Collaborative Computation Offloading for Multi-access Edge Computing

Shuai Yu
LIP6
Sorbonne University
4 Place Jussieu, 75005, Paris, France.
shuai.yu@lip6.fr

Rami Langar
LIGM/UPEM
University Paris Est
Cité Descartes, 77454, Marne-la-Vallée, France.
rami.langar@u-pem.fr

***Abstract*—Computation offloading is a proven successful paradigm for enabling resource-intensive applications on the mobile devices in multi-access edge computing (MEC) network. Moreover, in view of emerging mobile collaborative application (MCA), the offloaded tasks can be duplicated when multiple users are in the same proximity. This motivates us to design novel collaborative offloading schemes. In this context, we first consider the MEC offloading scenario, where multiple mobile users offload duplicated computation tasks to the network edge servers, and share the computation results among them. Our goal is to develop the optimal fine-grained collaborative offloading strategies with data caching enhancements to minimize the overall execution delay at the mobile terminal side. Next, we extend the MEC offloading to hybrid offloading (i.e., joint MEC and Device-to-Device (D2D) offloading) with social relationship consideration, and propose a hybrid D2D multicast-based task execution framework to achieve an energy-efficient task assignment policy for mobile users. To overcome the great complexity for the deployment problems, we then formulate the offloading decision problem as a multi-label classification problem and develop the Deep Supervised Learning (DSL) method to achieve a rapid offloading decisions making, as well as minimize the computation and offloading overhead. Last but not least, we evaluate their performance through extensive numerical study, which shows the superior performance of the proposed scheme.**

***Index Terms*—computation offloading, multi-access edge computing, data caching, multicast communication, socially aware, monte carlo tree search, deep learning.**

I. Introduction

Accompanied by the emergence of near-to-eye display technologies, such as Google Glass, a variety of mobile resource hungry applications are developed to meet the user's requirements, such as augmented reality (AR) [1], collaborative gaming and mobile crowd sensing applications [2]. These applications make use of complex algorithms for camera tracking, image processing and pattern recognition which are resource-intensive and, hence, beyond the capabilities of current mobile devices. A potential solution to address the challenges is to offload the computation to nearby resourceful cloudlet [3].

To this end, the European Telecommunications Standards Institute (ETSI) proposed multi-access edge computing (MEC) [4]. In the proposed architectures, substantial compute and storage resources are placed at the edge of the Internet, in close proximity to mobile devices, sensors, end users, and Internet of Things devices. This physical proximity improves latency, bandwidth, trust, and survivability, thus allowing a large class of state-of-the-art applications, like Big Data and the Internet of Things, to be deployed in a very effective way.

In addition, for certain types of mobile collaborative application (MCA), multiple users in the same neighborhood typically look at the same scene, track the same environment, and need to recognize the same objects, so they can benefit from collaboration and computation/data sharing [1]. A typical example is emerging mobile crowd sensing applications [2], where individual mobile user with sensing and computing devices collectively share data and extract information to measure and map phenomena of common interest. Similarly, AR applications [1] have the unique property that different users with the same objective can share part of the computational tasks and of the input and output data. This motivate us to design novel collaborative computing offloading schemes for multi-user MEC network.

In this paper, we will investigate fine-grained computation offloading framework for MCA execution in MEC network. Our work consists of three contributions that are summarised as follows:

- In the first contribution, we propose a fine-grained collaborative computation offloading and date caching strategy that optimizes the offloading decisions on the mobile terminal side with data caching enhancement. The objective is to minimize the overall execution latency for the mobile users within the network. We propose a concept of the cooperative call graph to model the offloading and caching relationship within multiple mobile users, and then compute the delay and energy overhead for each single user. Then, we explore the concept of the coalition formation game for the distributed caching scenario in multi-user multi-cell MEC network. This part of work was first published in the proceedings of *IEEE International Conference on Communications (ICC 2016)* [5], then, an extension version was published in the journal of *IEEE Transactions on Vehicular Technology* [6].
- In the second contribution, we propose a framework of socially aware D2D computation offloading (*SAHCO*) and then compute the energy overhead for each ap-

