

# Energy-Efficient Proactive Caching for Fog Computing with Correlated Task Arrivals

Hong Xing\*, Jingjing Cui<sup>§</sup>, Yansha Deng<sup>†</sup>, and Arumugam Nallanathan<sup>§</sup>

\*College of Information Engineering, Shenzhen University, Shenzhen, China

<sup>§</sup>School of EECS, Queen Mary University of London, London, U.K.

<sup>†</sup>Department of Informatics, King's College London, U.K.

E-mails: hong.xing@szu.edu.cn, j.cui@qmul.ac.uk, yansha.deng@kcl.ac.uk, nallanathan@ieee.org

**Abstract**—With the proliferation of latency-critical applications, fog-radio network (FRAN) has been envisioned as a paradigm shift enabling distributed deployment of cloud-clone facilities at the network edge. In this paper, we consider proactive caching for a one-user one-access point (AP) fog computing system over a finite time horizon, in which consecutive tasks of the same type of application are temporarily correlated. Under the assumption of predictable length of the task-input bits, we formulate a long-term weighted-sum energy minimization problem with three-slot correlation to jointly optimize computation offloading policies and caching decisions subject to stringent per-slot deadline constraints. The formulated problem is hard to solve due to the mixed-integer non-convexity. To tackle this challenge, first, we assume that task-related information are perfectly known *a priori*, and provide offline solution leveraging the technique of semi-definite relaxation (SDR), thereby serving as theoretical upper bound. Next, based on the offline solution, we propose a sliding-window based online algorithm under arbitrarily distributed prediction error. Finally, the advantage of computation caching as well the proposed algorithm is verified by numerical examples by comparison with several benchmarks.

**Index Terms**—Fog computing, mobile edge computing, computation caching, computation offloading, online algorithm.

## I. INTRODUCTION

Unprecedented growth of computation-intensive services (such as video streaming analysis, virtual reality (VR), and autonomous driving) prohibits the cloud-radio access network (CRAN) from continuously satisfying their latency-critical demands due to increasing transmission delay over long distance between the cloud and the users. To resolve such challenges, *fog-radio access network (FRAN)*, as an evolution of CRAN, is paving its way to provide ultra-reliable and low-latency (uRLLC) services for future wireless networks by pushing cloud-like capabilities, namely, *fog computing and edge caching*, to the network edge [1], [2].

Fog computing, also known as *mobile edge computing (MEC)*, endows the edge access points (APs) with computing and storage capacities, such that low-power wireless devices can seek nearby APs that are integrated with edge servers for task offloading, thus enabling energy-saving computation in real time. In the literature, a large amount of efforts have been devoted to achieving satisfied trade-offs between the cost of the network and latency by joint management of computation and

communication resource as well as task offloading decisions (see e.g., [3]–[5]).

Meanwhile, edge caching allows users to fetch popular contents from near by APs and/or users, thus alleviating the growing over-the-air traffic. Existing works have mainly focused on improving the efficiency of cache-enabled content distribution (see [6] and the references therein), whereas, caching aimed for saving the edge servers from repeated computing is less studied. The authors in [7] investigated proactive caching for achieving uRLLC in fog networks. However, they assumed that the popular computation tasks that had been cached *a priori* can be completely reused when requested later, which is too ideal in practice, since unlike content distribution, computation services usually adopt one-time data sets that are hardly rendered the same later on. Hence, it is crucial to understand what to cache by carefully exploiting the intrinsic data correlation among task arrivals. Note that although [8] and [9] considered joint service caching and task offloading, they did not model how the computation offloading can benefit from dynamic caching of correlated (not necessarily the same) task results.

In this work, we study proactive caching for a fog computing system consisting of one user terminal (UT) and one AP over a finite time horizon leveraging the correlation among *delay sensitive* task sequence such that the task results cached at the current slot can facilitate future computing. To our best knowledge, this is the first work aimed for minimizing the long-term weighted-sum energy by jointly optimizing computation offloading policies and caching decisions. With the correlation lying among three consecutive slots and imperfect task-input prediction, first, we provide an offline solution based on semi-definite relaxation (SDR), which serves as a performance upper bound. Next, we propose a sliding-window inspired online solution taking causally known prediction error into account. Finally, numerical results show striking performance gains brought by computation caching as well as the effectiveness of the proposed online algorithm.

We use the upper case boldface letters for matrices and lower case boldface ones for vectors. The superscripts  $(\cdot)^T$  and  $(\cdot)^*$  represent, respectively, the transpose and the optimum solution of vectors or matrices. We also denote the trace of a matrix by  $\text{Tr}(\cdot)$ .

