Proactive Edge Computing in Latency-Constrained Fog Networks

Mohammed S. Elbamby*, Mehdi Bennis*†, and Walid Saad‡
*Centre for Wireless Communications, University of Oulu, Finland,
emails: {mohammed.elbamby,mehdi.bennis}@oulu.fi
†Department of Computer Engineering, Kyung Hee University, South Korea
‡Wireless@VT, Bradley Department of Electrical and Computer Engineering,
Virginia Tech, Blacksburg, VA, USA, email: walids@vt.edu

Abstract—In this paper, the fundamental problem of distribution and proactive caching of computing tasks in fog networks is studied under latency and reliability constraints. In the proposed scenario, computing can be executed either locally at the user device or offloaded t o a n e dge c loudlet. Moreover, cloudlets exploit both their computing and storage capabilities by proactively caching popular task computation results to minimize computing latency. To this end, a clustering method to group spatially proximate user devices with mutual task popularity interests and their serving cloudlets is proposed. Then, cloudlets can proactively cache the popular tasks' computations of their cluster members to minimize computing latency. Additionally, the problem of distributing tasks to cloudlets is formulated as a matching game in which a cost function of computing delay is minimized under latency and reliability constraints. Simulation results show that the proposed scheme guarantees reliable computations with bounded latency and achieves up to 91% decrease in computing latency as compared to baseline schemes.

I. INTRODUCTION

The emergence of the Internet of things (IoT) and machine-to-machine communication is paving the way for a seamless connectivity of a massive number of resource-limited devices and sensors [1]. The unprecedented amount of IoT data communication and computation requirements impose stringent requirements in end-to-end latency, mandating ultra-reliable and low-latency communications (URLLC). However, the finite computation capabilities of end-user devices challenge the possibility of coping with the stringent computing and processing latency requirements of IoT networks. Therefore, mobile cloud computing (MCC) services have been recently proposed to allow end-users to offload their resource consuming tasks to remote cloud centers. However, despite having high computational resources, MCC solutions are inefficient in handling latency-critical computing services due to the high propagation delays between the end-user device and the cloud data center.

Recently, the idea of fog computing has been introduced [2] to bring computing resources closer to where tasks are requested. In order to minimize computing latency in fog networks, smarter communication and computing resource utilization schemes are needed [3]. A centralized joint communication and computation resource allocation scheme is proposed in [4]. The power-delay tradeoff in centralized mobile edge computing (MEC) systems is discussed in [5] using tools from stochastic optimization. However, these works rely on centralized solutions in which the MEC network has information about all users requests and channel-state information (CSI). Game-theoretic solutions are studied in [6] to design decentralized computing offloading schemes in cases of homogeneous and heterogeneous users. Recently, an online secretary framework for fog network formation is proposed in [7] under uncertainty on the arrival process of fog nodes.

While interesting, the vast majority of the literature in fog networking is based on the reactive computing paradigm in which task computing starts only after the task data is offloaded to the fog node [3]. Moreover, prior art has not explicitly accounted for stringent latency and reliability constraints in fog networks. Due to the distributed nature of these networks, having computing resources closer to the network edge allows for providing personalized type of computing services to end-users [3]. Clearly, harnessing the correlation between end-user requests motivates the need for *proactive computing* to minimize computing latency. For example, for augmented reality (AR) services provided in a museum, proactively computing popular AR services of visitors can aid in minimizing computational latency [8]. Using proactive computing, the fog network can keep track of the popularity patterns of user tasks and cache their computing results *in advance*. This eliminates the need to request the task data multiple times thus reducing the burden on the task offloading transmissions [3]. A possible first step towards proactive computing is the idea of task data prefetching [9], in which part of the upcoming task data is predicted and prefetched during the computing of the current one such that the fetching time is minimized.

While the idea of proactive networks has been recently studied in the context of wireless content caching, such as [10] and [11], none of these works investigate the problem of proactive caching of computing tasks. In contrast to content caching, computing caching poses new challenges. First, while in content caching, popular contents are prefetched from the core network during off-peak times to alleviate the burden on the backhaul links, computing caching decreases the load on the access link by providing computing results to end-user nodes (UNs) without the need to prefetch their task data beforehand. Second, computing tasks can be of diverse types
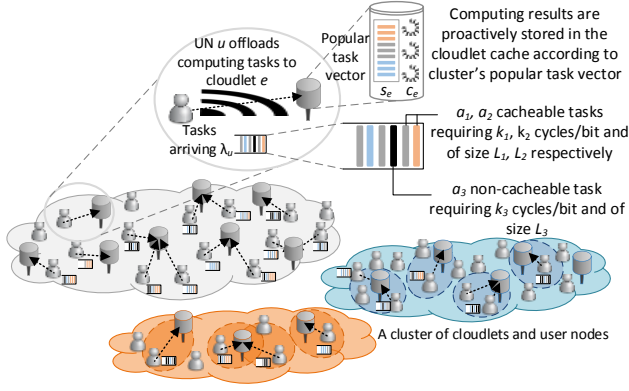
Figure 1. An illustration of the cache-enabled fog network model