978-3-903176-15-7

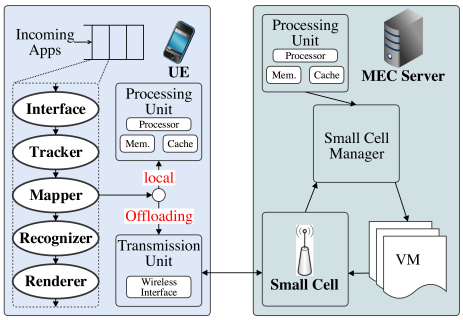

Fig. 1: Mobile computation offloading in MEC scenario

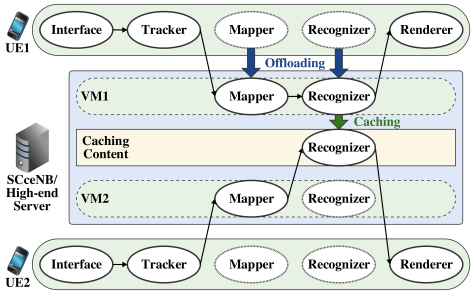

Fig. 2: Collaborative call graph with caching enhancement.

plication cluster. In order to enhance the performance, we solve the computation offloading problem for all the components of an application at the same time. To address the problem for the set of MCA components, we propose a new optimal task assignment approach based on Monte-Carlo search tree (MCTS) [7], named *TA-MCTS*. Our proposed solution, *TA-MCTS*, achieves an optimized computation offloading policy. This part of work was first published in the proceedings of *IEEE Global Communications Conference (GLOBECOM 2016)* [8], then, an extension version was submitted to the journal of *IEEE Transactions on Mobile Computing* [6], and under major revision now.

- In the last contribution, we propose a deep supervised learning (DSL) based computation offloading framework. Our objective is to achieve a rapid offloading decision, and minimize the offloading cost for the MEC network at the same time. The offloading actions taken by a mobile user consider the local execution overhead as well as varying network conditions (including wireless channel condition, available communication and computation resources). Our method provide a pre-calculated offloading solution which is employed when a certain level of knowledge about the application and network conditions. We formulate the continuous offloading decision problem as a multi-label classification problem. This modelling strategy largely benefits from the emerging deep learning methods in the artificial intelligence field. Our method approaches the optimal solution obtained by the exhaustive strategy performance with a very subtle margin. This part of work was published in the proceedings of the *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC 2017)* [9].
- The URL to the thesis text is shown in [10].

The reminder of this paper is organized as follows. Section II presents the system model. Section III introduce our first contribution, followed by descriptions of our second and third contributions in Section IV and Section V, respectively. Simulation results are presented in Section VI. Finally, conclusions are drawn in Section VII.

II. System Model

As illustrated in Fig.1, we consider a small cell-based MEC system, which is also known as small cell cloud. The basic idea is to enhance small cell base stations (e.g., pico, femto) by an additional computation and storage capabilities. In this article, the novel base station is called Small Cell cloud-enhanced e-Node B (SCceNB).

We consider our MEC system consists of a set $\mathcal{M} = \{1, 2, ..., M\}$ of mobile users and a set $\mathcal{N} = \{1, 2, ..., N\}$ of SCceNBs. Assume that each SCceNB can serves at most q mobile users. The maximum achievable uplink and downlink rate (in bps) over an additive white Gaussian noise (AWGN) channel for user m ($m \in \mathcal{M}$) to offload its application to SCceNB n ($n \in \mathcal{N}$) can be expressed as $r^{ul}_{n,m}$ and $r^{dl}_{n,m}$, respectively.

