Elbamby *et al. EURASIP Journal on Wireless Communications and Networking* (2018) 2018:209
https://doi.org/10.1186/s13638-018-1218-y

EURASIP Journal on Wireless Communications and Networking

RESEARCH Open Access

Proactive edge computing in fog networks with latency and reliability guarantees

Mohammed S. Elbamby[1*], Mehdi Bennis[1,3], Walid Saad[2], Matti Latva-aho[1] and Choong Seon Hong[3]

Abstract

This paper studies the problem of task distribution and proactive edge caching in fog networks with latency and reliability constraints. In the proposed approach, user nodes (UNs) offload their computing tasks to edge computing servers (cloudlets). Cloudlets leverage their computing and storage capabilities to proactively compute and store *cacheable* computing results. In this regard, a task popularity estimation and caching policy schemes are proposed. Furthermore, the problem of UNs' tasks distribution to cloudlets is modeled as a one-to-one matching game. In this game, UNs whose requests exceed a delay threshold use the notion of hedged-requests to enqueue their request in another cloudlet, and offload the task data to whichever is available first. A matching algorithm based on the deferred-acceptance matching is used to solve this game. Simulation results show that the proposed approach guarantees reliable service and minimal latency, reaching up to 50 and 65% reduction in the average delay and the 99th percentile delay, as compared to reactive baseline schemes.

Keywords: 5G, Caching, Fog networks, IoT, Hedged requests, Matching theory, Offloading, Resource allocation

1 Introduction

Evolving wireless networks to their next generation is no longer limited to boosting network capacity in the context of enhanced mobile broadband (eMBB). Instead, 5G networks aim to provide low-latency and high-reliability guarantees in addition to higher spectral efficiency. This essentially include the support for massive machine-type communication and ultra-reliable and low-latency communication (URLLC).

The emergence of Internet of things (IoT), which requires massive connectivity of resource-limited devices, poses unprecedented challenges in terms of required amount of data communication and computation [1, 2]. These stringent requirements in end-to-end latency, service reliability, and availability mandate a system design which incorporates latency and reliability aspects. Due to the limited computation and energy capabilities of IoT devices, it is hard to rely on local computing resources in satisfying stringent computing and processing latency requirements. Therefore, mobile cloud computing (MCC) solutions have been considered to offer computing services to IoT network nodes. In this regard, user nodes (UNs) offload their computing tasks to remote cloud centers, the cloud servers execute the task and send back the results . While this solution offers high resource capacity, it falls short of handling latency-critical computing services, owing to the high propagation delays between the UNs and the cloud data center. For this reason, the concept of fog computing has been introduced [3] to bring computing resources to the network edge. With short propagation delays, fog networks are able to strike a balance between high computation capabilities and short propagation distances, offering efficient computing services to latency-intolerant applications. However, smarter resource utilization schemes that efficiently distribute the communication and computing resources are needed to reap the benefits of fog networks [4, 5].

Several recent works have studied the problem of joint computing and communication resource optimization. A centralized joint communication and computation resource allocation scheme is proposed in [6]. The work

*Correspondence: mohammed.elbamby@oulu.fi
[1]Centre for Wireless Communications (CWC), University of Oulu, Oulu, Finland
Full list of author information is available at the end of the article

in [7] studies the resource allocation in mobile edge computing (MEC) networks for energy efficiency. The power-delay tradeoff in centralized MEC systems is discussed in [8] using tools from stochastic optimization. In [9], the problem of latency minimization under a resource utilization constraint is studied, where an algorithm is proposed to balance between minimizing computing latency and energy consumption. However, these works rely on centralized solutions in which the MEC network has information about all users requests and channel-state information (CSI). A semi-distributed solution for the joint computing and communication resource optimization is proposed in [10] under asymmetric user preferences and capabilities. A distributed framework to realize the power-delay tradeoff in latency constrained fog networks is proposed in [11]. Game-theoretic solutions are discussed in [12] to design decentralized computing offloading schemes for edge cloud computing. Game theory is also used in [13] to model the competition of IoT nodes over the fog network resources with the aim of minimizing both energy and delay. The fog network formation under the uncertainty of arriving and departing nodes is investigated in [5, 14, 15] with the aim of minimizing the maximum computing delay.

The vast majority of the literature in fog networking considers the reactive computing paradigm in which task computing starts only after the task data is offloaded to the fog node [4]. Moreover, most of the 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 [4]. This allows for harnessing the correlation between end-user requests through *proactive computing* to minimize computing latency [16]. The idea of *prefetching* is recently introduced as a first step towards proactive computing [17, 18], 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. Using proactive computing, the fog network can keep track of the popularity patterns of UNs' 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 [4, 19]. Caching computing results is applicable to several applications. For example, in an augmented reality (AR) service provided in a museum, edge servers can store popular AR requests of visitors to serve them with minimal latency [4]. In the context of MEC for vehicle navigation applications [20], correlation between computing tasks exists as multiple vehicles may request to detect the same object in a road. In such situations, the fog network can use proactive computing by keeping track of the popularity patterns of UNs' 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

The idea of proactive networks has been extensively studied in the context of wireless content caching [21, 22]. Proactive caching of computing tasks differs from content caching in several aspects. First, while content caching aims to alleviate the burden on the backhaul links by prefetching popular contents from the core network during off-peak times, computing caching decreases the load on the access link by providing computing results to end users without the need to prefetch their task data beforehand. Second, unlike content, computing tasks are of diverse types depending on the computing environment. Some computing tasks, such as object detection results, are *cacheable*, since they can be reused by other devices or in other time instants. On the contrary, computing data that are personalized or time dependent are not cacheable. Instead, they have to be computed in real time. Moreover, future computing networks will have dense deployments of servers and low density of UNs per server [23], which makes it impractical to follow the popularity patterns locally at each server. Alternatively, relying on central computing, either fully or partially, to build the popularity distributions over larger sets of servers is essential in providing a broader view on the popularity patterns of computing tasks.

The main contribution of this paper is to investigate the problem of URLLC in fog networks with proactive edge caching of computing results. Reliability is ensured by allowing UNs to offload their computing tasks to multiple edge computing nodes (cloudlets) based on the notion of *hedged requests* [24]. Furthermore, both computing and storage resources are exploited to minimize the computing latency via joint task offloading and proactive caching of *popular* and *cacheable* computing tasks. In the proposed framework, UNs and their serving cloudlets are grouped into well-chosen clusters based on spatial proximity and mutual interests in popular tasks. This allows cloudlets to proactively cache computing results of popular tasks in their clusters to ensure minimal computing 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 solve the problem of distributing UN requests to cloudlets, reaching a stable matching such that a reliable service in terms of maximum latency violation rate is guaranteed. Simulation results show that the proposed framework can guarantee reliable computations with bounded latency in different network density and storage capability conditions. Proactiveness is shown to minimize up to 50% and 65% of the average and the 99th percentile delay, respectively. Moreover, hedged

requests are shown to significantly improve the system reliability even without proactive capability.