and depend on the computing environment, while some of the content is cacheable for reuse by other devices, personal computing data is not cacheable, and must often be computed in real time. Finally, due to the nature of IoT networks, with large number of deployed servers and low density of UNs per server [12], it is not practical to build popularity patterns locally at each server. Instead, studying popularity distributions over larger sets of servers can provide a broader view on the popularity patterns of computing tasks.

The main contribution of this paper is to investigate the problem of edge computing and proactive edge caching in fog computing networks. We exploit both computing and storage resources to minimize computing latency via joint task offloading and proactive caching of *popular* and *cacheable* computing tasks. A cacheable task has a computing result that can be reused by several other devices. Moreover, we impose constraints on computing latency and reliability so as to ensure computing delay bounds with high levels of reliability. In the proposed framework, clusters of UNs and their serving edge computing nodes (cloudlets) are formed based on spatial proximity and mutual interests in popular tasks. Accordingly, cloudlets proactively cache computed results of popular tasks in their clusters thereby ensuring minimal latency. Moreover, the problem of task distribution to cloudlets is modeled as a matching game between cloudlets and UNs. To this end, an efficient distributed matching algorithm is proposed to reach a stable matching of UN requests to cloudlets or to their local device such that the minimal task computing delay is incurred and a reliable service latency is guaranteed. Simulation results show that the proposed framework can guarantee reliable computations with bounded latency and achieve up to 91% decrease in computing latency as compared to baseline schemes.

The rest of this paper is organized as follows. Section II describes the system model and problem formulation. The proposed clustering scheme as well as the joint caching and matching scheme are studied in Section III. The performance of the proposed framework is analyzed in Section IV. Finally, Section V concludes the paper.

II. SYSTEM MODEL

Consider a fog network that consists of a set $\mathcal{E}$ of E cloudlets, each of which having a CPU computing and storage capability of c_e and s_e, respectively, and a set $\mathcal{U}$ of U UNs that are distributed uniformly over the network area. Cloudlets share the same frequency channel and operate in time-division-duplex (TDD). In our model, we focus on the uplink transmission. UNs have computing tasks that arrive following a Poisson process with mean λ_u. UNs are interested in a set $\mathcal{A}$ of A tasks. Each task has a required CPU cycles of κ per bit of task data, and the task data size follows an exponential distribution of mean L_a. UNs can offload their computing tasks to any cloudlet within their coverage, where coverage is decided based on a threshold path loss value[1]. Task data must be offloaded to the cloudlet prior to computation. To minimize the task data offloading delay, cloudlets proactively cache the computed results of the most popular cacheable tasks. We assume that a subset $\mathcal{A}_c \subset \mathcal{A}$ of the tasks are cacheable and another subset $\mathcal{A}_{nc} \subset \mathcal{A}$ is non-cacheable such that $A_c \cup \mathcal{A}_{nc} = \mathcal{A}$. An illustration of the studied fog network model is shown in Fig. 1.

A. Computing Model

The computation of each task $a \in \mathcal{A}$ by UN u can be either performed locally or offloaded to a cloudlet. The total local computing time is:

$$D_{ua}^l(t) = \frac{\kappa L_a}{c_{\text{local}}} + W_{ua}^l(t) + \tau_{\text{LP}}, \tag{1}$$

where $W_{ua}^l(t)$ is the queuing delay of task a in the local queue of UN u at time t, c_{local} is the local computing capability in cycles/second, and τ_{LP} is the local processing delay.

Each task a requested by user u and offloaded to the cloudlet e experiences a total computing delay that consists of the task data transmission time, cloudlet computing time, cloudlet queuing time, and processing time, as follows:

$$D_{ea}^f(t) = \left(\frac{\kappa L_a}{c_e} + \frac{L_a}{r_{ue}(t)} + W_{ea}^f(t) \right)(1 - y_{ea}(t)) + \tau_{\text{EP}}, \tag{2}$$

where $r_{ue} = \text{BW} \log_2 \left(1 + P_u h_{ue}/(N_o + I_e) \right)$ is the uplink data rate[2] from UN u to the cloudlet e, $W_{ea}^f(t)$ is the waiting time of task a due to the previous computing tasks in the queue Q_e of cloudlet e, $y_{ea}(t)$ is a binary variable that equals 1 when the computation result of task a is cached in cloudlet e, and τ_{EP} is the cloudlet latency which accounts for the downlink transmission of computed data and the cloudlet processing latency. Consequently. the delay incurred by the computation of a given task a will be:

$$D_a(t) = x_{ea}(t)D_{ea}^f(t) + \left(1 - \sum_{e \in \mathcal{E}} x_{ea}(t) \right)D_{ua}^l(t), \tag{3}$$

where x_{ea} is a binary variable that equals 1 if task a is distributed to cloudlet e.

