A Two-Sided Matching Approach for Distributed Edge Computation Offloading

CS 466: Cloud computing project

Shreesha Bharadwaj - 181CO249
Nandan MK - 181CO234
Vignesh Srinivasan - 181CO258

Introduction:

With the rapid growth of Internet of Things (IoT), billions of mobile devices are connected to wireless networks. This inevitably leads to massive data and computation and requires high resource consumption. However, mobile devices are small physical devices with limited battery and computing capacities. To help combat this, cloud computing has been recognized as a possible solution that offloads computation tasks from mobile device to remote cloud centers. However, cloud computing causes excessive latency due to the long propagation distances.

Compared to cloud computing, mobile edge computing (MEC) has been proposed as an alternative effective technology to handle the strict low-latency requirement of the applications. MEC offloads latency-sensitive computation tasks from mobile devices to the physically proximal network edge (such as access point (AP) or base stations (BS)) instead of the remote cloud centers. This significantly reduces transmission latency and energy consumption. This enables MEC to support low latency requirements in the 5G mobile communication systems and thus become a key technology in 5G and IoT.

There are many works studying the joint radio-and computation resources allocation for MEC. Among them, a lot of works focus on energy minimization of mobiles and a variety of energy-efficient joint radio-and
computation resource allocation problems are studied.

Although current works make significant progress for MEC, there exist some important issues that are often neglected. Firstly, most of previous works just consider one AP (MEC server). However, realistically, there are usually multiple APs (MEC servers) with heterogeneous resources. Here, heterogeneous resources occur in different sizes and have different functions. Therefore, users can select the appropriate APs for task offloading according to their requirements and network situation. A more practical scenario with the heterogeneity of resources needs to be considered. Secondly, most of the existing solutions to resource allocation problems in MEC require the complete knowledge of the network and thus suffer from extreme signaling overhead. Hence, distributed schemes with local information as well as independent computing are necessary in the case of multiple APs with multiple servers.

System Model

Consider a multi-user MEC system, consisting of K users and M APs, where each AP is integrated with a MEC server, as shown in Fig. 1.

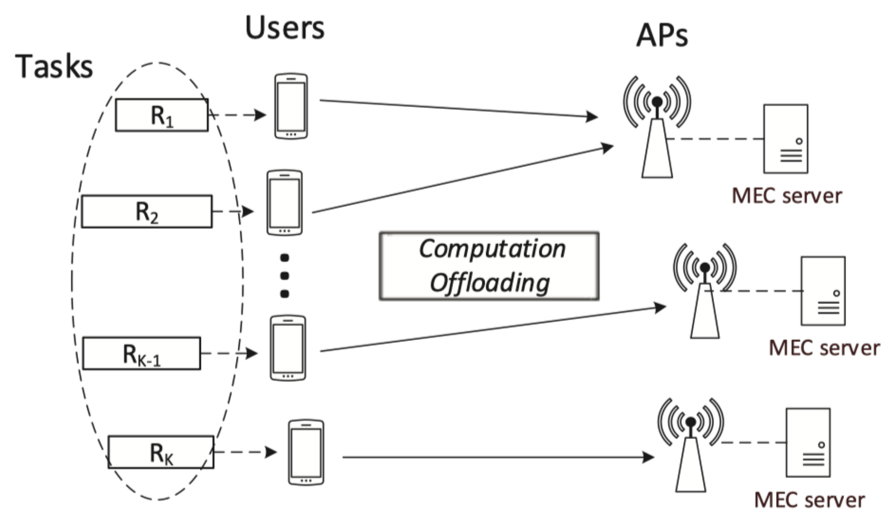


Figure 1: System model.

Let us assume that each user occupies an orthogonal channel so that they can simultaneously offload their data to the edge clouds without interference. Furthermore, partial offloading is considered, whereby a portion of the computation task of each user can be executed locally while the rest is offloaded to an AP at the same time. In addition, a quasi-static channel model is considered, in which the channel remains constant during each offloading period, but varies in different offloading periods.