## II. SYSTEM MODEL AND PROBLEM FORMULATION

We consider a fog computing system consisting of one UT equipped with one single antenna, and one access point (AP) equipped with  $M$  antennas, an edge server and cache facilities. During slot  $i$ , the UT solicits the nearby AP for computation task offloading. In this paper, we focus on a finite slotted-time horizon with each slot lasting  $T$  seconds, denoted by  $\mathcal{N} = \{1, \dots, N\}$ , over which sequential tasks featuring temporally correlated input data arrive at the UT as shown in Fig. 1. We assume that each task has to be executed by the end of the time slot. Since the computing results obtained at the current slot are also correlated with those at future slots, current task results can be cached at the AP to facilitate the future computation<sup>1</sup>. Due to the extra overhead caused by caching (delay, energy, storage), it may not be optimal to cache all the execution results at the edge server. Therefore, we introduce the following variable  $I_i$ ,  $i \in \mathcal{N}$ , to indicate whether the AP decides to cache the results at the end of slot  $i$ :

$$I_i = \begin{cases} 1, & \text{if the BS decides to cache the results,} \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

As a result, the workflow of the cache-enabled fog computing system in consideration can be described as follows. The UT offloads a proportion of the task to the AP while performing local computing for the rest of the task. If the AP decides not to cache the task results of the current slot, the UT just need to receive the execution results from the AP, otherwise the UT is also required to upload its local computing results to the AP at the end of the current slot. Since the AP is usually of sufficient communications resource. e.g., high transmitting power, we ignore the delay/energy caused by results downloading at the UT in the sequel.

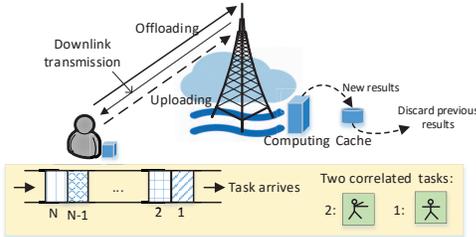


Fig. 1. System model of the one-user one-server fog computing system.

### A. Local Execution, Task Offloading and Computation Uploading at the UT

We assume that the length of task-input bits at slot  $i \in \mathcal{N}$ , denoted by  $L_i$ 's, is predictable but with finite estimation error shown as  $L_i = \hat{L}_i + \Delta L_i$ , in which  $\{\Delta L_i\}$  can be an arbitrary (deterministic or stochastic) sequence. At any slot  $i \in \mathcal{N}$ , the exact task-input length up to slot  $i$ , i.e.,  $L_k$ 's for  $k \leq i$ , is

<sup>1</sup>A typical example is matrix-vector multiplication of  $\mathbf{y}_i = \mathbf{A}\mathbf{x}_i$ ,  $i \in \mathcal{N}$ , where  $\mathbf{x}_i$ 's is the encoded task-input data. Supposing  $\mathbf{x}_i = \mathbf{x}_{i-1} + \boldsymbol{\varepsilon}$  with a sparse error  $\boldsymbol{\varepsilon}$ , the current computation can benefit from caching at slot  $i-1$  by executing only  $\boldsymbol{\varepsilon}$  with much shorter input length.

known to the AP, while only the predicted task-input length, i.e.,  $\hat{L}_k$ 's for  $k > i$ , is available for all future slots. We model the task-input bits that are required to be executed at slot  $i$  in terms of previous caching decisions as follows:

$$D_i = L_i \left( (I_{i-1}\tau_1 + \dots + \prod_{j=1}^{k-1} (1 - I_{i-j}) I_{i-k} \tau_k + \dots + \prod_{j=1}^{r-1} (1 - I_{i-j}) I_{i-r} \tau_r + \prod_{j=1}^r (1 - I_{i-j}) \right), \quad (2)$$

where  $\boldsymbol{\tau} = [\tau_1, \dots, \tau_r]^T$  with increasing  $\tau_j \in [0, 1]$ ,  $j = 1, \dots, r$ , is a prescribed vector capturing the diminishing effect of the previously cached results on reducing the current task-input length. Note that only the latest cached results are useful. (E.g., if  $I_{i-1} = 0$  and  $I_{i-2} = 1$ , (2) reduces to  $D_i = L_i \tau_2$  in spite of the values that  $I_{i-k}$  for  $k \geq 3$  take.) In addition, any results cached far more than  $r$  slots before are assumed to be no longer exploitable.

**Local Execution** The cache-enabled task-input bits  $D_i$ 's will be divided into  $l_i$  and  $D_i - l_i$  for local and remote execution, respectively, where  $l_i \in [0, D_i]$ . The required CPU cycles for UT's local execution at slot  $i$  is given by  $c_{\text{loc}} l_i$  [3], where  $c_{\text{loc}}$  in cycles per bit depends on the application type and the CPU architecture of the UT. Assuming constant CPU frequency  $f_{\text{loc}}$  adopted by the UT, the corresponding energy consumption for local computation at slot  $i$  is expressed as [10]

$$E_{c,i}^{\text{loc}} = \kappa_{\text{loc}} c_{\text{loc}} l_i f_{\text{loc}}^2, \quad (3)$$

where  $\kappa_{\text{loc}}$  is the effective capacitance coefficient of the UT's CPU chip.