---

[1]Path loss is used as a coverage metric such that UN's cloudlet list does not change frequently due to wireless channel dynamics.

[2]BW is the channel bandwidth, P_u is the transmit power of UN u, h_{ue} is the channel gain between UN u and cloudlet e, I_e is the interfering power from other UNs, and N_o is the noise power.

Similar to other works [9], we assume that the latency due to downlink transmission of computed data is negligible compared to the uplink task data offloading time and computing time, and, hence, it is not accounted for in the optimization problem. This assumption is due to the typically small size of computed data and the relatively high transmission power of cloudlet compared to end-user devices.

Our objective is to minimize the total task computing latency under reliability constraints, by efficiently distributing and proactively caching the results of computing tasks. The UN task distribution to cloudlets and task caching matrices are expressed as $\boldsymbol{X} = [x_{ea}]$ and $\boldsymbol{Y} = [y_{ea}]$, respectively. Reliability is modeled as a probabilistic constraint on the maximum offloaded computing delay. This optimization problem is:

$$\min_{\boldsymbol{X},\boldsymbol{Y}} \sum_{u \in \mathcal{U}} D_a(t) \tag{4a}$$

$$\Pr(D_{ea}^{\text{f}}(t) \geq D_{\text{th}}) \leq \epsilon, \ \forall e \in \mathcal{E}, \tag{4b}$$

$$\sum_{e \in \mathcal{E}} x_{ea}(t) \leq 1, \ \forall u \in \mathcal{U}, \tag{4c}$$

$$\sum_{a \in Q_e} x_{ea}(t) \leq 1, \ \forall e \in \mathcal{E}, \tag{4d}$$

$$\sum_{a \in \mathcal{A}} y_{ea}(t) \leq s_e, \ \forall e \in \mathcal{E}, \tag{4e}$$

where (4b) is a probabilistic delay constraint that ensures the latency is bounded by a threshold value D_{th} with a probability $1 - \epsilon$. Constraints (4c) and (4d) ensure the one-to-one correspondence of distributing new requests to cloudlets. (4e) limits the number of cached tasks to a maximum of s_e. The above problem is a combinatorial problem with a non-convex cost function and probabilistic constraints, for which finding an optimal solution is computationally complex [13]. The non-convexity is due to the service rate term in the delay equation which is function of the interference from other offloading UNs. To make the problem tractable, we use the Markov's inequality to convert the probabilistic constraint in (4b) to a linear constraint [13] expressed as $\mathbb{E}\{D_{ea}^{\text{f}}(t)\} \leq D_{\text{th}}\epsilon$, where $\mathbb{E}\{.\}$ denotes the expectation over time. Since the delay of computing a cached task is very small, we are interested in keeping the delay of non-cached tasks below a pre-defined threshold. Hence, the constraint can be written as:

$$\mathbb{E}\left\{\frac{\kappa L_a}{c_e} + \frac{L_a}{r_{ue}(t)} + W_{ea}^{\text{f}}(t) + \tau_{\text{EP}}\right\} \leq D_{\text{th}}\epsilon, \tag{5}$$

substituting the queuing time as $W_{ea}^{\text{f}}(t) = \sum_{a_i \in Q_e} \frac{L'_{a_i}(t)}{r_{ie}(t)}$:

$$\mathbb{E}\left\{\frac{L_a}{r_{ue}(t)}\right\} \leq D_{\text{th}}\epsilon - \mathbb{E}\left\{\sum_{a_i \in Q_e} \frac{L'_{a_i}(t)}{r_{ie}(t)}\right\} - \frac{\kappa L_a}{c_e} - \tau_{\text{EP}}, \tag{6}$$

where $L'_{a_i}(t)$ is the remaining task data of task a_i in the queue Q_e of cloudlet e at time instant t. Finally, the constraint can be expressed as:

$$\frac{L_a}{\bar{r}_{ue}(t)} \leq D_{\text{th}}\epsilon - \frac{\kappa L_a}{c_e} - \sum_{a_i \in Q_e} \frac{L'_{a_i}(t)}{\bar{r}_{ie}(t)} - \tau_{\text{EP}}. \tag{7}$$

The above constraint implies that to reach the desired reliability, a maximum value of $\frac{L_a}{\bar{r}_{ue}(t)}$ is allowed for the newly

admitted requests to the queue of cloudlet e. The average service rate $\bar{r}_{ue}(t)$ is estimated at each cloudlet e for each UN u within its coverage using a time-average rate estimation method, as follows:

$$\bar{r}_{ue}(t) = \nu(t)r_{ue}(t - 1) + (1 - \nu(t))\bar{r}_{ue}(t - 1). \tag{8}$$

Next, we propose a joint matching [14] and caching scheme to solve the optimization problem in (4).