Let $C(k)$ be the number of CPU cycles for computing 1 bit of input data at user k and $R(k)$ be the total input data of user k to be executed. Denote $\ell(k,m)$ as the total number of computation input bits offloaded from user k to the AP m, with $0 \le \ell_{k,m} \le R_k$, and the rest part of $(R_k - \ell(k,m))$ bits are locally executed at user k itself. We consider F_k as the local CPU frequency of user k that is measured by the number of CPU cycles per second. $f_{k,m}$ is the computational speed of the edge cloud assigned by AP m to user k.

Problem Formulation

First, we introduce a set of binary variables $\{ x_{k,m} \}$ which are related to AP section and can be written as

$$x_{k,m} = \begin{cases} 1, & \text{if user } k \text{ offloads computation to AP } m, \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

We assume that each user k can offload computation task to at most one AP, which is expressed as

$$\sum_{m=1}^{M} x_{k,m} \leq 1, \quad \forall k. \quad (2)$$

Matching Based Distributed Algorithm

P1 is the joint resource allocation problem and the two variables x and ℓ are coupled. So we attempt to obtain a suboptimal solution by solving the problem in two steps, and each step solves one of x and ℓ. The first step is to solve the optimal computation offloading bits $\ell*$. The second step is to solve the user-AP association $x*$ under given $\ell*$ by applying matching theory. Both the two steps can be implemented in a distributed manner.

Distributed Offloading Data l^*

Firstly, we solve the optimal $\ell*$. We denote t_k^{loc} and $t_{k,m}^{off}$ as the time used for local computing of user k and the time that user k spends for offloading computation task to AP m, respectively. Since the local computing and offloading can be performed concurrently, the delay of user k for executing the total R_k bits data can be expressed as

$$t_{k,m} = \max\{t_k^{\text{loc}}, t_{k,m}^{\text{off}}\}.$$

The delay of local computing for each user k can be given by

$$\boxed{t_k^{\mathrm{loc}} = (R_k - \ell_{k,m})C_k/F_k.}$$

The offloading time consists of three parts: the uplink transmission time, the execution time at AP m, and the downlink feedback time from the AP.

The optimal offloaded data can be described as a piecewise function:

$$\boxed{t_{k,m} = \begin{cases} t_k^{\mathrm{loc}} & 0 \leq \ell_{k,m} \leq \ell_{k,m}^*, \\ t_{k,m}^{\mathrm{u}} + t_{k,m}^{\mathrm{c}} + t_{k,m}^{\mathrm{d}} & \ell_{k,m}^* \leq \ell_{k,m} \leq R_k. \end{cases}}$$

The minimal offloading time can be obtained through the following equation:

$$\boxed{t_{k,m}^* = \frac{R_k C_k}{F_k}\left(1 - \frac{C_k}{\frac{F_k}{r_{k,m}^{\mathrm{u}}} + \frac{C_k F_k}{f_{k,m}} + C_k + \frac{\alpha_{k,m} F_k}{r_{k,m}^{\mathrm{d}}}}\right)}$$

Distributed User-AP Association x*

We apply the two-sided matching theory to the association problem, which allows each network node to decide their individual actions based on local information. In this section, we first formulate the association problem as a two-sided matching game. Then we present a matching algorithm which can find a stable matching.