**Task Offloading** By applying maximum ratio combing (MRC) at the AP, the achievable offloading rate at slot  $i$ ,  $i \in \mathcal{N}$ , is thus given by  $r_i^{\text{off}} = B_{\text{off}} \log_2(1 + p_i h_i)$ , where  $h_i$  is the normalized channel gain from the UT to the AP at slot  $i$ , and  $B_{\text{off}}$  is the pre-assigned transmission bandwidth (BW) for task offloading<sup>2</sup>. It thus takes  $t_i^{\text{off}} = (D_i - l_i) / r_i^{\text{off}}$ , for task offloading, and the associated energy consumption for task offloading is given by

$$E_i^{\text{off}} = \frac{p_i (D_i - l_i)}{r_i^{\text{off}}}. \quad (4)$$

**Computation Uploading** Suppose that there is little cache capacity allocated for computation caching at the UT. When the current task results are decided to be cached at the end of slot  $i$ ,  $i \in \mathcal{N}$ , the UT needs to upload its locally executed results to the AP so as to maintain the integrity of computation results for future use. Given the UT's uploading rate,  $r_i^{\text{up}}$ 's, the consumed energy for computation uploading at the UT is thus given by

$$E_i^{\text{up}} = I_i \frac{p_i R_i}{r_i^{\text{up}}}, \quad (5)$$

<sup>2</sup>We assume that frequency division multiple access (FDMA) is adopted for task offloading and computation results uploading, respectively.

where  $R_i$  is the length of the task-output bits, which is assumed to have been perfectly profiled given the type of application (c.f. footnote 1).

### B. Remote Execution and Computation Caching at AP

In the considered model, the AP is responsible for profiling the task information ( $\hat{L}_i$ 's and  $R_i$ 's) as well as the channel state information (CSI) ( $h_i$ 's  $g_i$ 's), and collecting other required information a priori. Based on these information, the AP will dynamically make and inform the UT of the caching decisions and the offloading policies.

**Remote Execution** Similar to (3), the energy consumption for remote execution is expressed as

$$E_{c,i}^e = \kappa_e c_e (D_i - l_i) f_e^2, \quad (6)$$

where  $\kappa_e$  and  $c_e$  denote the effective capacitance coefficient, and the number of cycles required for executing one task-input bit at the edge server's CPU, respectively.

**Computation Caching** If the AP decides to cache the task results at the current slot, it then expects to receive UT's uploading of its local computing results before combining them with the remotely executed task to form an integrated copy of the task results ready for caching.

### C. Problem Formulation

We are interested in minimizing the total weighted-sum energy consumption over the finite horizon  $\mathcal{N}$ , i.e.,  $\sum_{i \in \mathcal{N}} (\alpha_1 (E_{c,i}^{\text{loc}} + E_i^{\text{off}} + E_i^{\text{up}}) + \alpha_0 E_{c,i}^e)$ , where  $\alpha_1$  and  $\alpha_0$  satisfying  $\alpha_1 + \alpha_0 = 1$  are the coefficients balancing the energy saving priority between the UT and the AP. Under the per-slot deadline constraint for each task, we aim to jointly optimize the computation offloading policies  $\{l_i\}$  and the binary caching decisions  $\{I_i\}$ . Combining (3), (4), (5), and (6), the long-term energy minimization problem is thus formulated as:

$$(P1) : \underset{\{l_i, I_i\}}{\text{Min}} \sum_{i \in \mathcal{N}} \left( \alpha_1 \left( \kappa_{\text{loc}} c_{\text{loc}} l_i f_{\text{loc}}^2 + I_i \frac{p_i R_i}{r_i^{\text{up}}} + \frac{p_i (D_i - l_i)}{r_i^{\text{off}}} \right) + \alpha_0 \kappa_e c_e (D_i - l_i) f_e^2 \right)$$

$$\text{s.t. } \frac{D_i - l_i}{r_i^{\text{off}}} + \frac{c_e (D_i - l_i)}{f_e} \leq T, \quad \forall i \in \mathcal{N}, \quad (7a)$$

$$\frac{c_{\text{loc}} l_i}{f_{\text{loc}}} + I_i \frac{R_i}{r_i^{\text{up}}} \leq T, \quad \forall i \in \mathcal{N}, \quad (7b)$$

$$0 \leq l_i \leq D_i, \quad \forall i \in \mathcal{N}, \quad (7c)$$

$$I_i \in \{0, 1\}, \quad \forall i \in \mathcal{N}. \quad (7d)$$

## III. OFFLINE COMPUTATION OFFLOADING AND CACHING

In this section, we consider offline solution for problem (P1) by assuming that the predictable task-input length  $\{L_i\}$  are perfectly known *a priori* at the AP. The offline solution thus serves as fundamental performance upper bound for all other online schemes that are designed for practical implementation. In this paper, we focus on a special case of  $r = 2$  (c.f. (2)). More general cases will be studied in our future work.