III. JOINT TASK MATCHING AND CACHING

To simplify the computational complexity of the optimization problem in (4), we decouple the problem into two separate subproblems: distributing UN tasks to cloudlets and caching popular cacheable task results. Due to the large size of IoT networks, it is not practical to perform task matching over the whole network set of cloudlets and UNs. Therefore, a clustering scheme is introduced to group UNs into disjoint sets based on spatial proximity and mutual interest in popular tasks, followed by the calculation of a task popularity matrix. Subsequently, a joint task distribution and caching scheme is proposed. UN clustering and task popularity matrix calculations are assumed to be performed during a network training period during which information about UNs' requests and their serving cloudlets are reported to a higher level controller, e.g. a cloud data center. While a central controller is involved in the training period calculations, we emphasize that this process does not need to be updated as frequently as the task distribution and caching processes, since a given user's interests are likely to remain unchanged for a number of time instants N_t ($\gg 1$).

A. Network Clustering and Task Popularity Matrix

We start by grouping UNs into k disjoint clusters $\mathcal{C}_1, \ldots, \mathcal{C}_k$ based on their mutual-coupling in distance and task popularity such that a task popularity matrix, defined as $\boldsymbol{\Xi} = [\boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_k]$ is calculated, where $\boldsymbol{\xi}_i$ is a vector of the popularity order of tasks in cluster $\mathcal{C}_i$. Essentially, identifying the similarities between neighboring UNs and their mutual interests is the first step in bringing computing resources closer to them. To that end, we exploit the similarity of different UNs in terms of their similar task popularity patterns to allow cloudlets in their proximity to store the computing results of their tasks.

1) Distance-based Gaussian similarity: The Gaussian similarity metric is used to quantify the similarity between UNs based on their inter-distance. A distance Gaussian similarity matrix is defined as $\boldsymbol{S}_d = [d_{ij}]$, with d_{ij} being:

$$d_{ij} = \exp\left(\frac{-\parallel \boldsymbol{v}_i - \boldsymbol{v}_j \parallel^2}{2\sigma_d^2}\right), \tag{9}$$

where $\boldsymbol{v}_i$ is a vector of the geographical coordinates of UN i, and σ_d is a similarity parameter to control the neighborhood size.

2) Task popularity-based similarity: To discover the task popularity patterns of different UNs, the task request occurrence is recorded for each UN during a training period set. Subsequently, a task occurrence vector $\boldsymbol{n}_u = [n_{u,1}, \ldots, n_{u,|\mathcal{A}_c|}]$ is calculated for each UN. This vector

Algorithm 1 UN clustering and popularity matrix calculation.

1: **Training phase:** For a sequence of training time instants:
- Record $\boldsymbol{n}_u$ of each UN.
- Calculate the similarity matrix $\boldsymbol{S}$ from (11).
- Set $k_{\min} = 2$ and $k_{\max} = U/2$.
- Record the number of times a cloudlet served each UN.

2: **Clustering phase:**
- Perform spectral clustering using the similarity matrix $\boldsymbol{S}$, use the largest eigenvalue gap method [15] to select the number of clusters $k \in \{k_{\min}, \ldots, k_{\max}\}$.
- Obtain k disjoint clusters of UNs $\mathcal{C}_1, \ldots, \mathcal{C}_k$.

3: **Popularity list construction phase:**
- Mark a cloudlet most preferred cluster as the cluster from which it received the highest number of requests during the training period.
- Calculate the task popularity matrix $\boldsymbol{\Xi}$ of each cluster using the number of request occurrences $\boldsymbol{n}_u$ of its set of UNs.
- Report to each cloudlet the task popularity vector $\boldsymbol{\xi}_i$ of its most preferred cluster $\mathcal{C}_i$.

---

captures the UN's task arrival rate and helps to build similarity between UNs. A cosine similarity metric is considered to measure the similarity between UNs. The task popularity similarity matrix is $\boldsymbol{S}_p = [p_{ij}]$, where p_{ij} is expressed as:

$$p_{ij} = \frac{\boldsymbol{n}_i . \boldsymbol{n}_j}{\| \boldsymbol{n}_i \| \| \boldsymbol{n}_j \|}. \tag{10}$$

3) UN clustering and popularity matrix calculation: Since we are interested in groups of UNs that are close to each other and having similar task popularity patterns, we consider a similarity matrix that blends the distance and task popularity matrices together. The similarity matrix S is calculated as:

$$\boldsymbol{S} = \theta \boldsymbol{S}_d + (1 - \theta) \boldsymbol{S}_p, \tag{11}$$

where θ is a parameter that adjusts the impact of distance and task popularity. Subsequently, we use spectral clustering [15] to group UNs into k disjoint clusters, $\mathcal{C}_1, \ldots, \mathcal{C}_k$.