The rest of this paper is organized as follows. Section 2 describes the system model and problem formulation. The clustering scheme as well as the joint caching and task distribution schemes are studied in Section 3. The performance of the proposed scheme is analyzed in Section 4. Finally, Section 5 concludes the paper.

2 System model

Consider a fog network consisting of a set $\mathcal{E}$ of E cloudlets and a set $\mathcal{U}$ of U UNs, which are uniformly distributed over the network area. Each cloudlet has a CPU computing capability of c_e cycles/s and a storage of size s_e. Cloudlets share the same frequency band with bandwidth B, and schedule UNs' transmissions in orthogonal time instants (slots). Here, our optimization framework focuses only on the uplink transmissions [4, 17]. 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. Tasks have a task data size L_a that is drawn from an exponential distribution of mean $\bar{L}_a$ and processing density of κ cycles per bit of task data. A UN requesting a computing task is matched to one or more cloudlets within its coverage, where coverage is decided based on a threshold path loss value. Path loss is used as a coverage metric so that a UN's cloudlet list does not change frequently due to wireless channel dynamics. The task data is then offloaded to the one that schedules the UN request first. Task computing is performed after the task data is offloaded. Subsequently, the computed data is transmitted back to the UN. Due to the typically small computed data size to minimize the task data offloading delay, cloudlets proactively cache the pre-computed results of the most popular cacheable computing tasks. A subset $\mathcal{A}_c \subset \mathcal{A}$ of the tasks is assumed to be cacheable and another subset $\mathcal{A}_{nc} \subset \mathcal{A}$ is non-cacheable such that $A_c \cup \mathcal{A}_{nc} = \mathcal{A}$. The notations and abbreviations used throughout this paper are summarized in Table 1. An illustration of the studied fog network model is shown in Fig. 1.

2.1 Computing model

The computation of a task $a \in \mathcal{A}$ by UN u offloaded to cloudlet e experiences a transmission delay D_{ea}^{comm} and a computing delay D_{ea}^{comp}. The transmission delay at time instant t is expressed as

$$D_{ea}^{\text{comm}}(t) = \frac{L_a(t)}{r_{ue}(t)}, \tag{1}$$

where $r_{ue}(t)$ is the uplink data rate per time slot, calculated using the Shannon's formula as follows:

$$r_{ue}(t) = B \log_2 \left(1 + \frac{P_u |h_{ue}|^2}{\sigma^2 + \sum_{v \in \mathcal{U} \setminus \{u\}} P_v |h_{ve}|^2}\right), \tag{2}$$

with $|h_{ue}|$ being the channel gain between UN u and cloudlet e, and σ^2 being the noise variance, where the channel h_{ue} includes path loss and block fading.

Table 1 List of abbreviations and symbols

CCDF	Complementary cumulative distribution function
CSI	Channel state information
DA	Deferred acceptance
MCC	Mobile cloud computing
MEC	Mobile edge computing
QoS	Quality of service
TDD	Time division duplex
UN	User node
URLLC	Ultra-reliable and low latency communication
$\mathcal{A}$	Set of tasks
$\mathcal{C}_i$	Set of UNs in cluster i
$\mathcal{E}$	Set of edge cloudlets
$\mathcal{U}$	Set of UNs
B	System bandwidth
c_e	Computing power of cloudlet e
D_{ea}	Total delay of task a from cloudlet e
D_{ea}^{comp}	Computing delay of task a from cloudlet e
D_{ea}^{comm}	Transmission delay of task a from cloudlet e
D_{th}	Delay threshold
L_a	Size of task a
$\mathbf{n_u}$	Task occurrence vector of UN u
Q_e	Task queue of cloudlet e
r_{ue}	Data rate of UN u from cloudlet e
$\bar{r}_{ue}$	Estimated data rate of UN u from cloudlet e
S_d	Distance Gaussian similarity matrix
S_p	Task popularity similarity matrix
s_e	Storage size of cloudlet e
$\boldsymbol{\nu}_u$	Geographical coordinates vector of UN u
W_{ea}	Waiting time of task a in the queue of cloudlet e
x_{ea}	Association indicator
X_{max}	Maximum number of associated cloudlets
y_{ea}	Caching indicator
z	Zipf distribution parameter
ϵ	Target maximum delay violation
σ_d	Similarity parameter
κ	Task processing density
λ_u	Mean UN task arrival rate
θ	Clustering parameter
τ_{EP}	Edge processing delay
ν	Learning rate parameter
$\boldsymbol{\xi}_i$	Popularity task vector of cluster i

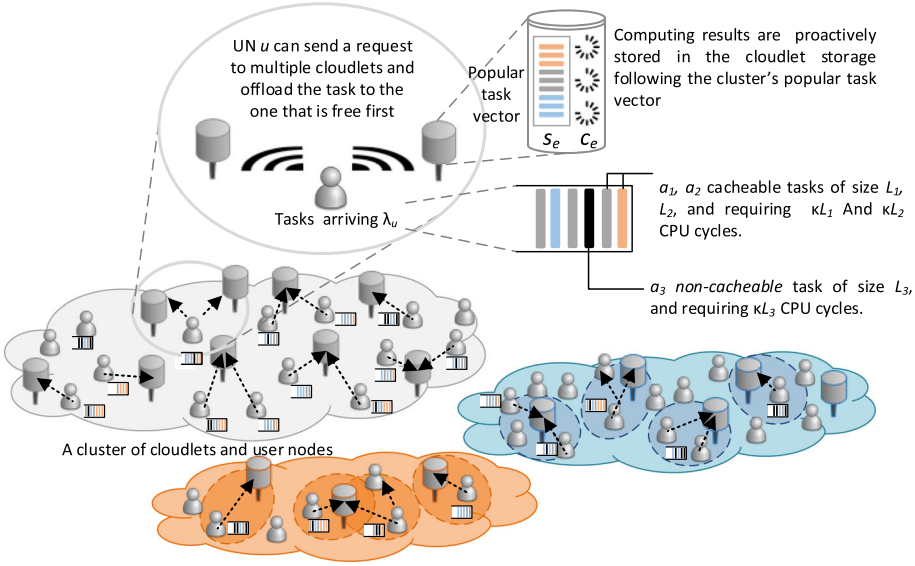

Fig. 1 Network illustration. An illustration of the clustered fog network model with cache-enabled cloudlets

After the task data is uploaded, a computing delay is D_{ea}^{comp} is experienced, expressed as:

$$D_{ea}^{\text{comp}}(t) = \frac{\kappa L_a(t)}{c_e}, \tag{3}$$

Adding the cloudlet queuing time, and processing time, the total delay becomes

$$\begin{aligned} D_{ea}(t) &= x_{ea}(t)\left[\left(D_{ea}^{\text{comp}}(t)+D_{ea}^{\text{comm}}(t)+W_{ea}(t)\right)\left(1-y_{ea}(t)\right)+\tau_{\text{EP}}\right], \\ &= x_{ea}(t)\left[\left(\frac{\kappa L_a(t)}{c_e}+\frac{L_a(t)}{r_{ue}(t)}+W_{ea}(t)\right)\left(1-y_{ea}(t)\right)+\tau_{\text{EP}}\right], \end{aligned} \tag{4}$$

where x_{ea} is a binary variable that equals 1 if task a is distributed to cloudlet e, $W_{ea}(t)$ 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. In other words, only the clouldlet latency is experienced if the requested task result was cached in the cloudlet. Similar to other works [4, 17], 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; hence, it is taken into account only as part of the cloudlet delay.

