: Simulation parameters associated with AlexNet and DeepFace job types.

Probabilistic Admission Control and Cut Assignment (PACCA). Under PACCA, we pre-determine $p_{c,k}$, the fraction of jobs that belong to users with channel quality c that should be offloaded at each cut location k to maximize revenue given the rate of incoming jobs. We let $\mathbf{p} = (p_{c,k} : c \in C, k \in \mathcal{S})$, and require $\sum_{k\in\mathcal{S}} p_{c,k} \leq 1$ for all $c \in C$. We denote the associated policy as PACCA ($\mathbf{p}$). If a job is not admitted for offloading under this policy, it will attempt to execute locally. Determining $\mathbf{p}$ is complex, as it involves multiple factors, such as contending offloads from users with different channel qualities, channel and server congestion, delay constraints, and revenue rate optimization. We propose to choose $\mathbf{p}$ based on the upper bounds from Theorem 1, either $\mathbf{q}^*_{\text{ow}}(\boldsymbol{\lambda})$ to maximize net timely offloaded workload or $\mathbf{q}^*_{\text{ps}}(\boldsymbol{\lambda})$ to maximize power savings with wastage. In later sections, for brevity we use $\mathbf{q}^*$ to refer to either $\mathbf{q}^*_{\text{ow}}(\boldsymbol{\lambda})$ or $\mathbf{q}^*_{\text{ps}}(\boldsymbol{\lambda})$. Given that there is no resource contention underlying Theorem 1 (see Remark 1), these probabilities reflect optimistic admission control and cut assignment for a given $\boldsymbol{\lambda}$. Nevertheless, they still reflect reasonable overall system tradeoffs.

Once a job is admitted for offloading, the user can either attempt to offload the job greedily at the assigned cut location or perform PA, only proceeding with the offload if it determines the job's deadline can be met. We refer to the former policy as PACCA ($\mathbf{q}^*$) + Greedy and the latter as PACCA ($\mathbf{q}^*$) + PA.

3.3 Simulation results

In this subsection, we evaluate the performance of our proposed offloading policies via discrete-time simulations in terms of: (i) net timely offloaded workload; (ii) power savings with wastage (explained in Section 2.7). We also plot the fraction of jobs completed. The simulations are conducted in MATLAB R2023a.

Settings. We consider a system with $N = 20$ users[3], where each user generates an equal rate of homogeneous jobs per second, according to a Poisson distribution[4], with intensity $\lambda/20$. This results in an aggregate job generation/arrival rate of λ per second. The system includes two channel qualities, half of the users (i.e., 10) have one channel quality, half the other[5]. A user's channel quality/capacity does not change, but the transmission rate at any instance depends on both the channel capacity and the number of competing users because of the Proportional Fair sharing of uplink resources. Table 1 summarizes the simulation parameters. We present results for the homogeneous scenario based on the job characteristics of AlexNet, a state-of-the-art Convolutional Neural Network for image classification. In the next section, we will also use DeepFace, which is used for face recognition. Figure 2 displays the job characteristics of AlexNet on the left and DeepFace on the right. The four bar graphs (from top to bottom) show the volume of data that gets offloaded, the amount of local vs. edge processing, the energy saved, and the amount of work that gets offloaded per cut location. The worst delay a job may experience is calculated as $\tau_{\max} = \max_k \left(\frac{\beta_k}{\delta} + \frac{d_k}{\min_{c\in C} r_c} + \frac{\gamma_k}{\omega} \right)$. However, this may not be the absolute worst case as it only considers the worst channel quality and not congestion. We evaluated our policies under both strict $\tau = 0.4\tau_{\max}$ and relaxed $\tau = 0.8\tau_{\max}$ delay deadlines. Results are averaged over 20 Monte Carlo simulations, each performed over 5e5 time slots. Here a time slot is 100 μs long.

Table 1: Simulation parameters

Parameter (units)	Value	Parameter (units)	Value
r_c (Mbps)	$(20, 40)$	g (FLOP/bit)	2
δ (FLOPs/sec)	$1125 * 10^6$	a (J/FLOP)	$6 * 10^{-9}$
ω (FLOPs/sec)	$11360 * 10^6$	b (J/bit)	$4 * 10^{-7}$

Results discussion. Our first set of results, presented in Figures 3a and 3b, include the net timely offloaded workload and fraction of completed jobs for our policies per total offered job rate, respectively, under a strict job deadline. In Figure 3a, we show the net computational workload offloaded, which depends on the fraction

[3]We chose this to represent typical traffic at a 5G BS/AP in a dense urban environment deployment scenario, we can increase/decrease the number as needed.