To bring the popular tasks closer to the network edge, the task popularity matrix of UN clusters is reported to cloudlets so that they cache the computing result of the most popular tasks. Accordingly, the most preferred cluster by a cloudlet is obtained by calculating how frequently the members of each cluster were assigned to this specific cloudlet during the training period. The vector $\boldsymbol{\xi}_i$ of tasks that are most popular for a cluster i is reported to the cloudlets that have cluster $\mathcal{C}_i$ as their most preferred cluster. The proposed UN clustering and task popularity matrix calculation is described in Algorithm 1.

B. Computing Caching Scheme

During network operation, cloudlets seek to minimize the service delay of their UNs' requests by proactively caching the computing results of the popular tasks they receive. The caches of each cloudlet are assumed to be empty at the beginning of the network operation. As UNs start to offload computing tasks, cloudlets will cache as many computing results as their storage capacity allows. Once a cloudlet's storage is full, a new arriving request that is more popular than the least popular task currently in the cache will replace it. The algorithm implementation per cloudlet is described in Algorithm 2.

Next, if the cloudlet receives a computation request of a task that is cached in its storage, there is no need to offload the task data or recompute the task, and only processing delay is incurred. Each cloudlet aims to find the optimal caching policy that minimizes the total latency.

Algorithm 2 Proactive task caching algorithm.

1: **Initialization:**
- Define the set Ψ_e as the cache content of cloudlet e.
- $\Psi_e = \phi, \quad \forall e \in \mathcal{E}$.

2: **foreach** $a \in Q_e$
3:   **if** $| \Psi_e | < s_e$
4:     $a \to \Psi_e$.
5:   **else if** $| \Psi_e | = s_e$
6:     **if** there exists at least one task $a_i \in \Psi_e$ with lower index than a in ξ_e
7:       task a_i is removed from Ψ_e.
8:       $a \to \Psi_e$.
9:     **else**
10:       the computing result of task a is not stored.
11:     **end if**
12:   **end if**
13: **end foreach**

C. UN Task Distribution

Our next step is to propose a task distribution scheme that solves the constrained minimization problem in (4). The task distribution problem is formulated as a matching game between UNs and cloudlets where, at each time instant, UNs requesting new tasks are matched to a serving cloudlet aiming to minimize their service delay. Matching theory [14] is a framework that solves combinatorial problems in which members of two sets of players are interested in forming *matching pair*s with a player from the opposite set. Preferences of both the cloudlets and UNs, denoted $\succ_e$ and $\succ_u$, represent how each player ranks the players of the opposite set.

Definition 1. Given the two disjoint sets of cloudlets and UNs $(\mathcal{E}, \mathcal{U})$, a *matching* is defined as a *one-to-one* mapping Υ from the set $\mathcal{E} \cup \mathcal{U}$ into the set of all subsets of $\mathcal{E} \cup \mathcal{U}$, such that for each $e \in \mathcal{E}$ and $u \in \mathcal{U}$:
1) For each $u \in \mathcal{U}, \Upsilon(u) \in \mathcal{E} \cup u$, where $\Upsilon(u) = u$ means that a UN is not matched to a cloudlet, but will perform local computing instead.
2) For each $e \in \mathcal{E}, \Upsilon(e) \in \mathcal{U} \cup \{e\}$, where $\Upsilon(e) = e$ means that no UN is assigned to the cloudlet e.
3) $| \Upsilon(u) |= 1, | \Upsilon(e) |= 1$;   4) $\Upsilon(u) = e \Leftrightarrow \Upsilon(e) = u$.

By inspecting the problem in (4), we can see that the constraints (4c)-(4d) are satisfied by the one-to-one mapping of the matching game. Moreover, matching allows defining preference profiles that capture the cost function of the players. To this end, the preference profiles of UNs are defined so as to minimize their task service delay as follows:

$$e \succ_u e' \Leftrightarrow D_{ea}^{\mathrm{f}}(t) < D_{e'a}^{\mathrm{f}}(t), \tag{12}$$

$$u \succ_u e \Leftrightarrow D_{ua}^{\mathrm{l}}(t) < D_{ea}^{\mathrm{f}}(t). \tag{13}$$

Note that since a UN has no information about the queue length at each cloudlet, it considers the transmission, computing and processing delay of its own task data in calculating its preference profile.