To overcome the large delay values due to long queues in the cloudlet's side, UNs leverage hedged requests to offload the task to multiple cloudlet if the estimated delay time exceeds a certain threshold. The UN will upload the task data to the cloudlet that frees its queue first, i.e., the one with the smallest waiting time w_{ea}, and discard the request to the other one. Consequently, the delay experienced to compute the task from a hedged request, denoted $D_a(t)$, is the delay value from the cloudlet with the smallest waiting time, which is expressed as

$$D_a(t) = \sum_{e\in\mathcal{E}|x_{ea}=1} D_{ea}(t) \prod_{e'\in\mathcal{E}|x_{e'a}=1, e'\neq e} \mathbb{1}_{w_{ea}<w_{e'a}}. \tag{5}$$

Our objective is to minimize the total task computing latency under reliability constraints, by efficiently distributing and proactively caching the results of the 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

$$\min_{\boldsymbol{X},\boldsymbol{Y}} \sum_{a\in\mathcal{A}} D_a(t) \tag{6a}$$

$$\Pr(D_a(t) \geq D_{\text{th}}) \leq \epsilon, \tag{6b}$$

$$\sum_{e\in\mathcal{E}} x_{ea}(t) \leq X_{\max}, \ \forall u \in \mathcal{U}, \tag{6c}$$

$$\sum_{a\in\mathcal{A}} y_{ea}(t) \leq s_e, \ \forall e \in \mathcal{E}, \tag{6d}$$

where (6b) is a probabilistic delay constraint that ensures the latency is bounded by a threshold value D_{th} with a probability $1-\epsilon$. Constraint (6c) indicates that a request is offloaded to a maximum of $X_{\max}$ cloudlets. (6d) 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 [25]. The non-convexity arises from the service rate expression in Eqs. (1)–(2) which is function of the offloading decisions of other UNs and the non-linear probabilistic constraint. Therefore, we replace the non-linear constraint in (6b) by a linear one. The Markov's inequality [25], for the non-negative random variable D_a and $D_{\text{th}} > 0$, states that $\mathbb{E}\{D_a(t)\} \leq D_{\text{th}}\epsilon$, 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. The constraint can be expressed for a single cloudlet as

$$\begin{aligned} \mathbb{E}\{D_{ea}(t)\} &\leq D_{\text{th}}\epsilon \\ \mathbb{E}\left\{\frac{\kappa L_a(t)}{c_e} + \frac{L_a(t)}{r_{ue}(t)} + W_{ea}(t) + \tau_{\text{EP}}\right\} &\leq D_{\text{th}}\epsilon, \end{aligned} \tag{7}$$

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

$$\mathbb{E}\left\{\frac{L_a(t)}{r_{ue}(t)}\right\} \leq D_{\text{th}}\epsilon - \mathbb{E}\left\{\sum_{a_i \in Q_e} \frac{L'_{a_i}(t)}{r_{ie}(t)}\right\} - \frac{\kappa L_a}{c_e} - \tau_{\text{EP}}, \tag{8}$$

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 URLLC constraint can be met by ensuring that the above inequality is satisfied at each time instant t, i.e.,

$$\frac{L_a(t)}{\bar{r}_{ue}(t)} \leq D_{\text{th}}\epsilon - \frac{\kappa L_a(t)}{c_e} - \sum_{a_i \in Q_e} \frac{L'_{a_i}(t)}{\bar{r}_{ie}(t)} - \tau_{\text{EP}}. \tag{9}$$

This implies that to reach the desired reliability, the above inequality has to hold for at least one cloudlet serving the request a of UN u, where $\bar{r}_{ue}(t)$ is the average service rate. Hence, if the constraint in (9) is not met for one cloudlet, UN can be matched to an additional cloudlet such that the reliability constraint is satisfied. The average service rate is estimated at each cloudlet e for each UN u within its coverage using a time-average rate estimation method, as follows:

$$\bar{r}_{ue}(t) = \nu(t) r_{ue}(t-1) + (1-\nu(t))\bar{r}_{ue}(t-1), \tag{10}$$

where $\nu(t)$ is a learning parameter.

Remark 1 *A learning parameter $\nu(t)$ where $\nu(t) \in (0,1]$ for any t such that $\lim_{t\to\infty} \nu(t) = 0$, ensures that $\bar{r}_{ue}(t)$ yields the time average rate of UN u at $t \to \infty$.*

In the following section, we propose a joint matching [26] and caching scheme to solve the optimization problem in (6).

3 The proposed method

In this section, we discuss our proposed approach to solve the optimization problem in (6). To simplify the computational complexity of the problem, we decouple it into two separate subproblems: task distribution to cloudlets and caching of popular cacheable task results. Since the IoT network size is typically large, adopting centralized optimization schemes over the entire cloudlet set is not practical. On the other hand, local schemes do not have enough local information about the popularity of computing tasks due to the low number of UNs per cloudlet. Therefore, our approach resorts to clustering tools to split the network into mutually coupled groups of UNs and their serving cloudlets. This clustering approach is carried out in a training period that can be repeated in a slow time scale, allowing for a broader view of the popularity patterns. In this regard, the proposed clustering scheme first groups UNs into disjoint sets based on both spatial proximity and mutual interest in popular tasks. Subsequently, a task *popularity matrix* is obtained. Finally, a joint task distribution and caching scheme is proposed. The UN clustering and the popularity matrix calculation are assumed to be carried out in the 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. During the training period, UNs are assigned to the cloudlets with the highest channel strength within their coverage area. 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 UN's interests are likely to remain unchanged for a number of time instants N_t ($\gg 1$).

3.1 Network clustering and task popularity matrix

The training phase starts by grouping UNs into k disjoint clusters $\mathcal{C}_1, \ldots, \mathcal{C}_k$ based on their mutual-coupling in distance and task popularity. The objective of clustering is to obtain a per-cluster task popularity matrix, defined as $\boldsymbol{\Xi} = \left[\boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_k\right]$, 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 harness the similarity between different UNs in terms of task popularity patterns to allow cloudlets in their proximity to store the computing results of their highly requested tasks. Next, we discuss how the distance similarity and task popularity similarity are calculated.