The major difficulty for solving (P1) lies in the binary variables  $I_i$ 's. To tackle this challenge, first, we equivalently formulate (7d) as  $I_i(I_i - 1) = 0, \forall i \in \mathcal{N}$ , and then transform the problem into a quadratically constrained quadratic program (QCQP) in terms of  $\mathbf{I} = [I_1, \dots, I_N]^T$ . Next, we convert the QCQP into a semi-definite programming (SDP) as follows. First, we define  $\mathbf{F}' = [(1 - \tau_2)\mathbf{F}, \frac{1}{2}\mathbf{v}; \frac{1}{2}\mathbf{v}^T, 0]$ , where  $\mathbf{F} = \sum_{i=3}^N \frac{p_i L_i}{r_i^{\text{off}}} \mathbf{G}_{i-2, i-1}$ ,  $\mathbf{G}_{i-2, i-1}$  is a symmetric matrix with only  $\mathbf{G}_{i-2, i-1}(i-2, i-1)$  and  $\mathbf{G}_{i-2, i-1}(i-1, i-2)$  being  $\frac{1}{2}$ ,  $\mathbf{v} = (\tau_1 - 1) \sum_{i=2}^N \frac{p_i L_i}{r_i^{\text{off}}} \mathbf{e}_{i-1} + (\tau_2 - 1) \sum_{i=3}^N \frac{p_i L_i}{r_i^{\text{off}}} \mathbf{e}_{i-2}$ , and  $\mathbf{e}_j$  denotes a vector with only the  $j$ th element being 1;  $\mathbf{W} = [\mathbf{0}_{N \times N}, \frac{1}{2}\mathbf{w}; \frac{1}{2}\mathbf{w}^T, 0]$ , where  $\mathbf{w} = [\frac{p_1 R_1}{r_1^{\text{up}}}, \dots, \frac{p_N R_N}{r_N^{\text{up}}}]^T$ ;  $\mathbf{u} = [\frac{p_1}{r_1^{\text{off}}}, \dots, \frac{p_N}{r_N^{\text{off}}}]^T$ ,  $\mathbf{G}' = [(1 - \tau_2)\mathbf{G}, \frac{1}{2}\mathbf{s}; \frac{1}{2}\mathbf{s}^T, 0]$ , where  $\mathbf{G} = c_e \sum_{i=3}^N L_i \mathbf{G}_{i-2, i-1}$ , and  $\mathbf{s} = (\tau_1 - 1) c_e \sum_{i=2}^N L_i \mathbf{e}_{i-1} + (\tau_2 - 1) c_e \sum_{i=3}^N L_i \mathbf{e}_{i-2}$ ;  $\mathbf{E}_i = [\mathbf{0}_{N \times N}, \frac{1}{2}\mathbf{e}_i; \frac{1}{2}\mathbf{e}_i^T, 0]$ ; and  $\mathbf{U}_i = [\text{diag}(\mathbf{e}_i), -\frac{1}{2}\mathbf{e}_i; -\frac{1}{2}\mathbf{e}_i^T, 0]$ . Next, we introduce  $\mathbf{a} = [\mathbf{I}; 1]$  and  $\mathbf{A} = \mathbf{a}\mathbf{a}^T$ . Then, by relaxing the rank-one constraint for  $\mathbf{A}$  [11] and some manipulations, problem (P1) is recast into an SDP as shown in the following proposition.

**Proposition 3.1:** By relaxing the rank-one constraint, problem (P1) is equivalent to an SDP shown below:

$$(P1') : \underset{\mathbf{A}, \mathbf{l}}{\text{Min}} \alpha_1 (\text{Tr}(\mathbf{A}\mathbf{F}') + \text{Tr}(\mathbf{A}\mathbf{W}) - \mathbf{u}^T \mathbf{l} +$$

$$\kappa_{\text{loc}} c_{\text{loc}} f_{\text{loc}}^2 \mathbf{1}^T \mathbf{l}) + \alpha_0 \kappa_e f_e^2 (\text{Tr}(\mathbf{A}\mathbf{G}') - c_e \mathbf{1}^T \mathbf{l})$$