The utility of cloudlets will essentially reflect the latency and reliability constraint in (7), taking into account the waiting time in the queue. Therefore, we define the utility when UN u is assigned to cloudlet e as:

$$\Phi_{eu}(t) = D_{th}\epsilon - \frac{k_a L_a}{c_e} - \sum_{a_i \in Q_e} \frac{L'_{a_i}(t)}{\bar{r}_{ie}(t)} - \tau_{\mathrm{EP}} - \frac{L_a}{\bar{r}_{ue}(t)}. \tag{14}$$

Algorithm 3 DA algorithm for UN-cloudlet matching.

1: **Initialization:** all UNs and cloudlets start unmatched.
2: Each UN constructs its preference list as per (12)-(13).
3: Each cloudlet constructs its preference list as per (15)-(16).
4: **repeat** an unmatched UN u, i.e., $\Upsilon(u) = \phi$ proposes to its most preferred cloudlets e that satisfies $e \succ_u u$.
5:   **if** $\Upsilon(e) = \phi$,
6:     UN u proposal is accepted.
7:     $\Upsilon(e) = u$, $\Upsilon(u) = e$.
8:   **elseif** $\Upsilon(e) = u'$,
9:     **if** $u' \succ_e u$
10:        UN u proposal is rejected.
11:        UN u removes cloudlet e from its preference list.
12:      **elseif** $u \succ_e u'$
13:        UN u proposal is accepted.
14:        $\Upsilon(e) = u, \Upsilon(u) = e$.
15:        $\Upsilon(u') = \phi$.
16:        UN u' removes cloudlet e from its preference list.
17:      **end if**
18:   **end if**
19: **until** all UNs are either matched or not having cloudlets that satisfy $e \succ_u u$. in their preference lists.
20: $\Upsilon(u) = u$ for all remaining unmatched UNs.
21: **Output:** a stable matching Υ.

The preference of each cloudlet can be expressed as follows:

$$u \succ_e u' \Leftrightarrow \Phi_{eu}(t) > \Phi_{eu'}(t), \tag{15}$$

$$e \succ_e u \Leftrightarrow \Phi_{eu}(t) < 0, \tag{16}$$

where (16) states that a cloudlet is not interested in being matched to a UN that will violate its reliability constraint. In other words, the utility of each cloudlet is to seek a matching that maximizes the difference between the right hand side and the left hand side of the inequality in (7), such that the constraint is met as a stable matching is reached.

The above problem is a one-to-one matching game. Next, we define matching stability and provide an efficient algorithm based on deferred acceptance (DA) [14] to solve this game.

Definition 2. Given a matching Υ with $\Upsilon(e) = u$ and $\Upsilon(u) = e$, and a pair (u', e') with $\Upsilon(e) \neq u'$ and $\Upsilon(u) \neq e'$, (u', e') is said to be blocking the matching Υ and form a blocking pair if: 1) $u' \succ_e u$, 2) $e' \succ_u e$. A matching $\Upsilon*$ is stable if there is no blocking pair.

***Remark* 1**. DA algorithm described in Algorithm 3, converges to a two-sided stable matching of UNs to cloudlets [14].

IV. SIMULATION RESULTS

In this section, we present and illustrate insights from simulation results of the proposed scheme. We also compare the proposed proactive computing scheme against the following two baseline schemes:

1) *Baseline 1*, which is a reactive version of the proposed scheme, in which the latency and reliability constrained task distribution scheme is considered, but with no caching capabilities in the cloudlets.
2) *Baseline 2*, in which latency and reliability constraints are not considered. Instead, UNs and cloudlets rank each other based on the wireless access link quality, without taking delay queues or proactiveness into account.
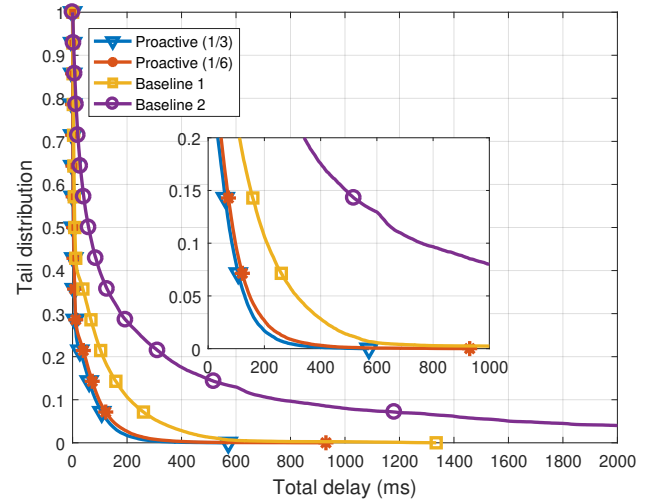

Figure 2. The total delay tail distribution for the proposed (proactiveness of $1/6$ and $1/3$ of cacheable contents) and the baseline schemes, with $E = 30$ cloudlets, $U = 3 \times 30$ UNs, and traffic intensity of 9 Mbps.