[4]We also ran the simulations for other arrival processes but due to space limitations only include the results for the Poisson arrival process.

[5]We consider this for simplicity; our work applies to any number of channel qualities.

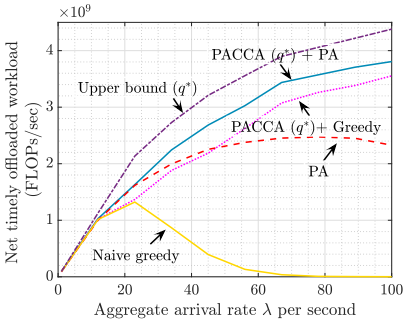
of completed jobs and the reward per completed job. Therefore, a policy can perform equally well in two cases: (i) completing numerous jobs with a low reward or (ii) completing fewer jobs with a high reward. Thus, we also present the fraction of completed jobs in Figure 3b for the same simulation setting for all policies.

We observe that as the offered job rate λ increases, the NG policy experiences throughput collapse, see Figure 3a. By contrast, policies like PA, which implement congestion-dependent offload abandonment, and PACCA, which determines admission control and cut assignments based on prior knowledge of incoming jobs per user and channel quality, perform well under heavy job loads. However, PA does not perform as well as PACCA, highlighting the benefit reserving channel and edge resources for jobs with good channels and/or high reward cut locations. The benefit of combining these policies, PACCA + PA becomes more evident at high-load regimes where admission control and congestion management is crucial.

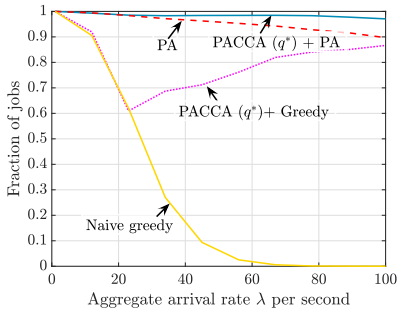
In Figure 3b, we show the fraction of completed jobs under different offloading policies as job arrival rate λ increases. With PA, more than 90% of jobs are completed in the load regimes considered. Indeed, for all the results reported hereafter, we only considered load regimes where PA has a high completion rate (at least 90%) and where the channel is the bottleneck. Interestingly, the fraction of jobs that complete under PACCA + Greedy is non-monotonic with increasing load. This is because initially (from 0 to 23 jobs/sec), PACCA selects the highest reward cut location for every job. However, since channel capacity is fixed, an increase in the number of jobs means a decrease in completions. Hence, the downward curve. Then, at 23 jobs/sec, PACCA determines it will earn more revenue by adjusting the distribution of cut locations, so that jobs have less data to offload. This results in less reward per job, but more completed jobs. PACCA makes this adjustment every time the rate of incoming jobs increases beyond 23 jobs/sec, thus the upward curve. We observe similar non-monotonic behavior for PACCA + PA though it is barely perceptible in the figure.

In Figure 3a, we saw the advantage of combining PACCA with PA under a strict delay deadline. We observe similar benefits under a relaxed delay deadline, see Figure 4; however, the performance gap between PACCA + PA and PACCA + Greedy decreases. This is because, with a relaxed delay deadline, a user has a higher chance of completing an offload, even with network congestion. Thus, network congestion is detrimental only under strict delay deadlines necessitating a congestion-aware policy like PA. We see similar trends with power savings with wastage in Figures 5a and 5b. Additionally, as we relax the completion deadline, our best-performing policy (PACCA + PA) approaches the upper bound for both performance metrics.

Robustness study. We demonstrate the robustness of PACCA + PA to imperfect knowledge of offered jobs per sec in Figure 6, for net timely offloaded workload under a strict delay deadline[6]. In these simulations, we added errors to the estimates of job arrival rates per user, which affects the aggregate arrival rate per channel quality. We optimize PACCA + PA for the corresponding load and compare three scenarios: (i) PACCA + PA (exact), where we provide PACCA with the exact load, i.e., λ ; (ii) PACCA + PA



(a) Net timely offloaded workload (FLOPs/sec).