3.1.1 *Distance-based Gaussian similarity*

First, we introduce a distance similarity metric that measures the coupling between each pair of UNs based on their geographical locations. The Gaussian similarity

metric is commonly used in the literature to quantify the similarity between two nodes. A distance Gaussian similarity matrix is defined as $\boldsymbol{S}_d = \left[d_{ij}\right]$, with d_{ij} being

$$d_{ij} = \exp\left(\frac{- \parallel \boldsymbol{v}_i - \boldsymbol{v}_j \parallel^2}{2\sigma_d^2}\right), \tag{11}$$

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.

3.1.2 Task popularity-based similarity

To discover the task popularity patterns of different UNs, the task request occurrence is recorded for each UN during the training period set. Subsequently, a task occurrence vector, expressed as, $\boldsymbol{n}_u = \left[n_{u,1}, \ldots, n_{u,|\mathcal{A}_c|}\right]$ is calculated for each UN. Ideally, this vector captures the UN's task arrival rate and helps to build similarity between UNs. To calculate the task popularity-based similarity between UNs, we consider a cosine similarity metric. The cosine similarity is often used in vector-based attributes in which the similarity value corresponds to the angle between the two vectors, i.e., parallel vectors mean maximally similar, and perpendicular vectors mean maximally dissimilar. In other words, the cosine similarity takes into account only the vector direction and is agnostic to its magnitude. Hence, the task popularity similarity matrix is defined as $\boldsymbol{S}_p = \left[p_{ij}\right]$, where p_{ij} is expressed as

$$p_{ij} = \frac{\boldsymbol{n}_i . \boldsymbol{n}_j}{\parallel \boldsymbol{n}_i \parallel \parallel \boldsymbol{n}_j \parallel}. \tag{12}$$

3.1.3 UN clustering and popularity matrix calculation

A network cluster should essentially include UNs that are close to each other and following a similar task popularity patterns. Therefore, we consider a similarity matrix that blends together distance and task popularity matrices. The similarity matrix, denoted S, is calculated as

$$\boldsymbol{S} = \theta \boldsymbol{S}_d + (1 - \theta)\boldsymbol{S}_p, \tag{13}$$

where θ is a parameter that prioritizes the impact of distance and task popularity. Subsequently, spectral clustering [27] is used to group UNs into k disjoint clusters, denoted $\mathcal{C}_1, \ldots, \mathcal{C}_k$.

Remark 2 *Spectral clustering of U points requires the setting of a minimum and maximum number of clusters,* $k_{\min}$ *and* $k_{\max}$ *where* $k_{\max} \leq U$ [27]. *Here, we select* $k_{\min} = 2$ *in order to have at least two groups of UNs, and* $k_{\max} = U/2$ *since having a higher number of clusters will imply having multiple clusters with a single member, which contradicts with the idea of grouping.*

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.

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 (13).
- 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 $\boldsymbol{S}$ and using the largest eigenvalue gap method [27] 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:**

- Select each cloudlet's most preferred cluster as the cluster from which the cloudlet 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 $\mathcal{C}_i$.

3.2 Computing caching scheme

Following the training phase, cloudlets seek to minimize the service delay of their UNs' requests during the network operation by proactively caching the computing results of the popular tasks they receive. The caching process is performed in an online-manner, in a faster time scale than the clustering and popularity calculation phase. Each cloudlet aims to optimize its caching policy to minimize the total computing latency of its serving UNs. The cache storage of each cloudlet is assumed to be empty at the beginning of the network operation. As UNs start to offload their 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. This is equivalent to a least frequently used (LFU) replacement strategy in content caching. The algorithm implementation per cloudlet is described in Algorithm 2.

Algorithm 2 Proactive task caching algorithm

Initialization:

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

for each $a \in Q_e$
if $\mid \Psi_e \mid < s_e$
  $a \rightarrow \Psi_e$.
else if $\mid \Psi_e \mid = s_e$
  if there exists at least one task $a_i \in \Psi_e$ with lower index than a in ξ_e
    task a_i is removed from Ψ_e.
    $a \rightarrow \Psi_e$.
  else
    the computing result of task a is not stored.
  end **if**
end **if**
end **for each**

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. Hence, the computing delay of this task consists only of the cloudlet delay, which can be obtained from Eq. (4) by setting $(1 - y_{ea}(t)) = 0$ for task a cached in cloudlet e.

3.3 UN task distribution

Our next step is to propose a task distribution scheme that solves the constrained minimization problem in (6). 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 one or more serving cloudlets, depending on their latency requirements, such that their service delay is minimized. Matching theory [26] 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 are denoted as $\succ_e$ and $\succ_u$, and they represent how each player in one side ranks the players of the opposite side. From the optimization problem in (6), a UN seeks to be matched to more than one cloudlet if the first one is not satisfying its reliability metric, given the estimated rate $\bar{r}_{ue}(t)$. Therefore, the proposed matching algorithm first finds a one-to-one matching of UN requests to a single cloudlet each. The estimated delay of each request is then calculated. If a subset of the matched UNs' requests are not guaranteed to satisfy the reliability constraint, the matching algorithm is run again to match the UNs to additional cloudlets, subject to the availability of vacant cloudlets.

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 $u \in \mathcal{U}, \Upsilon(u) \in \mathcal{E}$.
2. For each $e \in \mathcal{E}, \Upsilon(e) \in \mathcal{U}$.
3. $\mid \Upsilon(u) \mid = 1, \mid \Upsilon(e) \mid = 1$.
4. $\Upsilon(u) = e \Leftrightarrow \Upsilon(e) = u$.

The selection of the preference profiles of matching allows for capturing 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:

$$e \succ_u e' \Leftrightarrow D_{ea}(t) < D_{e'a}(t). \tag{14}$$

This preference allows UNs to seek matchings that minimizes their own QoS by selecting the cloudlets that provides minimal computing latency. Note that due to the impracticality of having knowledge of the queue state of each cloudlet at each UN, UNs consider the transmission and computing delay of their own task data in calculating their preference profiles.

Cloudlets, having enough information about their own queue length, seek to maximize the service reliability of the requests of their serving UNs. Hence, the utility of cloudlets will essentially reflect the reliability constraint in (9), taking into account the waiting time in the queue. Therefore, we define the utility when UN u is assigned to cloudlet e as

$$\Phi_{eu}(t) = D_{th}\epsilon - \frac{k_a L_a}{c_e} - \sum_{a_i \in Q_e} \frac{L'_{a_i}(t)}{\bar{r}_{ie}(t)} - \tau_{\mathrm{EP}} - \frac{L_a}{\bar{r}_{ue}(t)}. \tag{15}$$

The preference of each cloudlet can be expressed as follows:

$$u \succ_e u' \Leftrightarrow \Phi_{eu}(t) > \Phi_{eu'}(t). \tag{16}$$

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 (9), such that the constraint is met as a stable matching is reached. Note from (9) that, to satisfy the reliability constraint, $\Phi_{eu}(t)$ has to be greater or equal to zero. Therefore, matched UNs with $\Phi_{eu}(t) < 0$ will seek to be matched to additional cloudlets to satisfy their reliability requirements.