We use a set of default parameters[3] unless stated otherwise. Three different sets of task popularity distributions are assigned randomly to UNs, where popularity varies among tasks following the Zipf popularity model with parameter z [10]. Accordingly, the request rate for the ith most popular task is proportional to $1/i^z$. Furthermore, one third of the tasks, uniformly selected, are assumed to be cacheable.

A. Proactiveness and Computation Delay

In Fig. 2, we show the tail distribution of the instantaneous total computing delay, i.e., the complementary cumulative distribution function (CCDF) $\bar{F}_D(d) = \Pr(D > d)$, for different schemes. The proposed scheme is simulated for different proactiveness levels of $1/3$ and $1/6$. In other words, the cloudlet storage can store up to $1/3$ and $1/6$ of the computing results of the cacheable tasks. From Fig. 2, we can see that the proposed scheme maintains a 99% reliability constraint ($\epsilon = 0.01$) for both the proactive and the reactive cases. Moreover, the probability of having higher delay values significantly decreases as the proactiveness level increases since storing more computing results closer to UNs will further reduce the computing delay.

The average total delay performance is presented in Fig. 3 under different proactiveness levels. Comparing Baseline 1 and Baseline 2 schemes, about 72% decrease in the average computing delay can be seen. In Baseline 2 scheme, requests that will violate the latency constraints are not admitted, and are computed locally instead. Furthermore, the proactive scheme significantly decreases the computing delay as the proactiveness level increases. By storing more computing results close to UNs, up to 91% decrease in delay is observed.

The impact of proactiveness on the delay and cache hit rate is investigated in Fig. 4 for different discrepancy levels

---

[3]$E = 30$ cloudlets, $U = 3 \times 30$ UNs, $\mid A \mid = 90$ tasks, $z = 0.6$, $D_{\text{th}} = 1$s, $\epsilon = 0.01$, UN power = 20 dBm, $\theta = 0.5$, $\sigma_d^2 = 500$, $\nu(t) = 1/t^{0.55}$, $\kappa/c_{\text{local}} = 10^{-7}, \kappa/c_e = 10^{-8}$, $s_e = 10$ tasks, $\tau_{\text{LP}} = \text{Unif}(0, \frac{1}{8})$ ms, $\tau_{\text{EP}} = \text{Unif}(\frac{1}{8}, \frac{1}{4})$ ms.
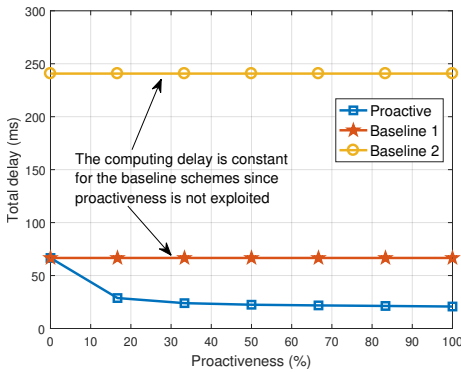
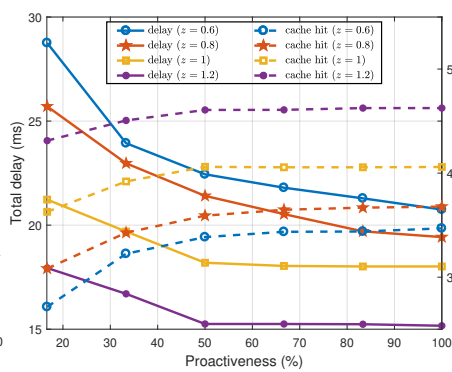Figure 3. Total delay performance at different proactiveness levels

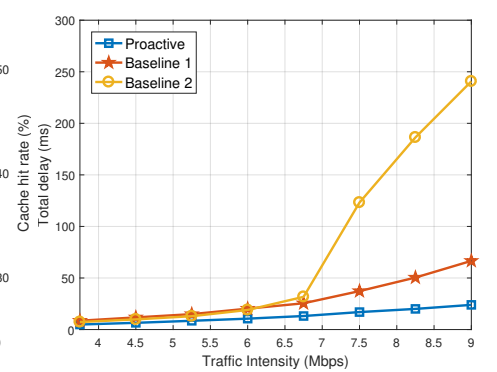Figure 4. Total delay (solid lines) and cache hit rate (dashed lines) at different values of z

Figure 5. Total delay performance as the arrival traffic intensity increases

of the task popularity distributions, represented by the Zipf parameter z. As z increases, the popularity gap between the most and least popular tasks increases. At high values of z, most of the computing requests are for the most popular tasks. Accordingly, it is possible to serve more requests from the cache, resulting in low computing delay and high cache hit rate, even at low proactiveness levels. On the other hand, at low values of z, high proactiveness levels are needed to store the most popular task results. Therefore, a steep decrease in computing delay and an increase in the cache hit rate are observed as proactiveness level increases.