(b) Fraction of completed jobs.

Figure 3: Comparing the net timely offloaded work and fraction of jobs that complete for different policies when the job's delay deadline is strict, i.e., $\tau = 0.4\tau_{\max}$.

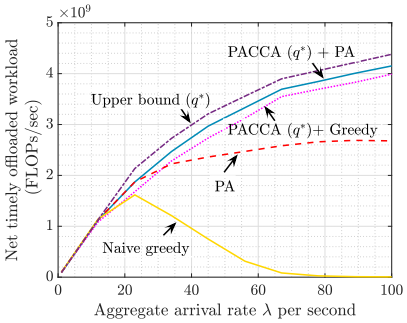
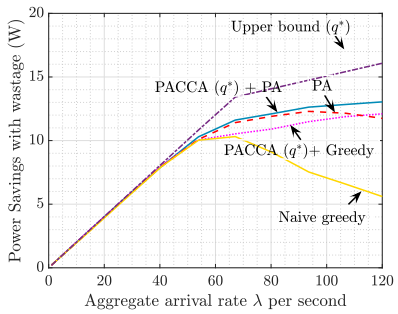


Figure 4: Comparing the net timely offloaded workload for different policies when the job's delay deadline is relaxed, i.e., $\tau = 0.8\tau_{\max}$.

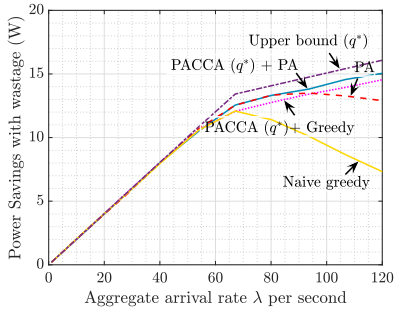
(overestimate), where we provide PACCA with an overestimate of load, i.e., $\lambda \cdot (1 + x\%)$, leading to less offloading compared to (i); and (iii) PACCA + PA (underestimate), where we provide PACCA with an underestimate of load, i.e., $\lambda \cdot (1 - x\%)$, resulting in more offloading compared to (i). We show PA as a baseline. The results show that for an estimation error of 25% (i.e., $x = 25$)[7], both PACCA + PA (overestimate) and PACCA + PA (underestimate) are within 10% of PACCA + PA (exact). Additionally, we observe that PACCA + PA (underestimate) performs at least as well as PA, as huge underestimation errors result in admitting every job at the highest reward cut location, effectively imitating PA.

[6]Due to space constraints, we exclude similar results for the metric power savings with wastage.

[7]We also performed simulations for $x = 50\%$ but exclude it due to space limitations.



(a) Strict delay deadline, $\tau = 0.4\tau_{\max}$.



(b) Relaxed delay deadline, $\tau = 0.8\tau_{\max}$.

Figure 5: Comparing the power savings with wastage for different policies when the job's delay deadline is strict vs. relaxed.

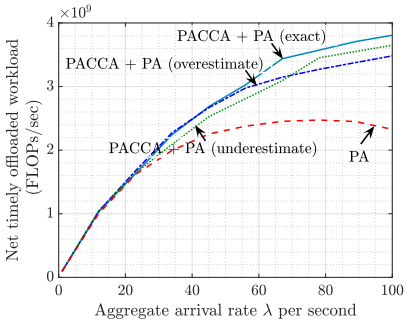


Figure 6: Evaluating robustness of PACCA + PA for a 25% deviation from exact load knowledge when the job's delay deadline is strict, i.e., $\tau = 0.4\tau_{\max}$.

4 HETEROGENEOUS JOBS

In this section, we investigate the performance of our policies in networks with heterogeneous jobs, where jobs no longer have identical deadlines, cut locations, and requirements per cut location (such as computation and data to offload).