Next, we define matching stability and provide an efficient algorithm based on the deferred acceptance (DA) [26] to solve this game.

Definition 2 *Given a matching* Υ*, 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)* $u' \succ_{e'} \Upsilon(e')$*, 2)* $e' \succ_{u'} \Upsilon(u')$*. A matching* Υ^* *is stable if there is no blocking pair.*

Remark 3 *The DA algorithm, described in Algorithm 3, converges to a two-sided stable matching of UNs to cloudlets* [26]*.*

The architecture of the proposed approach is illustrated in the chart in Fig. 2.

4 Simulation results

In this section, we present and analyze simulation results of the proposed scheme. To illustrate the performance of both the proposed hedged requests and proactive computing, we consider the following three baselines:

Algorithm 3 The DA algorithm for UN-cloudlet matching

Initialization: all UNs with new requests start unmatched.
Previously unmatched UNs with $\Phi_{eu}(t) < 0$ are marked unmatched to be associated to an additional cloudlet.
Each unmatched UN constructs its preference list as per (14).
Each cloudlet constructs its preference list as per (16).
repeat an unmatched UN u, i.e., $\Upsilon(u) = \phi$ proposes to its most preferred cloudlets e that satisfies $e \succ_u u$.
  if $\Upsilon(e) = \phi$,
    UN u proposal is accepted.
    $\Upsilon(e) = u, \Upsilon(u) = e$.
  else if $\Upsilon(e) = u'$,
    if $u' \succ_e u$
      UN u proposal is rejected.
      UN u removes cloudlet e from its preference list.
    else if $u \succ_e u'$
      UN u proposal is accepted.
      $\Upsilon(e) = u, \Upsilon(u) = e$.
      $\Upsilon(u') = \phi$.
      UN u' removes cloudlet e from its preference list.
    end if
  end if
until all UNs are matched.
Output: a stable matching Υ.

1. *Baseline 1 (BL1)*, which is similar to the proposed scheme, in which hedged requests are enabled to satisfy UNs' reliability. However, caching is performed by estimating the popularity distribution locally at each cloudlet and considering a least recently used (LRU) caching replacement strategy.
2. *Baseline 1 (BL2)*, which is a reactive version of the proposed scheme, in which hedged requests are enabled to satisfy UNs' reliability, but with no caching or proactive capabilities in the cloudlets.
3. *Baseline* 2 *(BL3)*, in which neither proactiveness nor hedged requests are considered. UNs and cloudlets rank each other based on the wireless access link quality, and a UN can only be matched to a single cloudlet.

We use the set of default parameters listed in Table 2 unless stated otherwise. Popularity is assumed to vary among tasks following a Zipf-like popularity model with parameter z [21]. Accordingly, the rate of requesting the ith most popular task is proportional to $1/i^z$. Three different sets of task popularity distributions are assigned randomly to UNs. Furthermore, one third of the tasks, uniformly selected, is assumed to be cacheable, whereas the remaining tasks have to be computed in real time[1].

4.1 Delay tail performance

First, we investigate the tail of the instantaneous delay of the different schemes as a measure of service reliability. The delay tail is represented by the complementary cumulative distribution function (CCDF) $\bar{F}_D(d) = \Pr(D > d)$. As shown in Fig. 3, the proposed scheme significantly minimizes the delay tail, as compared to the two reactive baseline schemes. In particular, the proposed proactive scheme achieves 40% and 57% gains in 90th percentile delay and 99th percentile delay, respectively, as compared to BL2 scheme. Moreover, the BL2 scheme, with no proactivity capability, outperforms BL3 in the delay tail performance. This is due to leveraging the hedged requests to minimize the latency of requests with long waiting time. It can also be seen that the caching-based BL1 scheme has a tail performance close to that of the proposed scheme, but with higher average delay.

4.2 Impact of network density

Next, we investigate the impact of the network density in the service delay performance. The network density is varied by changing the number of cloudlets distributed within the network area. The number of UNs is also changed to maintain the same UN density. The average delay performance is presented in Fig. 4a, whereas the 99th percentile delay is shown in Fig. 4b as a measure of the service reliability. The results show that, for the proposed approach, the service delay decreases as

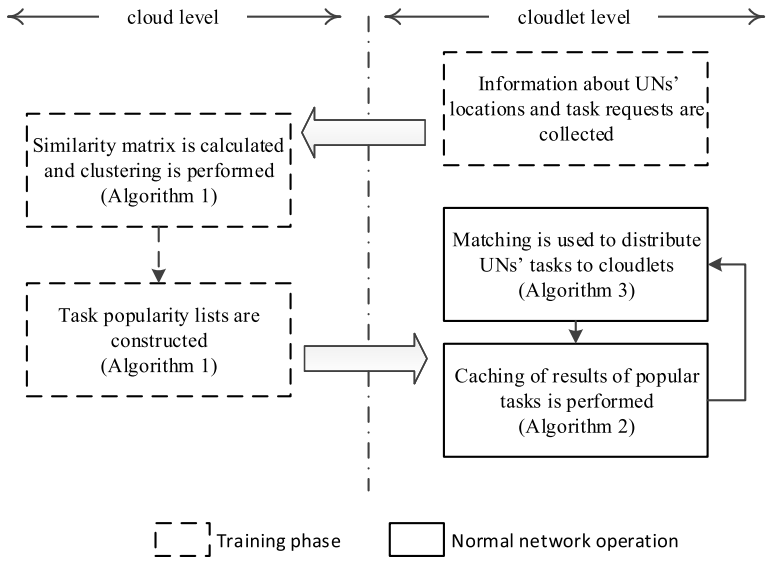

Fig. 2 The proposed approach architecture. The architecture of the proactive edge computing framework

the network density increases. This is due to the availability of more cloudlets in the coverage area of each UN, which increases both the availability of matching and the flexibility of matching. Moreover, due to serving cacheable requests directly without task data offloading, the increased network density does not incur high uplink interference levels to affect the delay. However, BL2 suffers from increased delay due to having higher interference in denser networks, which significantly increases the delay. This effect is slightly alleviated as the number of cloudlets further increases, due to high number of available cloudlets to each UN.