A. Application Model

We assume that a mobile application can be split into multiple components which in the granularity of either method or thread (i.e., a fine-grained partitioning) [11]. Then, we model the relationship between components as a weighted directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ denotes the set of components, and $\mathcal{E}$ the data dependencies between components. We assume each edge $\mathcal{E}_{u,v}$ ($\mathcal{E}_{u,v} \in \mathcal{E}$) represents the data communication (computation result) between two components. We let ϕ_v ($v \in \mathcal{V}$) denotes the weight of component v, which specifies the workload (CPU cycles) for the component v. For a given input data size $\mathcal{E}_{u,v}$, $\phi_v = \omega \cdot \mathcal{E}_{u,v}$, where the ω in CPU cycles/byte (cpb) indicates the number of clock cycles a microprocessor will perform per byte of data processed in an algorithm. The parameter depends on the nature of the component, e.g., the complexity of the algorithm.

Then, we propose a concept of the collaborative call graph for multi-user MCA execution scenario, as shown in Fig. 2. When a group of mobile users connect to the same SCceNB, they can cooperate through sharing their input data and computation results in the server. For example, Fig. 2 shows that UE1 and UE2 offload their computation to the edge server and share the corresponding computation results. They can benefit from the fact that the result of component "Recognizer" is already cached in the edge server to reduce the execution latency. Or if there is no such result cached in the edge server, they can

collaboratively execute the component in the edge server for one time, instead of two times separately.

B. Execution Model

For the MEC deployment, each component can be executed either on the mobile device or offloaded to a SCceNB. The offloading decision is based on the workload of components $\mathcal{V}$, data communication $\mathcal{E}$, data rate $r^{dl}_{n,m}$ and $r^{ul}_{n,m}$.

If a component v is executed on mobile device, the completion time is t^{local}_v. Conversely, if component v is offloaded to a Virtual Machine (VM) in a SCceNB, the mobile device is idle before receiving the computation results. We denote t^{remote}_v as the completion time of component v executed on the SCceNB.

When component v is offloaded to SCceNB and the input data $\mathcal{E}_{u,v}$ from its previous component u is stored locally (i.e., stored in the mobile device), $\mathcal{E}_{u,v}$ must be sent to SCceNB before the execution of component v. Therefore, the time for UE m sending input data from component u to component v in SCceNB n is $t\_s^{u,v}_{n,m} = \frac{|\mathcal{E}_{u,v}|}{r^{ul}_{n,m}}$. Conversely, if component v is executed locally, and its previous component u is executed in SCceNB, the output data $\mathcal{E}_{u,v}$ of component u must be sent back to mobile device before the execution of component v. Therefore, the delay for UE m receiving output data from component u to component v in SCceNB n is $t\_r^{u,v}_{n,m} = \frac{|\mathcal{E}_{u,v}|}{r^{dl}_{n,m}}$.

III. Computation Offloading with Data Caching Enhancement

In this section, we present our first contribution. We focus on the reduction of average latency for MCA collaborative execution with data caching enhancements.

A. Problem Formulation

In order to minimize the average delay for UEs, we first formulate the single-user single-cell offloading problem as an optimal offloading strategy under given caching lists of the SCceNBs, which is a simple 0-1 programming problem that aims at selecting the optimal number of components to be offloaded at SCceNB n for UE m. Let $K_{n,v}$ ($K_{n,v} \in K^*_n$) denotes the binary computation results caching variable: which is equal to one, if UE m cannot find the computation results of component v cached in SCceNB n, and zero, if UE m can find the results in the corresponding SCceNB n through local caching. $I_{n,m,v}$ is the offloading decision variable, which is equal to one, if component v is processed remotely, or zero, if the component is executed locally.

We define the optimal offloading decision $I^*_{n,m} = \{I_{n,m,v}, v \in \mathcal{V}\}$ under caching list K^*_n as **Optimization Problem 1:**

$$I^*_{n,m}(K^*_n, \mathcal{G}) = \underset{I^l_{n,m,v}}{\operatorname{argmin}}\ T_{n,m}, \tag{1}$$

where $T_{n,m}$ denotes the total execution delay for UE m through offloading his components to SCceNB n.