We have a set of users generating different types of jobs, denoted by $\mathcal{J} = \{1, 2, ..., J\}$. For each job type j in $\mathcal{J}$, we let $\mathcal{U}^j$ be the set of users generating such jobs, and $\lambda^{j,u}$ denotes the arrival rate of job type j generated by user u in $\mathcal{U}^j$. The total arrival rate of job type j is denoted as $\lambda^j = \sum_{u\in\mathcal{U}^j} \lambda^{j,u}$. We define the arrival rate of job type j over channel quality c in C from all users as $\lambda_c^j = \sum_{u\in\mathcal{U}^j} \lambda^{j,u}\mathbb{1}(c_u = c)$, and let $\boldsymbol{\lambda}^j = (\lambda_c^j, c \in C)$. The arrival rate of each job type is captured by $\boldsymbol{\Lambda} = (\boldsymbol{\lambda}^1, ..., \boldsymbol{\lambda}^J)$. We use τ^j to represent the delay constraint, and $\mathcal{S}^j$ to capture the set of possible cut locations for type j jobs. As before, $\mathcal{S}_c^j$ is a subset of $\mathcal{S}^j$ that only includes cut locations that are feasible for a given user's channel quality c under time budget τ^j, i.e., a location k is in $\mathcal{S}_c^j$ if

$$\underbrace{\frac{\beta_k^j}{\delta}}_{\text{local processing}} + \underbrace{\frac{d_k^j}{r_c}}_{\text{data offloading}} + \underbrace{\frac{\gamma_k^j}{\omega}}_{\text{edge processing}} \leq \tau^j \qquad (16)$$

where β_k^j represents the computational burden (in FLOPs) on the user (including overhead for cutting), d_k^j denotes the volume (in bits) of data transfer to the edge server, and γ_k^j captures the computational burden (in FLOPs) on the edge server for a type j job cut at location k.

For an offered load, $\boldsymbol{\Lambda}$, under offloading policy $\pi \in \Pi$, we define $\mathbf{Q}(\boldsymbol{\Lambda}, \pi) = (\mathbf{q^1}(\boldsymbol{\Lambda}, \pi), ..., \mathbf{q^J}(\boldsymbol{\Lambda}, \pi))$, where $\mathbf{q^j}(\boldsymbol{\Lambda}, \pi) = (q_{c,k}^j(\boldsymbol{\Lambda}, \pi) : c \in C, k \in \mathcal{S}^j)$, and $q_{c,k}^j(\boldsymbol{\Lambda}, \pi)$ is the long-term fraction of type j jobs that belong to users with channel quality c, are cut at location k, and complete. These fractions must sum to less than or equal to 1 (some jobs may not complete), i.e., $\sum_{k\in\mathcal{S}^j} q_{c,k}^j(\boldsymbol{\Lambda}, \pi) \leq 1$ for all $j \in \mathcal{J}$, $c \in C$. We define $\mathcal{T}(\boldsymbol{\Lambda}) = \mathbf{Q}(\boldsymbol{\Lambda}, \pi)$ for $\pi \in \Pi$ as the set of feasible long-term fractions, which is convex through time sharing argument presented in the homogeneous case.

Just as in the homogeneous scenario, we have two revenue metrics: weighted net timely offloaded workload revenue and weighted power savings with wastage revenue. We let α_k^j denote the reward generated from a successful job completion when a type j job on a user is cut and offloaded at cut location k.

Weighted net timely offloaded workload revenue. We define it as a weighted sum of net timely offloaded workload for each job type, where w^j is the weight [8] for job type j and $\sum_{j\in\mathcal{J}} w^j = 1$. For a given offered load $\boldsymbol{\Lambda}$ and offloading policy π, the revenue is defined as:

$$O(\mathbf{Q}(\boldsymbol{\Lambda}, \pi), \boldsymbol{\Lambda}) = R_{\text{ow}}(\mathbf{Q}(\boldsymbol{\Lambda}, \pi), \boldsymbol{\Lambda}) - L_{\text{ow}}(\boldsymbol{\Lambda}, \pi). \qquad (17)$$

Here

$$R_{\text{ow}}(\mathbf{Q}(\boldsymbol{\Lambda}, \pi), \boldsymbol{\Lambda}) = \sum_{j\in\mathcal{J}} w^j \sum_{c\in C} \lambda_c^j \sum_{k\in\mathcal{S}^j} \alpha_k^j q_{c,k}^j(\boldsymbol{\Lambda}, \pi) \qquad (18)$$

denotes the aggregate reward, and $L_{\text{ow}}(\boldsymbol{\Lambda}, \pi)$ is the rate of computational work on users' devices associated with jobs that do not complete on time and with jobs that a user attempts to offload but ends up completing locally.