Table 2 Simulation parameters

Parameter	Value
System bandwidth	10 MHz
Number of cloudlets E	30
UN density U/E	3
Maximum associated cloudlets $X_{\max}$	2
Mean UN task arrival rate λ_u	10 task/s
Number of tasks $\mid A \mid$	100
Cloudlet storage size s_e	10 tasks
Mean task size $\bar{L}_a$	200 Kbit
Processing density[1] κ	162.5 cycle/bit [28]
Cloudlet computing power c_e	10^{10} cycle/s [29]
Zipf parameter z	0.6
Delay threshold D_{th}	1 s
Target delay violation ϵ	0.01
Clustering parameter θ	0.5
Processing delay τ_{EP}	$Unif\left(\frac{1}{8}, \frac{1}{4}\right)$ ms
Neighborhood size parameter σ_d^2	500 m
Scheduling slot	1 ms
Network size	$500 \times 500\ m^2$
UN transmit power	10 dBm
learning parameter $\nu(t)$	$1/t^{0.51}$

Comparing BL2 and BL3 schemes, we can clearly see the trade-off between ensuring a reliable service and maintaining a low delay. At low network density conditions, there are not enough available cloudlets for each UN. Accordingly, boosting reliability through hedged-requests costs an increase in the system load, which in turn increases the average latency. However, as the number of cloudlets increases, the increased, hedged-requests are shown to minimize both the average delay and the 99th percentile delay. This highlights the advantage of hedged-requests in computing networks, specially in the absence of proactive capabilities. Moreover, the gains brought by proactiveness reaches up to 50% and 65% in the average and the 99th percentile delay, respectively.

4.3 Impact of caching capability

Here, we investigate the impact of the capability of caching computing tasks, where the capability is measured with respect to the task cacheability and storage availability. First, we show the impact of storage size in the performance of the proposed scheme, as compared to the BL2 without proactive capability. In Fig. 5, we show the average delay and the 99th percentile delay at different cloudlet storage size. The results show that the storage size has a significant impact on the performance of the proposed approach. For example, an increase in the storage size

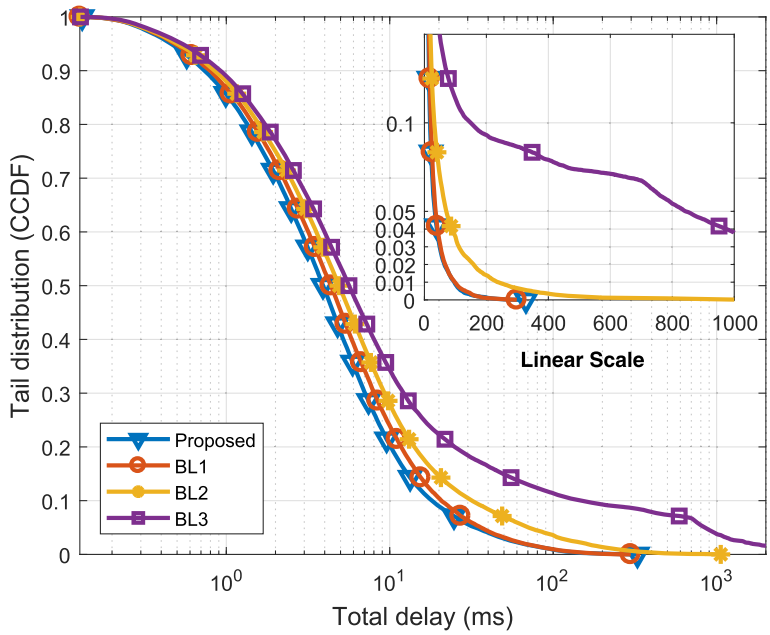

Fig. 3 The delay tail distribution. The total delay tail distribution for the proposed scheme (with a storage size s_e of 10), and the baseline schemes, with $E = 30$ cloudlets, and UN density $U/E = 3$

from 2 to 4 decreases the average and the 99th percentile delay by almost by 20 and 30%, respectively. However, as the storage size further increases, the rate of reduction in latency decreases. This is due to the caching policy in which popular tasks are cached first. Hence, small storage sizes can be sufficient to considerably minimize the computing latency.

Next, we show the impact of task cacheability on the performance of the proposed scheme. In Fig. 6, we plot the delay performance for different cacheable ratio, defined

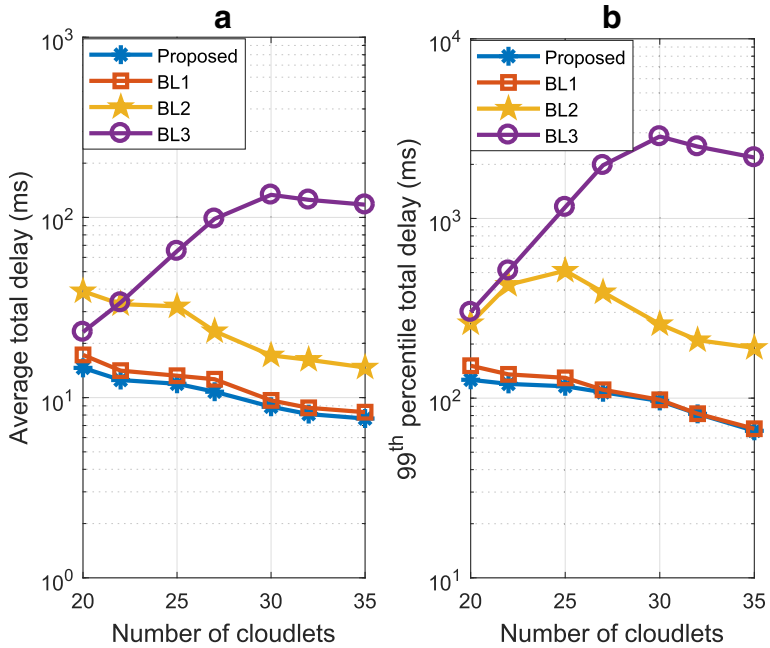

Fig. 4 The delay performance vs. network density. **a** The average delay and **b** the 99th percentile delay performance for different network densities, with $E = 30$ cloudlets, and UN density $U/E = 3$

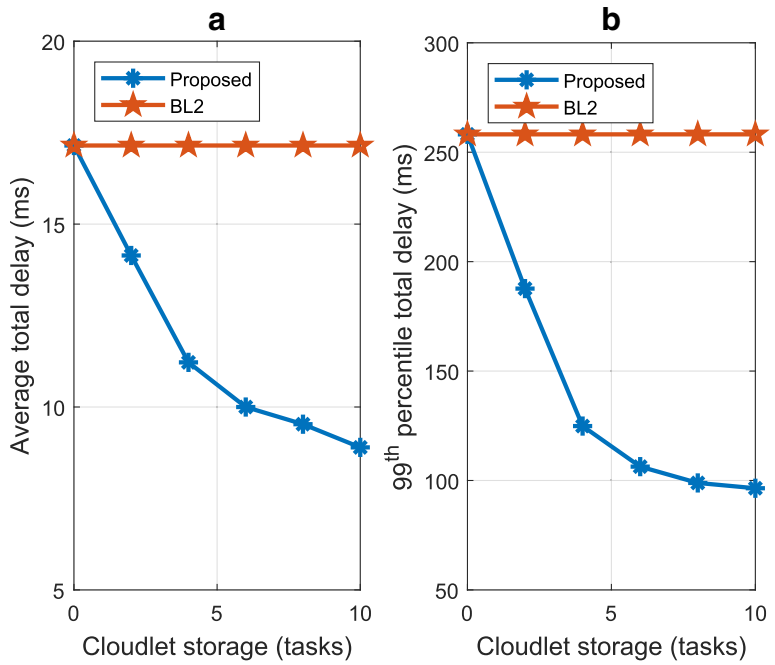

Fig. 5 The delay performance vs. storage size. **a** The average delay and **b** the 99th percentile delay performance for different cloudlet storage sizes, with $E = 30$ cloudlets, and UN density $U/E = 3$

as the ratio of the tasks whose computing results can be cached and reused. It can be shown that with the proposed scheme, both the average and 99th percentile delays decrease significantly when more tasks are cacheable, for the same storage size, since more task results can be reused with minimal latency. Similar behavior is observed with BL1 due to its caching capability. However, a higher latency is observed as compared to the proposed scheme, since BL1 relies on distributed caching that does not cluster similar UNs together.