$$\text{s.t. } \left( \frac{1}{r_i^{\text{off}}} + \frac{c_e}{f_e} \right) (D_i(\mathbf{A}) - \mathbf{e}_i^T \mathbf{l}) \leq T, \quad \forall i \in \mathcal{N}, \quad (8a)$$

$$\frac{R_i}{r_i^{\text{up}}} \text{Tr}(\mathbf{A}\mathbf{E}_i) + \frac{c_{\text{loc}}}{f_{\text{loc}}} \mathbf{e}_i^T \mathbf{l} \leq T, \quad \forall i \in \mathcal{N}, \quad (8b)$$

$$\mathbf{e}_i^T \mathbf{l} - D_i(\mathbf{A}) \leq 0, \quad \forall i \in \mathcal{N}, \quad (8c)$$

$$\text{Tr}(\mathbf{A}\mathbf{U}_i) = 0, \quad \forall i \in \mathcal{N}, \quad (8d)$$

$$\mathbf{A}(N+1, N+1) = 1, \quad (8e)$$

$$\mathbf{l} \geq 0, \quad \mathbf{A} \succeq 0. \quad (8f)$$

*Proof:* Due to the space limitation, we only provide a key step in the proof, i.e., to express  $D_i$ 's in terms of  $\mathbf{A}$ . Since  $D_i = L_i((\tau_1 - 1)I_{i-1} + (\tau_2 - 1)I_{i-2} + (1 - \tau_2)I_{i-1}I_{i-2} + 1)$ ,  $i \geq 3$ , it follows that  $D_i = L_i \text{Tr}(\mathbf{A}\mathbf{H}_i)$ ,  $i \geq 3$ , where  $\mathbf{H}_i = [(1 - \tau_2)\mathbf{G}_{i-2, i-1}, \frac{1}{2}((\tau_1 - 1)\mathbf{e}_{i-1} + (\tau_2 - 1)\mathbf{e}_{i-2}); \frac{1}{2}((\tau_1 - 1)\mathbf{e}_{i-1} + (\tau_2 - 1)\mathbf{e}_{i-2})^T, 1]$ . ■

As (P1') is an SDP, we can solve (P1') by some off-the-shelf convex software tools, such as CVX [12]. Since there is no guarantee that  $\mathbf{A}^*$  for (P1') is rank-one, it in general only serves as a lower-bound solution for (P1). To construct the binary caching decisions, we need to retrieve  $\mathbf{I}$  from  $\mathbf{A}^*$ . Specifically, if  $\text{rank}(\mathbf{A}^*) = 1$ ,  $\mathbf{I}^*$  can be recovered by singular-value decomposition (SVD) such that  $\mathbf{A}^* = \mathbf{a}^* \mathbf{a}^{*T}$ . Otherwise, we propose to approximate  $I_i$ 's as follows.

$$I_i^{\text{app}} = \text{round}(A^*(i, N+1)), \quad i \in \mathcal{N}, \quad (9)$$

which is based on the following lemma [4].

**Lemma 3.1:** The optimum  $\mathbf{A}^*$  for problem (P1') satisfies  $\mathbf{A}^*(i, N+1) \in [0, 1]$ ,  $i \in \mathcal{N}$ .

Once  $\mathbf{I}^{\text{app}}$  is ready, the corresponding offloading policies  $\mathbf{l}^{\text{app}}$  can be easily obtained by solving (P1') with  $\mathbf{A} = \mathbf{a}^{\text{app}} \mathbf{a}^{\text{app}T}$  fixed ( $\mathbf{a}^{\text{app}} = [\mathbf{I}^{\text{app}}; \mathbf{1}]$ ), which then turns out to be a linear programming (LP) problem in terms of  $\mathbf{l}$ .

As per Lemma 3.1, when  $\mathbf{A}^*$  is rank-one, the approximation is tight because  $I_i^* = a_i^* = a_i^* a_{N+1}^* = \mathbf{A}^*(i, N+1)$ . It thus implies that the effectiveness of the approximated caching decisions primarily depends on the rank property of  $\mathbf{A}^*$ . The following proposition reveals a sufficient condition for achieving low-rank  $\mathbf{A}^*$  that is easily satisfied in practice [4].

**Proposition 3.2:** When the constraints given by (8c) are all inactive, i.e., non-zero task offloading at all the slots,  $\text{rank}(\mathbf{A}^*) \leq 2$ .

*Proof:* Please refer to Appendix A. ■

#### IV. ONLINE COMPUTATION OFFLOADING AND CACHING

In the previous section, we have provided an SDR-based offline solution under the ideal assumption that the random task-input length  $L_i$ 's is perfectly predicted without error. In this section, inspired by the offline solution, we propose a sliding-window based online scheme that applies to error sequence  $\{\Delta L_1, \dots, \Delta L_N\}$  following arbitrary stochastic process [13].