Weighted power savings with wastage revenue. We define it as a weighted sum of power savings with wastage per job type j. Given an offered load $\boldsymbol{\Lambda}$ and an offloading policy π, we calculate it as follows:

$$P(\mathbf{Q}(\boldsymbol{\Lambda}, \pi), \boldsymbol{\Lambda}) = R_{\text{ps}}(\mathbf{Q}(\boldsymbol{\Lambda}, \pi), \boldsymbol{\Lambda}) - L_{\text{ps}}(\boldsymbol{\Lambda}, \pi) \qquad (19)$$

where $R_{\text{ps}}(\cdot) = R_{\text{ow}}(\cdot)$ except that reward, α_k^j, is based on energy saved per job completion. $L(\boldsymbol{\Lambda}, \pi)$ represents the power expended

[8] The weights w^j are used to prioritize one job type over another in the admission control policy. However, we still assume the underlying wireless scheduler is round-robin and unaware of job types.

at devices associated with jobs that miss their deadlines and jobs that a user attempts to offload but ends up executing locally.

4.1 Upper bound

Let $\mathbf{Q} = (\mathbf{q^1}, ..., \mathbf{q^J})$ be a vector of vectors, where $\mathbf{q^j} = (q^j_{c,k} : c \in C, k \in \mathcal{S}^j)$, and $q^j_{c,k}$ is the fraction of type j jobs that belong to users with channel quality c, are cut at location k, and complete. We define

$$\Sigma = \{\mathbf{Q} \mid \mathbf{q}^j \geq \mathbf{0}, \sum_{k \in \mathcal{S}^j} q^j_{c,k} \leq 1 \; \forall \; j \in \mathcal{J}, \; c \in C, \\ \text{and } q^j_{c,k} = 0 \; \forall \; j \in \mathcal{J}, \; c \in C, \; k \in \mathcal{S}^j \setminus \mathcal{S}^j_c\} \tag{20}$$

as the set of such possible vector of vectors. We then define the channel and edge server utilization per job type j based on $\mathbf{Q}$, which is long term fraction of completed jobs, as

$$\rho^j_{\text{ch}}(\mathbf{Q}) = \sum_{c \in C} \lambda^j_c \sum_{k \in \mathcal{S}^j} q^j_{c,k} \frac{d^j_k}{r_c} \mathbb{1}(k \in \mathcal{S}^j_c), \tag{21}$$

$$\rho^j_{\text{ed}}(\mathbf{Q}) = \sum_{c \in C} \lambda^j_c \sum_{k \in \mathcal{S}^j} q^j_{c,k} \frac{\gamma^j_k}{\omega} \mathbb{1}(k \in \mathcal{S}^j_c), \tag{22}$$

and let $\rho_{\text{ch}}(\mathbf{Q}) = \sum_{j \in \mathcal{J}} \rho^j_{\text{ch}}(\mathbf{Q})$ and $\rho_{\text{ed}}(\mathbf{Q}) = \sum_{j \in \mathcal{J}} \rho^j_{\text{ed}}(\mathbf{Q})$ denote the total channel and the total edge server utilization, respectively.

Recall that we defined $\mathcal{T}(\mathbf{\Lambda})$ to be the set of *feasible* long term fractions of successful job completion by stationary offloading policies when the system load is $\mathbf{\Lambda}$. Here we define

$$\overline{\mathcal{T}}(\mathbf{\Lambda}) = \{\mathbf{Q} \mid \mathbf{Q} \in \Sigma, \; \rho_{\text{ch}}(\mathbf{Q}) \leq 1 \text{ and } \rho_{\text{ed}}(\mathbf{Q}) \leq 1\} \tag{23}$$

as a natural outer bound.