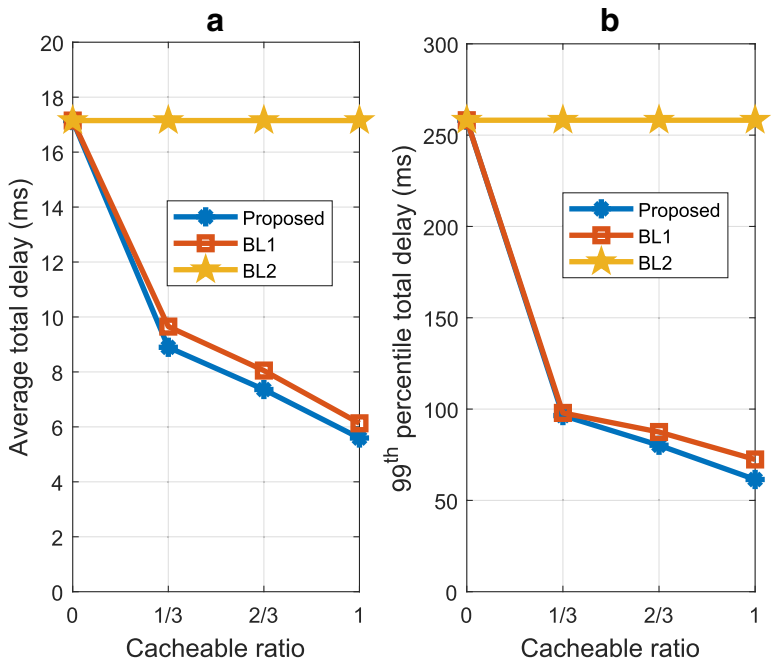

Fig. 6 The delay performance vs. cacheable ratio. **a** The average delay and **b** the 99th percentile delay performance for different ratios of cacheable tasks, with $E = 30$ cloudlets, UN density $U/E = 3$, and a storage size s_e of 10

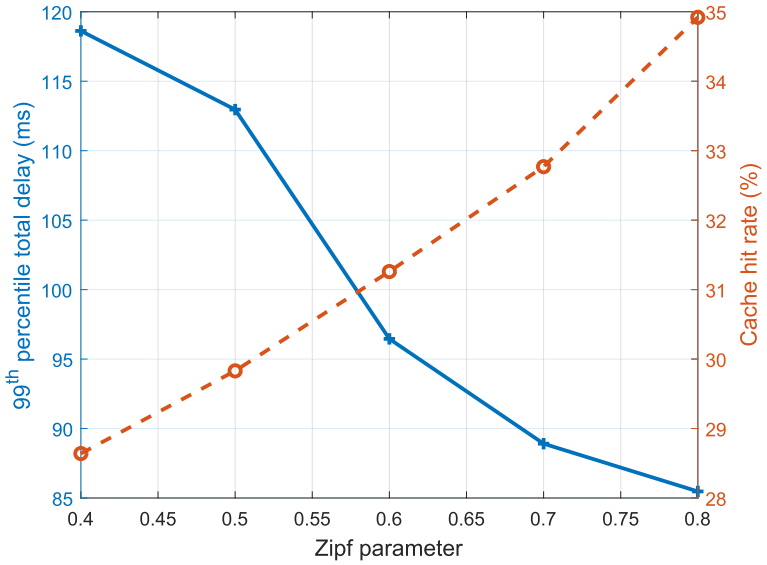

Fig. 7 The cache hit rate and delay performances vs. the popularity distribution. The 99th percentile delay and cache hit rate as the Zipf parameter z varies, with $E = 30$ cloudlets, UN density $U/E = 3$, and a storage size s_e of 10

4.4 Impact of popularity distribution

Finally, we show the impact of the different popularity distributions on the performance of the proposed proactive approach. The 99th percentile delay and the cache hit rate are investigated in Fig. 7 against different values of the Zipf parameter z. The Zipf parameter reflects the discrepancy level of the task popularity distribution. When the discrepancy level increases (higher z), the popularity gap between the most and least popular tasks increases. Hence, at high values of z, the likeliness of serving a request from the cache increases as compared to lower values, for the same storage size. Accordingly, we can see from Fig. 7 that as z increases, the cache hit rate (defined as the ratio of the requests served from the cache) increases. As a result, the 99th percentile delay decreases.

5 Conclusions

In this paper, a proactive computing and task distribution scheme for ultra-reliable and low-latency fog computing networks has been introduced. In the proposed approach, clusters of cloudlets and edge user nodes are formed based on spatial proximity and similar interests in computing results. Each cluster proactively caches computing results in the storage of its cloudlets to minimize the computing latency. Moreover, a matching algorithm to distribute the computing tasks to cloudlets is presented. The matching algorithm allows latency-critical requests to be matched to multiple cloudlets such that their latency constraints are met. Simulation results have shown that the proposed approach strikes a balance between minimizing the service latency and maximize the service reliability at different network conditions.

Endnote

[1] The processing density is application dependent, we consider a variable bit rate (VBR) encoding process as an example, reported in [28].

Funding
This research was supported by the Academy of Finland (CARMA) project, NOKIA donation on fog (FOGGY project), and by the US Office of Naval Research (ONR) under Grant N00014-15-1-2709.

Availability of data and materials
The paper is self-contained. Simulations description and parameters are provided in details in Section 4.

Authors' contributions
All authors have contributed to this manuscript and approved the submitted manuscript.

Competing interests
The authors declare that they have no competing interests.

Publisher's Note

Author details
[1]Centre for Wireless Communications (CWC), University of Oulu, Oulu, Finland. [2]Wireless@VT, Bradley Department of Electrical and Computer Engineering, Virginia Tech, Blacksburg, VA, USA. [3]Department of Computer Science and Engineering, Kyung Hee University, Yongin-si, Korea.

Received: 26 March 2018 Accepted: 30 July 2018
Published online: 20 August 2018

References

1. Z Dawy, W Saad, A Ghosh, JG Andrews, E Yaacoub, Toward massive machine type cellular communications. IEEE Wirel. Commun. **24**(1), 120–128 (2017). https://doi.org/10.1109/MWC.2016.1500284WC
2. M Mozaffari, W Saad, M Bennis, M Debbah, Unmanned aerial vehicle with underlaid device-to-device communications: Performance and tradeoffs.