B. Impact of Traffic Intensity

Next, we study how the performance changes with the traffic intensity. Intuitively, at low traffic intensity conditions, cloudlets can cope with the computing requests with minimal latency, and there is no need to assign requests to local computing. However, at high traffic conditions, offloading all requests causes the cloudlet queues to grow rapidly, unless stringent latency requirements are imposed. From Fig. 5, we can see that as the traffic intensity increases, there exists a threshold point in which higher traffic intensity will cause severe delay for the baseline scheme with unbounded latency. Below this point, Baseline 1 scheme achieves similar or lower delay values than Baseline 2 as there is no compelling need to maintain latency bounds by assigning requests to local computing. Moreover, both the reactive and proactive schemes achieve low delay performance, with proactiveness gains of up to 65% at high traffic intensity.

V. Conclusions

In this paper, we have proposed a task distribution and proactive computing scheme for cache-enabled fog computing networks under ultra-reliability and low-latency constraints. In the proposed scheme, clusters of cloudlets and edge user nodes are formed based on spatial proximity and similar interests in computing results. To ensure minimal computing delays, each cluster proactively caches computing results in the storage of its cloudlets. Moreover, we have proposed a matching algorithm to distribute the computing tasks to cloudlets such that computing delay is minimized and latency constraints are met. Simulation results have shown that the proposed scheme significantly minimizes the computing delay under different proactiveness and traffic intensity levels, and is able to guarantee minimal latency bounds with high levels of certainty.

References

[1] Z. Dawy, W. Saad, A. Ghosh, J. G. Andrews, and E. Yaacoub, "Toward massive machine type cellular communications," *IEEE Wireless Commun.*, vol. 24, no. 1, pp. 120–128, February 2017.

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proc. the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12, 2012, pp. 13–16.

[3] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "Mobile edge computing: Survey and research outlook," *ArXiv e-prints*, 2017.

[4] S. Barbarossa, S. Sardellitti, and P. D. Lorenzo, "Joint allocation of computation and communication resources in multiuser mobile cloud computing," in *Proc. IEEE 14th Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, June 2013, pp. 26–30.

[5] Y. Mao, J. Zhang, S. H. Song, and K. B. Letaief, "Power-delay tradeoff in multi-user mobile-edge computing systems," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec 2016, pp. 1–6.

[6] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, Oct 2016.

[7] G. Lee, W. Saad, and M. Bennis, "An online secretary framework for fog network formation with minimal latency," in *Proc. IEEE Int. Conf. on Commun. (ICC)*, May 2017, pp. 1–6.

[8] E. Baştuğ, M. Bennis, M. Médard, and M. Debbah, "Towards interconnected virtual reality: Opportunities, challenges and enablers," *IEEE Commun. Mag.*, to be published, 2017.

[9] S. W. Ko, K. Huang, S. L. Kim, and H. Chae, "Live prefetching for mobile computation offloading," *IEEE Trans. Wireless Commun.*, vol. PP, no. 99, pp. 1–1, 2017.

[10] E. Baştuğ, 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.

[11] M. S. ElBamby, M. Bennis, W. Saad, and M. Latva-aho, "Content-aware user clustering and caching in wireless small cell networks," in *Proc. 11th Intl. Symp. on Wireless Communications Systems (ISWCS)*, Aug 2014, pp. 945–949.

[12] A. Anpalagan, M. Bennis, and R. Vannithamby, *Design and Deployment of Small Cell Networks*. Cambridge University Press, 2015.

[13] A. Mukherjee, "Queue-aware dynamic on/off switching of small cells in dense heterogeneous networks," in *IEEE Globecom Workshops (GC Wkshps)*, Dec 2013, pp. 182–187.

[14] Y. Gu, W. Saad, M. Bennis, M. Debbah, and Z. Han, "Matching theory for future wireless networks: fundamentals and applications," *IEEE Commun. Mag.*, vol. 53, no. 5, pp. 52–59, May 2015.

[15] J. Cranshaw, R. Schwartz, J. I. Hong, and N. Sadeh, "The livehoods project: Utilizing social media to understand the dynamics of a city," in *Proc. International AAAI Conference on Weblogs and Social Media*, 2012, p. 58.