Specifically, as stated in Section II, at any slot  $i$ , the exact task-input length is perfectly known up to the current slot, i.e.,  $\{L_1, \dots, L_i\}$ , whereas only the predictable task-input length, i.e.,  $\hat{L}_{i+1}, \dots, \hat{L}_N$ , is available for all future slots. First, we define a set  $\mathcal{S} = \{1, \dots, S\}$ , where  $S$  is the length of the sliding-window. Note that since the parameter  $S$  balances between exploitation of the long-term prediction and accuracy of the algorithm, it is required to be carefully chosen in practice. Second, we focus on minimizing the weighted-sum energy over the span of the sliding-window from slot  $i$ , i.e., slots  $\{i, \dots, i+S-1\}$ . Then, by specifying the parameters using their consecutive  $S$ -slot values from slot  $i^3$ , e.g.,  $\{L_1^{(i)}, L_2^{(i)}, \dots, L_S^{(i)}\} = \{L_i, \hat{L}_{i+1}, \dots, \hat{L}_{i+S-1}\}$ , we sequentially solve the following problem for all the slots.

$$\begin{aligned}
& \text{(P1-ol)} : \underset{\mathbf{A}^{(i)}, \mathbf{I}^{(i)}}{\text{Min}} \alpha_1 (\text{Tr}(\mathbf{A}^{(i)} \mathbf{F}^{(i)}) + \text{Tr}(\mathbf{A}^{(i)} \mathbf{W}^{(i)}) - \mathbf{u}^{(i)T} \mathbf{l}^{(i)}) \\
& + \kappa_{\text{loc}} c_{\text{loc}} f_{\text{loc}}^2 \mathbf{1}^T \mathbf{l}^{(i)} + \alpha_0 \kappa_e f_e^2 (\text{Tr}(\mathbf{A}^{(i)} \mathbf{G}^{(i)}) - c_e \mathbf{1}^T \mathbf{l}^{(i)}) \\
& \text{s.t.} \left( \frac{1}{r_{\text{off}}^{(i)}} + \frac{c_e}{f_e} \right) (D_j^{(i)}(\mathbf{A}^{(i)}) - e_j^T \mathbf{l}^{(i)}) \leq T, \forall j \in \mathcal{S}, \\
& \frac{R_j^{(i)}}{r_{\text{up}}^{(i)}} \text{Tr}(\mathbf{A}^{(i)} \mathbf{E}_j) + \frac{c_{\text{loc}}}{f_{\text{loc}}} e_j^T \mathbf{l}^{(i)} \leq T, \forall j \in \mathcal{S}, \\
& e_j^T \mathbf{l}^{(i)} - D_j^{(i)}(\mathbf{A}^{(i)}) \leq 0, \forall j \in \mathcal{S}, \\
& \text{Tr}(\mathbf{A}^{(i)} \mathbf{U}_j) = 0, \forall j \in \mathcal{S}, \\
& \mathbf{A}^{(i)}(N+1, N+1) = 1, \\
& \mathbf{l}^{(i)} \geq 0, \mathbf{A}^{(i)} \succeq 0,
\end{aligned}$$

<sup>3</sup>When the last index of the sliding-window exceeds  $N$ , we substitute the (prediction) values of the parameters from slot 1 to  $S-1$  for those from slot  $N+1$  to  $N+S-1$ .

where  $e_j$ ,  $j \in \mathcal{S}$ , is similarly defined as in (P1'), and so are  $\mathbf{E}_j$ 's and  $\mathbf{U}_j$ 's through proper dimension modification. Next, by reconstructing  $\mathbf{I}^{\text{app}(i)}$  and  $\mathbf{l}^{\text{app}(i)}$  from the solution to (P1-ol), we attain the proposed online computation offloading policies  $\{\tilde{l}_i\}$  and caching decisions  $\{\tilde{I}_i\}$  by  $\tilde{l}_i = l_1^{\text{app}(i)}$  and  $\tilde{I}_i = I_1^{\text{app}(i)}$ ,  $i \in \mathcal{N}$ , respectively. The above procedure for the online scheme is summarized in Table I.

TABLE I  
PROPOSED ONLINE ALGORITHM FOR PROBLEM (P1)

---

**Require:**  $i \leftarrow 1$

1: **repeat**

2: Solve (P1-ol) at slot  $i$ , and obtain its optimal solution  $\mathbf{A}^{(i)*}$ ;

3: Reconstruct  $\mathbf{I}^{\text{app}(i)}$  based on  $\mathbf{A}^{(i)*}$  by similar means of (9);

4: Given  $\mathbf{I}^{\text{app}(i)}$ , solve the reduced LP associated with (P1-ol) to obtain  $\mathbf{l}^{\text{app}(i)}$ ;

5:  $\tilde{I}_i \leftarrow I_1^{\text{app}(i)}$  and  $\tilde{l}_i \leftarrow l_1^{\text{app}(i)}$ ;

6:  $i \leftarrow i + 1$ .