IEEE Trans. Wirel. Commun. **15**(6), 3949–3963 (2016). https://doi.org/10.1109/TWC.2016.2531652
3. F Bonomi, R Milito, J Zhu, S Addepalli, in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*. Fog computing and its role in the internet of things (ACM, New York, 2012), pp. 13–16. https://doi.org/10.1145/2342509.2342513
4. Y Mao, C You, J Zhang, K Huang, KB Letaief, A survey on mobile edge computing: The communication perspective. IEEE Commun. Surv. Tutorials. **PP**(99), 1–1 (2017). https://doi.org/10.1109/COMST.2017.2745201
5. G Lee, W Saad, M Bennis, An online optimization framework for distributed fog network formation with minimal latency. ArXiv e-prints (2017). http://arxiv.org/abs/1710.05239
6. S Barbarossa, S Sardellitti, P Di Lorenzo, in *2013 IEEE 14th Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*. Joint allocation of computation and communication resources in multiuser mobile cloud computing, (2013), pp. 26–30. https://doi.org/10.1109/SPAWC.2013.6612005
7. C You, K Huang, H Chae, BH Kim, Energy-efficient resource allocation for mobile-edge computation offloading. IEEE Trans. Wirel. Commun. **16**(3), 1397–1411 (2017). https://doi.org/10.1109/TWC.2016.2633522
8. Y Mao, J Zhang, SH Song, KB Letaief, in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*. Power-delay tradeoff in multi-user mobile-edge computing systems, (2016), pp. 1–6. https://doi.org/10.1109/GLOCOM.2016.7842160
9. YH Kao, B Krishnamachari, MR Ra, F Bai, Hermes: Latency optimal task assignment for resource-constrained mobile computing. IEEE Trans. Mob. Comput. **16**(11), 3056–3069 (2017). https://doi.org/10.1109/TMC.2017.2679712
10. X Lyu, H Tian, C Sengul, P Zhang, Multiuser joint task offloading and resource optimization in proximate clouds. IEEE Trans. Veh. Technol. **66**(4), 3435–3447 (2017). https://doi.org/10.1109/TVT.2016.2593486
11. C Liu, M Bennis, HV Poor, in *2017 IEEE Globecom Workshops (GC Wkshps)*. Latency and reliability-aware task offloading and resource allocation for mobile edge computing. ArXiv e-prints, (2017), pp. 1-7. https://doi.org/10.1109/GLOCOMW.2017.8269175
12. X Chen, L Jiao, W Li, X Fu, Efficient multi-user computation offloading for mobile-edge cloud computing. IEEE/ACM Trans. Networking. **24**(5), 2795–2808 (2016). https://doi.org/10.1109/TNET.2015.2487344
13. H Shah-Mansouri, VWS Wong, Hierarchical fog-cloud computing for IoT systems: A computation offloading game. IEEE Internet of Things Journal, 1–1 (2018). https://doi.org/10.1109/JIOT.2018.2838022
14. G Lee, W Saad, M Bennis, in *2017 IEEE International Conference on Communications (ICC)*. An online secretary framework for fog network formation with minimal latency, (2017), pp. 1–6. https://doi.org/10.1109/ICC.2017.7996574
15. Lee, G, Saad, W, Bennis, M, in *2017 IEEE Fog World Congress (FWC)*. Online optimization for low-latency computational caching in Fog networks, (2017), pp. 1–6. https://doi.org/10.1109/FWC.2017.8368529
16. MS ElBamby, C Perfecto, M Bennis, K Doppler, Toward low-latency and ultra-reliable virtual reality. IEEE Netw. **32**(2), 78–84 (2018). https://doi.org/10.1109/MNET.2018.1700268
17. SW Ko, K Huang, SL Kim, H Chae, Live prefetching for mobile computation offloading. IEEE Trans. Wirel. Commun. **PP**(99), 1–1 (2017). https://doi.org/10.1109/TWC.2017.2674665
18. SW Ko, K Huang, SL Kim, H Chae, in *2017 IEEE International Conference on Communications (ICC)*. Energy efficient mobile computation offloading via online prefetching, (2017), pp. 1–6. https://doi.org/10.1109/ICC.2017.7997341
19. MS Elbamby, M Bennis, W Saad, in *2017 European Conference on Networks and Communications (EuCNC)*. Proactive edge computing in latency-constrained fog networks, (2017), pp. 1–6. https://doi.org/10.1109/EuCNC.2017.7980678
20. Q Hu, C Wu, X Zhao, X Chen, Y Ji, T Yoshinaga, Vehicular multi-access edge computing with licensed sub-6 ghz, ieee 802.11p and mmwave. IEEE Access. **6**, 1995–2004 (2018). https://doi.org/10.1109/ACCESS.2017.2781263
21. E Baştuğ, M Bennis, M Debbah, Living on the edge: the role of proactive caching in 5g wireless networks. IEEE Commun. Mag. **52**(8), 82–89 (2014). https://doi.org/10.1109/MCOM.2014.6871674
22. MS ElBamby, M Bennis, W Saad, M Latva-aho, in *Proc. 11th Intl. Symp. on Wireless Communications Systems (ISWCS)*. Content-aware user clustering and caching in wireless small cell networks, (2014), pp. 945–949. https://doi.org/10.1109/ISWCS.2014.6933489
23. A Anpalagan, M Bennis, R Vannithamby, *Design and deployment of small cell networks*. (Cambridge University Press, UK, 2015)
24. J Dean, LA Barroso, The tail at scale. Commun. ACM. **56**(2), 74–80 (2013). https://doi.org/10.1145/2408776.2408794
25. A Mukherjee, in *IEEE Globecom Workshops (GC Wkshps)*. Queue-aware dynamic on/off switching of small cells in dense heterogeneous networks, (2013), pp. 182–187. https://doi.org/10.1109/GLOCOMW.2013.6824983
26. Y Gu, W Saad, M Bennis, M Debbah, Z Han, Matching theory for future wireless networks: fundamentals and applications. IEEE Commun. Mag. **53**(5), 52–59 (2015). https://doi.org/10.1109/MCOM.2015.7105641
27. J Cranshaw, R Schwartz, JI Hong, N Sadeh, in *Proc. International AAAI Conference on Weblogs and Social Media*. The livehoods project: utilizing social media to understand the dynamics of a city, (2012), p. 58
28. AP Miettinen, JK Nurminen, in *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing. HotCloud'10*. Energy efficiency of mobile clients in cloud computing (USENIX Association, Berkeley, 2010), pp. 4–4. http://dl.acm.org/citation.cfm?id=1863103.1863107
29. X Ma, C Lin, X Xiang, C Chen, in *Proceedings of the 18th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems. MSWiM '15*. Game-theoretic analysis of computation offloading for cloudlet-based mobile cloud computing (ACM, New York, 2015), pp. 271–278. https://doi.org/10.1145/2811587.2811598