Theorem 2. Given an offered load $\mathbf{\Lambda}$ we have that $\mathcal{T}(\mathbf{\Lambda}) \subseteq \overline{\mathcal{T}}(\mathbf{\Lambda})$. Then the maximum weighted net timely offloaded workload revenue achievable by any stationary offloading policy is defined as

$$O^*(\mathbf{\Lambda}) := \max_{\mathbf{Q} \in \mathcal{T}(\mathbf{\Lambda})} O(\mathbf{Q}, \mathbf{\Lambda}) \leq \max_{\mathbf{Q} \in \overline{\mathcal{T}}(\mathbf{\Lambda})} R_{\text{ow}}(\mathbf{Q}, \mathbf{\Lambda}) \tag{24}$$

Similarly, the maximum weighted power savings with wastage revenue achievable by any stationary offloading policy is defined as

$$P^*(\mathbf{\Lambda}) := \max_{\mathbf{Q} \in \mathcal{T}(\mathbf{\Lambda})} P(\mathbf{Q}, \mathbf{\Lambda}) \leq \max_{\mathbf{Q} \in \overline{\mathcal{T}}(\mathbf{\Lambda})} R_{\text{ps}}(\mathbf{Q}, \mathbf{\Lambda}) \tag{25}$$

Proof. Similar to proof of Theorem 1. □

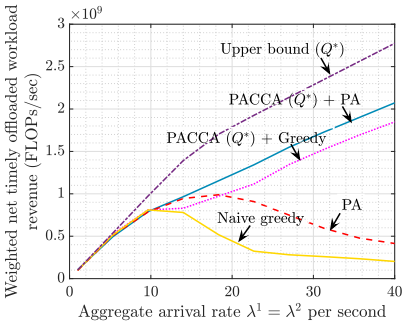
Remark 2. We let $\mathbf{Q}^*_{\text{ow}}(\mathbf{\Lambda}) = \text{argmax}_{\mathbf{Q} \in \overline{\mathcal{T}}(\mathbf{\Lambda})} R_{\text{ow}}(\mathbf{Q}, \mathbf{\Lambda})$ and $\mathbf{Q}^*_{\text{ps}}(\mathbf{\Lambda}) = \text{argmax}_{\mathbf{Q} \in \overline{\mathcal{T}}(\mathbf{\Lambda})} R_{\text{ps}}(\mathbf{Q}, \mathbf{\Lambda})$ denote the maximizers associated with the bounds for the two reward models. Once again, for brevity we will use $\mathbf{Q}^*$.

4.2 Simulation results

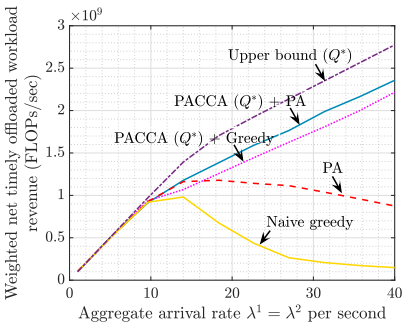
In this subsection, we compare the performance of various offloading management policies when dealing with heterogeneous jobs. For PACCA, we determine the admission control probabilities per combination of job types, user channel qualities, and cut location based on the fractions that maximize system revenue, i.e., $\mathbf{Q}^*$. We then again schedule the offloading of admitted jobs using either Greedy or PA policies, i.e., PACCA + Greedy or PACCA + PA. Due to space constraints, we omitted our study of the robustness of PACCA + PA for the heterogeneous case.

Setting. The simulation involves 20 users generating two job types, AlexNet and DeepFace. Half of the users (i.e., 10) generate job Type 1, while the other half generates job Type 2. Each user in either category generates jobs at an equal rate per second based on a Poisson distribution with intensity $\lambda^j/10$, where λ^j is the aggregate arrival rate for job type j. We have set equal aggregate arrival rate for the two job types. Users for each job type are divided equally into two channel quality groups, with half (i.e., 5) offloading over channel Quality 1 and the other half over channel Quality 2. For more information on the simulation parameters, refer to Table 1 and Figure 2.

Results discussion. Figures 7 and 8 illustrate the weighted net timely offloaded workload and weighted power savings with wastage revenue achieved by various offloading policies, respectively. As observed in the case of homogeneous jobs, the PACCA + PA policy outperforms other policies. However, the relative performance of PA policy has declined compared to the homogeneous case since it only considers the residual data and channel capacity, ignoring a job's weight relative to others. In contrast, PACCA coordinates job admission control and cut assignment based on how much relative revenue each job type and cut will generate.