Algorithm 1: Optimal Network Partition for MEC network

Initial network:
initial network partition for the UEs: $\{\{\emptyset\}, \{1\}, \{2\}, ..., \{M\}\}$.

Step 1: Component Offloading Decision
UEs work in a Non-cooperative manner
Input of Step 1: Parameters M, N, K^*_n, t^{local}_v, t^{remote}_v, $\mathcal{G}$, $t\_s^{u,v}_{n,m}$, $t\_r^{u,v}_{n,m}$.
1) Each UE builds a top preferred SCceNB list $I^*_{n,m}$ according to **Optimization Problem 1**.
2) Each UE selects its best preferred SCceNB as its serving SCceNB and submit its offloading requests.
3) For the UEs whose list is empty, they join S_0.
Output of Step 1: $I^*_{n,m}$.

Step 2: Coalition Formation
UEs work in a cooperative manner
Input of Step 2: Parameters $I^*_{n,m}$.
4) Each SCceNB receives the requests. Due to the limited computation capacity of SCceNBs, each SCceNB keeps the top UEs, and reject the rest.
5) The rejected UEs will re-apply to their next best SCceNB of their list $I^*_{n,m}$, and each SCceNB updates its serving UEs list.
6) Repeat 5), until convergence to a final Nash-stable partition π^*. For the UEs who cannot be allocated to a SCceNB, they execute the application locally and join S_0.
Output of Step 2: optimal network partition π^*

In the multi-user multi-cell MEC scenario, different SCceNBs have different data caching contents. Therefore, the caching policies K^*_n are different if one user attach to different SCceNBs, which can change their offloading decisions. On the other hand, users' offloading request can affect the local caching content of its serving SCceNB, and thus affect the offloading decision of other users who attach to the same SCceNB. Therefore, when UEs make offloading decisions in a collaborative manner, and we can minimize the average delay for the UEs as **Optimization Problem 2:**

$$\min\ \frac{1}{M} \cdot \sum_{m=1}^{M} T_{n,m}\left(I^*_{n,m}, K^*_n\right). \tag{2}$$

B. Algorithm Design

Then, we present our proposed optimal offloading with caching-enhancement scheme (OOCS) for the **Optimization Problem 2**. In order to identify which SCceNB can serve the attached users and execute the offloading decisions, we explore the concept of coalition formation game [12] to solve the problem. Note that the attachment of mobile users to a particular SCceNB can be seen as a coalition formation game in partition form with transferable utility. Specifically, let M UEs be players, and π be the set of existing users in the network. We assume that UEs in each coalition connect to a single SCceNB and form a coalition. Let S_n denotes the set of UEs that are served by SCceNB n, and S_0 denote the set of UEs that execute the application locally, i.e. without offloading. Based on this, the optimal network partition (coalition formation) is given by *Algorithm 1*. It is worth noting that UEs who can not be allocated to a SCceNB will execute their

application locally and join the coalition S_0. The final network partitions will be thus given as $\pi^* = \{S_0, S_1, ..., S_N\}$.

IV. A Socially Aware Hybrid Computation Offloading Framework for MEC

In this section, we present our second contribution. We propose a novel socially aware hybrid (D2D/MEC) computation offloading (*SAHCO*) for MEC, where a crowd of mobile devices at the network edge leverage network-assisted device-to-device (D2D) collaboration for wireless distributed computing (MDC) and outcome sharing.

A. Description of the Proposed Framework

At the beginning of each offloading decision making, the mobile devices upload the information required for offloading decision to the base station. This information relates to both the offloading data as well as the mobile device and network characteristics.

Based on the received information, the base station first classifies the mobile users into multiple application clusters as shown in Fig. 3 (a). Each cluster is formed by users executing the same application and sharing their inputs to each other. The outputs (i.e., computation results) can be also shared among them.