Algorithm 1 Two-Sided Matching Based Resource allocation Algorithm

Input: $\mathcal{P}_k{}^{(t)}$, $\mathcal{P}_m{}^{(t)}$, $\forall k, m$.
Initialize: $t = 1$, $\mu^{(1)} \triangleq \{\mu(k)^{(1)}, \mu(m)^{(1)}\}_{k\in\mathcal{K}, m\in\mathcal{M}} = \emptyset$, $F_m^{\mathrm{res}(1)} = F_m$, $\zeta_m^{(1)} = \emptyset$, $\forall k, m$.
$t \to t+1$,
For $\forall k$, update $\mathcal{P}_k^{(t)}$ according to given $\mu(m)^{(t-1)}$.
For $\forall k$, select the most preferred m in its $\mathcal{P}_k{}^{(t)}$.
while $k \notin \mu(m)^{(t)}$ and $\mathcal{P}_k{}^{(t)} \neq \emptyset$ **do**
    if $F_m^{\mathrm{res}(t)} < \ell_{k,m} C_k$ **then**
        The users ranked lower than k in current matching $\mu(m)^{(t)}$ form $\mathcal{P}_m^{'}{}^{(t)} = \{k' \in \mu(m)^{(t)} \mid k \succ_m k'\}$.
        $j_{lp} \leftarrow$ the least preferred $k' \in \mathcal{P}_m^{'}{}^{(t)}$.
        while $(\mathcal{P}_m^{'}{}^{(t)} \neq \emptyset) \bigcup (F_m^{\mathrm{res}(t)} < \ell_{k,m} C_k)$ **do**
            $\mu(m)^{(t)} \leftarrow \mu(m)^{(t)} \setminus j_{lp}$, $\mathcal{P}_m^{'}{}^{(t)} \leftarrow \mathcal{P}_m^{'}{}^{(t)} \setminus j_{lp}$.
            $F_m^{\mathrm{res}(t)} \leftarrow F_m^{\mathrm{res}(t)} + \ell_{j_{lp},m} C_{j_{lp}}$.
        end while
        if $F_m^{\mathrm{res}(t)} < \ell_{k,m} C_k$ **then**
            $j_{lp} \leftarrow k$.
        end if
    else
        $\mu(m)^{(t)} \leftarrow \mu(m)^{(t)} \bigcup k$, $F_m^{\mathrm{res}(t)} \leftarrow F_m^{\mathrm{res}(t)} - \ell_{k,m} C_k$.
    end if
end while
$\zeta_m^{(t)} = \{j \in \mathcal{P}_m{}^{(t)} \mid j_{lp} \succ_m j\} \bigcup \{j_{lp}\}$.
for $j \in \zeta_m^{(t)}$ **do**
    $\mathcal{P}_j{}^{(t)} \leftarrow \mathcal{P}_j{}^{(t)} \setminus m$, $\mathcal{P}_m{}^{(t)} \leftarrow \mathcal{P}_m{}^{(t)} \setminus j$.
end for
Output: $\mu^{(t)}$.

Implementation

The following values of variables were used for the simulation.

- Number of APs, M = 3
- Number of Users, k ∈ [10, 15, 20, 25, 30]
- The local CPU frequency, F_k ∈ [400,500] MHz
- Data size of each user, C_k ∈ [500,1500] cycles/bit
- BIts of each user, R_k ∈ [50,80] kb
- Computational speed of the edge cloud assigned by AP m to user k, $F_{k,m}$ ∈ [3,15] GHz
- CPU cycles of AP, F_m ∈ [50,100] MHz/slot
- Uplink transmission rates, R_{ukm} ∈ [0.7,1.2] Mbp/s
- Downlink transmission rates, R_{dkm} ∈ [2,5] Mbp/s
- Out-input ratio of the offloaded data, α_{km} = 0.2

We have implemented the above algorithm in a python program. The first step was to initialize the values of the network variables as mentioned above by using the uniformly distributed random function. The random variables are generated once for all possible values of k and the same is used throughout.