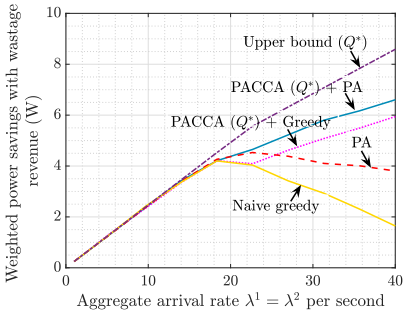


(a) Strict delay deadline, $(\tau^1, \tau^2) = (0.4\tau^1_{\max}, 0.8\tau^2_{\max})$.

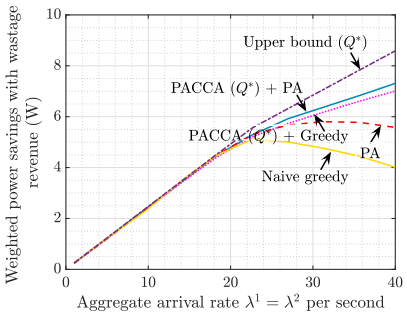


(b) Relaxed delay deadline, $(\tau^1, \tau^2) = (0.8\tau^1_{\max}, 0.8\tau^2_{\max})$.

Figure 7: Comparing the weighted net timely offloaded workload revenue for different policies with $(w^1, w^2) = (0.97, 0.03)$.



(a) Strict delay deadline, $(\tau^1, \tau^2) = (0.4\tau^1_{\max}, 0.8\tau^2_{\max})$.



(b) Relaxed delay deadline, $(\tau^1, \tau^2) = (0.8\tau^1_{\max}, 0.8\tau^2_{\max})$.

Figure 8: Comparing the weighted power savings with wastage revenue for different policies with $(w^1, w^2) = (0.97, 0.03)$.

5 CONCLUSION

Managing heterogeneous compute job offloading in the MEC fabric subject to delay constraints presents significant challenges that require careful management of user device, channel, and edge server resources while considering different job characteristics and system loads. To address this, we have detailed a comprehensive framework, which relies on job profiling, using probabilistic admission control and cut assignment, coupled with a predictive abandonment policy that abandons offloads unlikely to meet their deadline (this not only frees up resources for jobs with more promise but also avoids throughput collapse). Our proposed approach, PACCA + PA, is expected to perform robustly and effectively but requires prior knowledge of offered job rates across job types and channel qualities. If this is not known, signal processing techniques such as window averaging can be employed to learn the offered job rate over time.

ACKNOWLEDGEMENTS

This work was supported by Intel, an affiliate of the 6G@UT center within the Wireless Networking and Communications Group at The University of Texas at Austin.

REFERENCES

[1] Pavlos Athanasios Apostolopoulos, Eirini Eleni Tsiropoulou, and Symeon Papavassiliou. 2020. Risk-Aware Data Offloading in Multi-Server Multi-Access Edge Computing Environment. *IEEE/ACM Transactions on Networking* 28, 3 (2020), 1405–1418. https://doi.org/10.1109/TNET.2020.2983119

[2] Jiawen Chen, Yajun Yang, Chenyang Wang, et al. 2021. Multi-Task Offloading Strategy Optimization based on Directed Acyclic Graphs for Edge Computing. *IEEE Internet of Things Journal* (2021), 1–1. https://doi.org/10.1109/JIOT.2021.3110412

[3] Thinh Quang Dinh, Jianhua Tang, Quang Duy La, et al. 2017. Offloading in Mobile Edge Computing: Task Allocation and Computational Frequency Scaling. *IEEE Transactions on Communications* 65, 8 (2017), 3571–3584. https://doi.org/10.1109/TCOMM.2017.2699660

[4] Jianbo Du, Liqiang Zhao, Jie Feng, and Xiaoli Chu. 2018. Computation Offloading and Resource Allocation in Mixed Fog/Cloud Computing Systems With Min-Max Fairness Guarantee. *IEEE Transactions on Communications* 66, 4 (2018), 1594–1608. https://doi.org/10.1109/TCOMM.2017.2787700

[5] Hyuk-Jin Jeong, InChang Jeong, Hyeon-Jae Lee, and Soo-Mook Moon. 2018. Computation Offloading for Machine Learning Web Apps in the Edge Server Environment. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*. 1492–1499. https://doi.org/10.1109/ICDCS.2018.00154

[6] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. 2017. Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge. *ASPLOS* 52, 4 (2017), 615–629. https://doi.org/10.1145/3093336.3037698