In order to establish reliable D2D communications among the mobile users, the latter is allowed to share the resources and computation results to its trusted users. Thus, the base station builds a social trust graph (as showed in Fig. 3 (b)) for the mobile users based on the received social relationship information. Then, it observes the current network state, computes the immediate costs of each component assignment strategy. Afterwards, based on these costs, it triggers the offloading decision process. The objective of the decision action is to select either: i) the MEC server (i.e., Base station) or ii) the set of mobile users, able to compute the components. To this end, we propose a Monte Carlo Tree Search based algorithm, named, *TA-MCTS* for the task assignment problem. For example, Fig. 3(c) shows that UEs 2, 7 and the base station are selected. Next, based on the offloading decision, the remaining mobiles users offload their computation (i.e., input data) to their corresponding servers. Note that the latter can be either the base station (i.e., MEC offloading) or a nearby mobile user (D2D offloading). Accordingly, the selected mobile users are responsible for processing and sending computation results to the other users (i.e., the transmitter in each multicast cluster). Thus we refer to this kind of mobile users as *offloadee* in this paper, and the corresponding receivers as *offloader* (e.g., UEs 1 and 3 in the coverage of offloadee UE 2).

This process is repetitively executed until either reaching the energy budget threshold or exiting the component phase. In that case, the application has been executed in the application cluster. The output of our hybrid offloading framework is a sequence of fine-grained components assignment strategy for each component of an application.

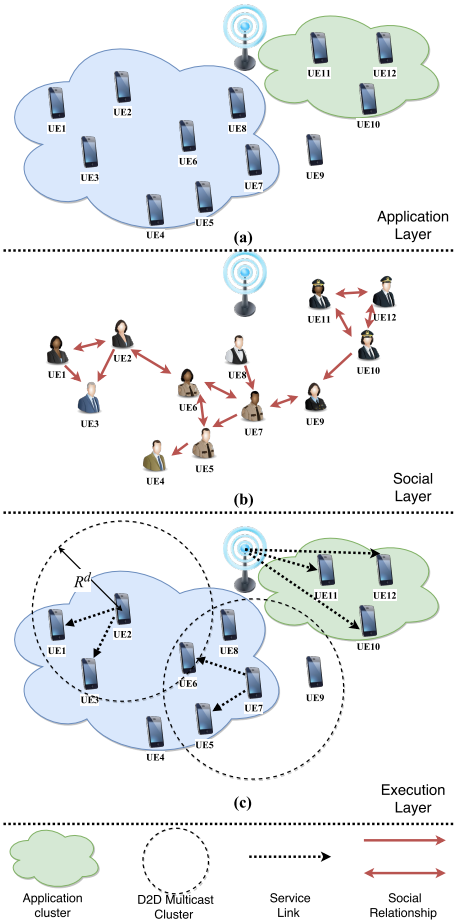

Fig. 3: Proposed socially aware hybrid computation offloading (SAHCO) system.

V. A Deep Learning based Computation Offloading for MEC

In this section, we present our last contribution. We tackle the key issue of achieving rapid offloading decision for single-user single cell MEC network (as shown in Fig. 1). Our objective is to minimize the offloading cost in time-varying wireless environment, with network resource usage consideration.

A. Problem Formulation

Our problem can be described as reinforcement learning scenario or Markov Decision Processes (MDPs). The objective is to find an agent which makes optimal offloading policy for each application. A composite state for the current system state of computation offloading can be denoted by S, which consists of mobile user's task profiles and network resource status. For the current decision period, we define immediate cost $C(S, a_v)$ as follows:

$$C(S, a_v) = \begin{cases} C_l(S) = T_l(v), & a_v = 0 \\ C_r(S), & a_v = 1 \end{cases} \quad (3)$$

where $C_l(S)$ is the immediate local execution cost (equals to the local execution delay), and $C_r(S)$ is the immediate

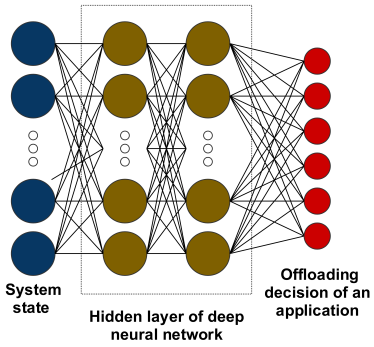

Fig. 4: Deep learning-based offloading framework.