7: **until**  $i = N$

**Ensure:**  $\{\tilde{I}_i, \tilde{l}_i\}$

---

#### V. NUMERICAL RESULTS

In this section, we verify the effectiveness of our proposed online computation offloading and caching scheme against theoretical performance upper bound and other benchmark schemes through numerical simulations. Specifically, 'Lower-bound' shows the optimal solution to (P1') based on SDR, which is only achievable when the approximation is tight; 'Random caching' is obtained by setting  $\{I_i\}$  as a  $\frac{1}{2}$ -Bernoulli process; 'No caching' refers to the results ignoring the correlation among task-input data; and "All caching" provides the case when  $\{I_i = 1\}$ . At each slot, we consider Rayleigh fading channel models with the distance-dependent pathloss set as  $-117\text{dB}$  (0.5km) over transmission BWs of  $B_{\text{off}} = B_{\text{up}} = 2.5\text{MHz}$ . The estimation of the task-input length follows a uniform distribution, denoted by  $\hat{L}_i \sim \mathcal{U}[10^5, 10^6]\text{bits}$ ,  $i \in \mathcal{N}$ , and the profile of the associated task-output length is set as  $R_i \sim \mathcal{U}[10^5, 10^6]\text{bits}$ . Other parameters are set as follows unless otherwise specified:  $M = 3$ ;  $\alpha_1 = 0.85$ ,  $\alpha_0 = 0.15$ ;  $\{\tau_1 = \frac{1}{2}, \tau_2 = \frac{3}{4}\}$ ;  $\{p_i = 24\}\text{dBm}$ ;  $f_{\text{loc}} = 800\text{MHz}$ ,  $f_e = 2\text{GHz}$ ;  $C_{\text{loc}} = C_e = 10^3$  cycles/bit; and  $\kappa_{\text{loc}} = \kappa_e = 10^{-28}$ . The results shown below are obtained by averaging over 500-time realizations of the predication error sequence  $\{\Delta L_i\}$ , in which  $\Delta L_i$ 's is modelled as *i.i.d.* Gaussian variables with zero mean and variance of  $\sigma^2$ .

Fig. 2 shows the average weighted-sum energy versus the computation deadline  $T$  with  $\sigma^2 = 10^4$ . It is observed that the weighted-sum energy for all the schemes gradually goes down as the per-slot deadline gets extended, which is intuitively true, since the longer  $T$  is, the higher the chances that more of the task-input bits can be executed locally within the deadline, which thus saves UT's energy for task offloading. The approximate offline solution is also shown to approach the lower-bound SDR solution with negligible gap. Furthermore, the proposed online joint computation offloading and caching scheme outperforms all the other fixed-caching schemes, which corroborates the importance of computation caching in latency-critical scenarios.

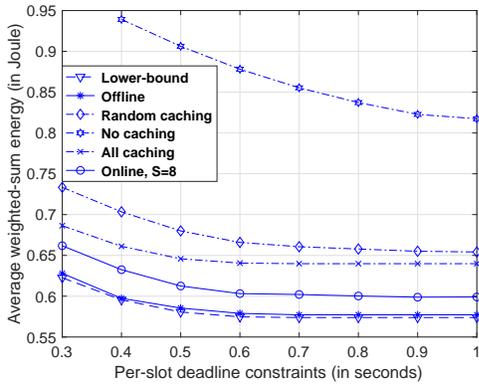


Fig. 2. Average weighted-sum energy versus the per-slot deadline constraint.

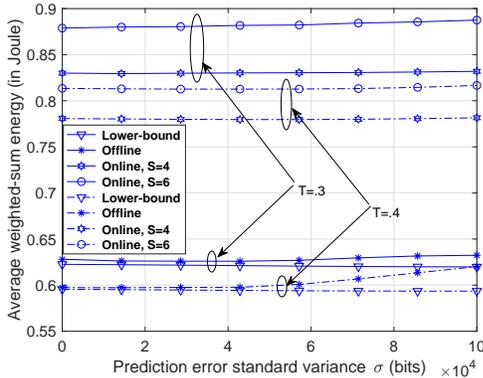


Fig. 3. Average weighted-sum energy versus standard variance of the task-input length prediction error.