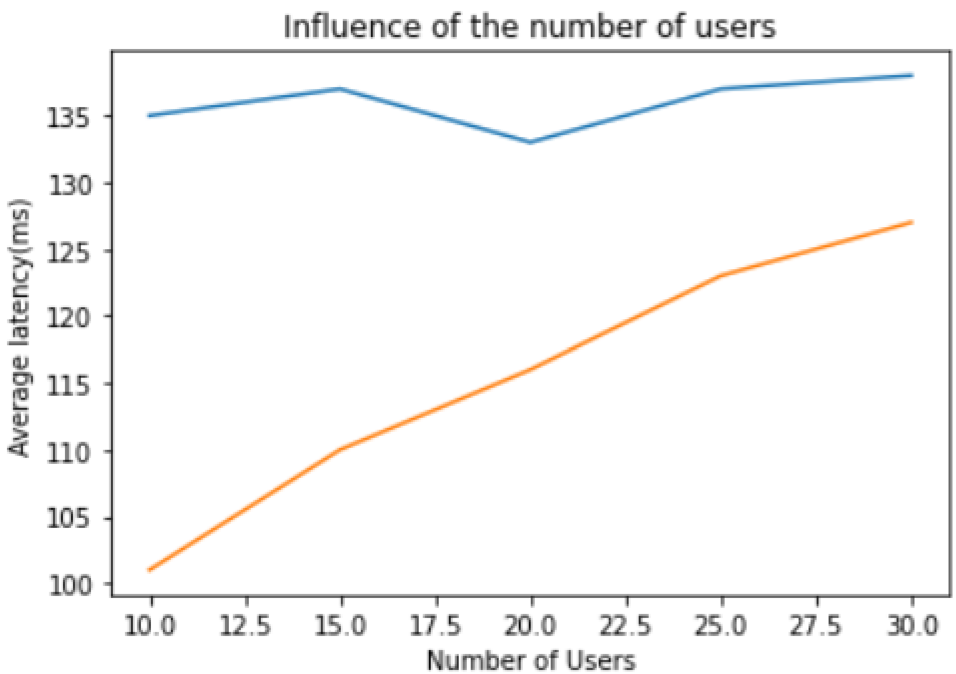
The first step was to compute the optimal amount for bits that has to be offloaded by every user k to an AP m so that the latency of bits computed locally and those which are computed at the AP have similar latencies. This ensures that the total computation time remains a minimum. This was done using the formula for $L_{k,m}$ mentioned in the previous section.

The next step was to map the users to the corresponding AP to which the data will be offloaded. The preference profile for each user and each AP is required to perform this step. To perform the mapping for every user its preferred AP is selected and checked if resources are available in that particular AP. If it is available then bits are offloaded. If the bits are not available then the preference of the AP is checked and depending on the position of the user, other users are dropped to accommodate the current user. This process is followed until all users have their offloaded or no APs are available. From the mapping thus obtained the average latencies of users is found.

In order to evaluate the performance of the proposed algorithm, we also consider the local computing scheme and a heuristic algorithm for a comparison purpose. In the local computing scheme, all users adopt local computing for their tasks. In the heuristic scheme, all users are ranked in a decreasing order according to the data-size of the tasks R_k and offload each entire task R_k to an AP in this order until this AP's computational capacity is fulfilled and then offload to another AP. If all APs are fulfilled, the rest R_k's are executed locally.

The results from the above algorithms were plotted to compare their performance. The first comparison was to compare average latency with the changing in the number of users present in the system. The second graph was to compare the results of average latency changes compared with the edge cloud computation capacity.

Results



The above graph represents the results obtained from the comparison of the local only computing and the two sided offloading approach. It gives the average latencies of the two approaches to differing number of users in the system.
Here Blue line represents the two sided approach and orange line represents local only approach.

From the graph above, we can see that the average latency for the case all computation is done locally is pretty much constant. For the case where some bits of the computation are offloaded to the AP, the average latency is consistently lesser than that of local computation. The latency for this case slowly goes up as the number of users increases naturally owing to increased bandwidth requirements and limited computing resources resources available at the AP.

References

H. Bao and Y. Liu, "A Two-Sided Matching Approach for Distributed Edge Computation Offloading," 2019 IEEE/CIC International Conference on Communications in China (ICCC), 2019, pp. 535-540, doi: 10.1109/ICCChina.2019.8855906.