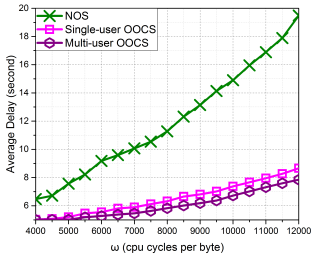

Fig. 5: Average application execution delay vs. ω.

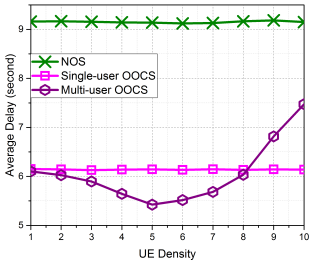

Fig. 6: Average application execution delay vs. UE density.

offloading cost which consists of communication and computation resource usage cost, the SCceNB computation cost and the data transmission cost for offloading. a_v denotes an action decision of either executing the component v locally on the mobile device (denoted $a_v = 0$) or offloading to the SCceNB (denoted as $a_v = 1$).

Thus, the optimal offloading policy denoted by $\pi^*(\pi^* \in \pi)$ can minimize the system cost given by:

$$\gamma^* = \text{argmin} \sum_{v \in \mathcal{V}} C(S, a_v). \tag{4}$$

Note that in our work, the system cost $\sum_{v \in \mathcal{V}} C(S, a_v)$ is a long-term cost, which is not provided immediately, but until all the components been processed. This cost is formally named as *delayed cost*.

B. Algorithm Design

In this section, we describe our proposed deep supervised learning based offloading scheme (DOS) for decision making.

We formulate the fine-grained offloading problem as a multi-label classification [13] framework. Specifically, given an application $\mathcal{G}$ which contains $|\mathcal{V}|$ components, the input of our model is the observation of the network states of all components. Our decision is a $|\mathcal{V}|$-dimensional vector. If a component is offloaded, its value is 1, otherwise 0. We evaluate our output with respect to the optimal output by multi-label accuracy, which is defined as the proportion of the predicted correct labels to the total number of labels for that application.

Our algorithm operates in three steps, i.e. initial phase, training phase, and action or testing phase. In the following, we describe these three phases.

1) Initial Phase: The objective of the initial phase is to obtain the raw data for training our deep supervised learning framework. We run 10,000 times the random and exhaustive-based optimal offloading strategy. Note that we used an exhaustive algorithm to search the optimal offloading policy from all the $2^{|\mathcal{V}|}$ offloading possibilities. In each execution, we vary the network conditions. Specifically, we record network state S and the corresponding optimal offloading strategy. This data is further randomly split into K_{tr} for training phase and K_{te} for testing phase.

2) Training Phase: In this phase, the features of training data are trained using deep neural network. We cross validate our training data to define the number of hidden layers to be 2, and the number of neurons to be both 128, as shown in Fig.4. Conventionally, we use rectified linear unit (ReLU) as the activation function, dropout as the regularization and sigmoid as the output. This network takes the state of S as input, and the offloading decision a_v as output. The objective is to minimize the multi-label accuracy with respect to the optimal decisions. Since this is a multi-label classification problem, we conventionally set the loss function as binary cross entropy. For optimizing the neural network, we use Adam optimizer. The neuron number of the output layer is set to $|\mathcal{V}|$. If one of the output neurons is greater than 0.5, it is decided to offload, otherwise it is not offloaded.