Fig. 3 demonstrates the average weighted-sum energy versus the standard variance of the prediction error  $\Delta L_i$ 's. It is seen that our online algorithms under different deadline constraints are overall robust against a wide range of standard variance. In both cases of  $T = 0.3$  and  $T = 0.4$  seconds, the performance of the online scheme with a window length of  $S = 6$  is inferior to that with a window length of  $S = 4$  with noticeably larger gap in the more strict deadline constraint of  $T = 0.3$ , which is due to the advantage of the short-size window in coping with uncertainties. Furthermore, the online algorithm with  $S = 6$  becomes worse off when  $\sigma$  exceeds about  $5 \times 10^4$  ( $7 \times 10^4$ ) in the case of  $T = .3$  ( $T = .4$ ) seconds, since the effectiveness of the long-term prediction starts being compromised by the increasing estimation error.

## VI. CONCLUSION

This paper studied a one-UT one-AP fog computing system over a finite time-slotted horizon, in which each computation task was required to be executed by the end of the slot, and dynamic computation caching was allowed such that the AP could decide whether to cache the current task results for relieving its computation burden in the future. Under the assumption of three-slot correlation and imperfect estimation of the task-input bit-length, a joint computation offloading and caching optimization problem was formulated to minimize the

long-term weighted-sum energy consumption of the UT and the AP. To tackle the challenging mixed-integer non-convex problem, we approximated the problem by an SDP, based on which an offline solution assuming perfect knowledge of task-input length was provided, while a sliding-window based online scheme was also developed to cater for unknown prediction error of the future task arrivals. By comparison with several benchmark schemes, the proposed online algorithm with short-size window demonstrated striking robustness against prediction error. In addition, the approximation was also shown to be near-optimal by numerical examples under practical settings.

## APPENDIX A

Only a sketch of the proof is provided herein due to the space limitation, and detailed proof will be presented in the longer version of this paper. First, by providing the (partial) Lagrangian of (P1') in terms of  $\mathbf{A}^*$  and the associated Karush-Kuhn-Tucker (KKT) conditions, show that  $\mathbf{A}^* \in \mathbb{R}^{(N+1) \times (N+1)}$  lies in the null space of a matrix containing a tri-diagonal sub-matrix. Next, show that under the above sufficient condition, the rank of this matrix is no less than  $N - 1$ , and thus  $\text{rank}(\mathbf{A}^*) \leq 2$  is proved.

## REFERENCES

- [1] R. Tandon and O. Simeone, "Harnessing cloud and edge synergies: toward an information theory of fog radio access networks," *IEEE Commun. Mag.*, vol. 54, no. 8, pp. 44–50, Aug. 2016.
- [2] Y. Y. Shih, W. H. Chung, A. C. Pang, T. C. Chiu, and H. Y. Wei, "Enabling low-latency applications in fog-radio access networks," *IEEE Netw.*, vol. 31, no. 1, pp. 52–58, Jan. 2017.
- [3] F. Wang, J. Xu, X. Wang, and S. Cui, "Joint offloading and computing optimization in wireless powered mobile-edge computing systems," *IEEE Trans. Wireless Commun.*, vol. 17, no. 3, pp. 1784–1797, Mar. 2018.
- [4] M. Chen, B. Liang, and M. Dong, "Multi-user multi-task offloading and resource allocation in mobile cloud systems," *IEEE Trans. Wireless Commun.*, vol. 17, no. 10, pp. 6790–6805, Oct. 2018.
- [5] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 12, pp. 3590–3605, Dec. 2016.
- [6] E. Bastug, M. Bennis, and M. Debbah, "Living on the edge: The role of proactive caching in 5g wireless networks," *IEEE Commun. Mag.*, vol. 52, no. 8, pp. 82–89, Aug. 2014.
- [7] M. S. Elbamby, M. Bennis, and W. Saad, "Proactive edge computing in latency-constrained fog networks," in *Proc. European Conference on Networks and Communications (EuCNC)*, Oulu, Finland, Jun. 2017.
- [8] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *Proc. IEEE International Conference on Computer Communications (INFOCOM)*, Honolulu, HI, USA, Apr. 2018.
- [9] Y. Hao, M. Chen, L. Hu, M. S. Hossain, and A. Ghoneim, "Energy efficient task caching and offloading for mobile edge computing," *IEEE Access*, vol. 6, pp. 11 365–11 373, Mar. 2018.
- [10] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, third quart. 2017.
- [11] Z.-Q. Luo, W.-K. Ma, A. M.-C. So, Y. Ye, and S. Zhang, "Semidefinite relaxation of quadratic optimization problems," *IEEE Signal Process. Mag.*, vol. 27, no. 3, pp. 20–34, May 2010.
- [12] M. Grant and S. Boyd, "CVX: Matlab software for disciplined convex programming, version 2.1," Mar. 2014. [Online]. Available: <http://cvxr.com/cvx>
- [13] K. Rahbar, J. Xu, and R. Zhang, "Real-time energy storage management for renewable integration in microgrid: An off-line optimization approach," *IEEE Trans. Smart Grid*, vol. 6, no. 1, pp. 124–134, Jan. 2015.