[7] Jong Hwan Ko, Taesik Na, Mohammad Faisal Amir, and Saibal Mukhopadhyay. 2018. Edge-Host Partitioning of Deep Neural Networks with Feature Space Encoding for Resource-Constrained Internet-of-Things Platforms. In *IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. 1–6. https://doi.org/10.1109/AVSS.2018.8639121

[8] Jiagang Liu, Ju Ren, Yongmin Zhang, Xuhong Peng, Yaoxue Zhang, and Yuanyuan Yang. 2023. Efficient Dependent Task Offloading for Multiple Applications in MEC-Cloud System. *IEEE Transactions on Mobile Computing* 22, 4 (2023), 2147–2162. https://doi.org/10.1109/TMC.2021.3119200

[9] Quyuan Luo, Shihong Hu, Changle Li, Guanghui Li, and Weisong Shi. 2021. Resource Scheduling in Edge Computing: A Survey. *IEEE Communications Surveys and Tutorials* 23, 4 (2021), 2131–2165. https://doi.org/10.1109/COMST.2021.3106401

[10] Xinchen Lyu, Hui Tian, Wei Ni, Yan Zhang, et al. 2018. Energy-Efficient Admission of Delay-Sensitive Tasks for Mobile Edge Computing. *IEEE Transactions on Communications* 66, 6 (2018), 2603–2616. https://doi.org/10.1109/TCOMM.2018.2799937

[11] Wenqi Shi, Yunzhong Hou, Sheng Zhou, Zhisheng Niu, Yang Zhang, and Lu Geng. 2019. Improving Device-Edge Cooperative Inference of Deep Learning via 2-Step Pruning. In *IEEE Conference on Computer Communications Workshops*. 1–6.

[12] Chenmeng Wang, Chengchao Liang, F. Richard Yu, et al. 2017. Computation Offloading and Resource Allocation in Wireless Cellular Networks With Mobile Edge Computing. *IEEE Transactions on Wireless Communications* 16, 8 (2017), 4924–4938. https://doi.org/10.1109/TWC.2017.2703901

[13] Jin Wang, Jia Hu, Geyong Min, et al. 2021. Fast Adaptive Task Offloading in Edge Computing Based on Meta Reinforcement Learning. *IEEE Transactions on Parallel and Distributed Systems* 32, 1 (2021), 242–253. https://doi.org/10.1109/TPDS.2020.3014896

[14] Amy R. Ward and Ward Whitt. 2000. Predicting response times in processor-sharing queues. In *In Proc. of the Fields Institute Conf. on Comm. Networks*. 1–29.

[15] Mengwei Xu, Feng Qian, Mengze Zhu, et al. 2020. DeepWear: Adaptive Local Offloading for On-Wearable Deep Learning. *IEEE Transactions on Mobile Computing* 19, 2 (2020), 314–330. https://doi.org/10.1109/TMC.2019.2893250

[16] Changsheng You, Kaibin Huang, Hyukjin Chae, et al. 2017. Energy-Efficient Resource Allocation for Mobile-Edge Computation Offloading. *IEEE Transactions on Wireless Communications* 16, 3 (2017), 1397–1411. https://doi.org/10.1109/TWC.2016.2633522

[17] Tian Zhang. 2018. Data Offloading in Mobile Edge Computing: A Coalition and Pricing Based Approach. *IEEE Access* 6 (2018), 2760–2767. https://doi.org/10.1109/ACCESS.2017.2785265

[18] Weiwen Zhang and Yonggang Wen. 2018. Energy-Efficient Task Execution for Application as a General Topology in Mobile Cloud Computing. *IEEE Transactions on Cloud Computing* 6, 3 (2018), 708–719. https://doi.org/10.1109/TCC.2015.2511727

[19] Weiwen Zhang, Yonggang Wen, and Dapeng Oliver Wu. 2015. Collaborative Task Execution in Mobile Cloud Computing Under a Stochastic Wireless Channel. *IEEE Transactions on Wireless Communications* 14, 1 (2015), 81–93. https://doi.org/10.1109/TWC.2014.2331051

[20] Chao Zhu, Giancarlo Pastor, Yu Xiao, and et.al. 2018. Fog Following Me: Latency and Quality Balanced Task Allocation in Vehicular Fog Computing. In *2018 15th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*. 1–9. https://doi.org/10.1109/SAHCN.2018.8397129