3) Testing Phase: Once the training phase of the deep neural network is finished, we can apply it onto any unseen application. This step is called testing phase. At this time, the deep neural network takes the state as the input and outputs the decision for all components in the application. We evaluate the performance of our network based on the outputs of the testing phase.

VI. Performance Evaluation

We first illustrate the performance of our OOCS scheme for the first contribution and compare it with respect to the no offloading scheme (NOS). Figs. 5 and 6 show the multi-user delay performance versus ω and UE density λ_u, respectively. Note that our multi-user OOCS performs better in delay reduction when ω grows as shown in Fig. 5. The reason is that the offloading probability increases with ω since more users can reduce their delay through joining our coalition game performed in OOCS. Fig. 6 shows that our multi-user OOCS performs better as UE density grows from 10^{-3} to 8×10^{-3}. A peak is observed when $\lambda_u = 5 \times 10^{-3}$, which corresponds to 11.71% and 40.61% delay reduction, compared to single-user OOCS and NOS, respectively. When $\lambda_u > 8 \times 10^{-3}$, single-user OOCS performs better. The reason is that, when λ_u grows larger ($> 5 \times 10^{-3}$), the SCceNBs cannot afford such many UEs (we assume that each SCceNB can handle 6 UEs in our simulations), and thus a large number of UEs will be rejected and run the application locally. As a result, the average delay increases and will be close to the local execution delay.

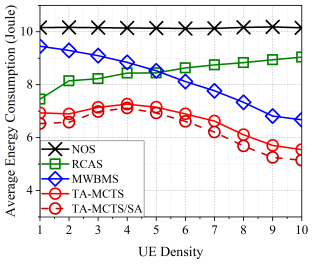

Fig. 7: Average energy consumption vs. UE density λ (10^{-3})

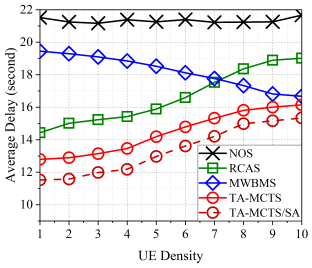

Fig. 8: Average delay vs. UE density λ (10^{-3})

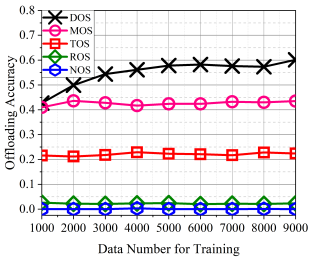

Fig. 9: Offloading accuracy vs. K_{te}.

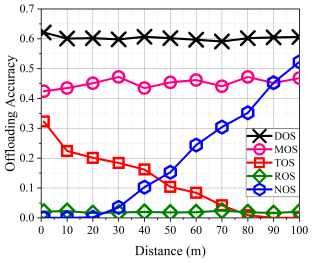

Fig. 10: Offloading accuracy vs. distance d.

For the second contribution, we illustrate the performance of our TA-MCTS scheme and compare it with respect to the NOS, Random Component Assignment Scheme (RCAS), TA-MCTS without social-award Scheme and Minimum Weighted Bipartite Matching-based Scheme (MWBMS) [8]. Fig. 7 and Fig. 8 show the average energy consumption and average delay for different mobile user density, respectively. We notice that our proposal *TA-MCTS* outperforms the related benchmark policies in term of energy saving. Note that this is observed for both cases: with or without social consideration. Consequently, our approach guarantees the delay constraints.

For the last contribution, Fig. 9 shows the impact of the number of data used to train our system (i.e. K_{tr}) on the offloading accuracy metric. We can observe that the accuracy of our DOS scheme increases as K_{tr} grows, whereas K_{tr} has little influence on the other schemes. This means that our deep learning scheme has a more powerful learning capacity than the other schemes. Fig.10 reports the impact of the distance between UE and SCceNB on the offloading accuracy metric. We can see that DOS scheme always performs better than the coarse-grained offloading schemes (i.e. TOS and NOS). In addition, we observe that the distance d has little influence on our DOS scheme. This means that our DOS scheme supports UE's mobility and remains stable under various network conditions.

VII. Conclusion

In this paper, we addressed the issue of collaborative computation offloading in MEC. We considered the execution of emerging mobile collaborative applications through MEC offloading and hybrid offloading. To this end, we split the applications into several loosely coupled software components and studied fine-grained computation offloading strategies for the components in MEC network. In the first contribution, we considered the MEC computation offloading scenario. Our objective is to develop the optimal fine-grained offloading strategies with caching enhancements to minimize the overall execution delay at the mobile terminal side. In the second contribution, we considered the hybrid offloading scenario that combines of MEC offloading and D2D offloading. A key objective of the hybrid offloading is to achieve an energy-efficient task assignment policy for mobile users. In our final contribution, we developed a dynamic offloading framework for mobile users, considering the local overhead in the mobile terminal side, as well as the limited communication and computation resources in the network side. We formulated the offloading decision problem as a multi-label classification problem and develop the Deep Supervised Learning (DSL) method to minimize the computation and offloading overhead.

References

[1] T. Verbelen, P. Simoens, F. D. Turck, and B.Dhoedt, "Leveraging cloudlets for immersive collaborative applications," IEEE Pervasive Comput., vol. 12, no. 4, pp. 30-38, Oct. 2013.

[2] G. Merlino, S. Arkoulis, S. Distefano, C. Papagianni, A. Puliafito, and S. Papavassiliou, "Mobile crowdsensing as a service," Future Gener. Comput. Syst., vol. 56, no. C, pp. 623-639, Mar. 2016.

[3] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," IEEE Pervasive Comput., vol. 8, no. 4, pp. 14-23, Oct. 2009.

[4] ETSI. (2018) "Multi-access edge computing (mec)," [On-line]. Available: https://www.etsi.org/technologies-clusters/technologies/multi-access-edge-computing

[5] S. Yu, R. Langar, W. Li, and X. Chen, "Coalition-based energy efficient offloading strategy for immersive collaborative applications in femto-cloud," in Proc. IEEE International Conference on Communications, (ICC16), Kuala Lumpur, Malaysia, May 2016.

[6] S. Yu, R. Langar, X. Fu, L. Wang, and Z. Han, "Computation offloading with data caching enhancement for mobile edge computing," IEEE Transactions on Vehicular Technology, vol. 67, no. 11, pp. 11098-11112, Nov 2018.

[7] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," IEEE Trans. Comput. Intell. AI in Games, vol. 4, no. 1, pp. 1-43, Feb. 2012.

[8] S. Yu, R. Langar, and X. Wang, "A d2d-multicast based computation offloading framework for mobile edge computing," in Proc. IEEE Global Communication Conference, (GLOBECOM16), Washington, DC, USA, Dec. 2016.

[9] S. Yu, X. Wang, and R. Langar, "Computation offloading for mobile edge computing: A deep learning approach," in Proc. 2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Commnications, (PIMRC17), Montreal, QC, Canada, Oct. 2017.

[10] S. Yu, "Multi-user Computation Offloading in Mobile Edge Computing," [Online]. https://www.researchgate.net/publication/328629402_Multi-user_Computation_Offloading_in_Mobile_Edge_Computing

[11] E. Cuervo, A. Balasubramanian, D. ki Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: Making smartphones last longer with code offload," in Proc. the 8th international conference on Mobile systems, applications, and services, (MobiSys10), ACM, New York, NY, Jun. 2010, pp. 49-62.

[12] W. Saad, Z. Han, M. Debbah, A. Hjrungnes, and T. Basar, "Coalitional game theory for communication networks," IEEE Signal Process. Mag., vol. 26, no. 5, pp. 77-97, Sep. 2009.

[13] G. Tsoumakas and I. Katakis, "Multi-label classification: An overview," Int J Data Warehousing and Mining, vol. 2007, pp. 1-13, 2007.