Research of Offloading Decision and Resource Scheduling in Edge Computing Based on Deep Reinforcement Learning

Zhen-Jiang Zhang[1](✉), Tong Wu[2], Zhiyuan Li[2], Bo Shen[2], Naiyue Chen[3], and Jian Li[2]

[1] Department of Software Engineering, Key Laboratory of Communication and Information Systems, Beijing Municipal Commission of Education, Beijing Jiaotong University, Beijing 100044, China
zhangzhenjiang@bjtu.edu.cn

[2] Department of Electronic and Information Engineering, Key Laboratory of Communication and Information Systems, Beijing Municipal Commission of Education, Beijing Jiaotong University, Beijing 100044, China
{18125066,19120091,bshen,lijian}@bjtu.edu.cn

[3] School of Computer and Information Technology, Beijing Jiaotong University, Beijing Municipal Commission of Education, Beijing 100044, China
nychen@bjtu.edu.cn

Abstract. The increasing scale of the IOT poses challenges to the energy consumption, transmission bandwidth and processing delay of centralized cloud computing data centers. The cloud computing data centers is moving from the center of the network to edge nodes with lower latency, namely, edge computing. Meanwhile, it can meet the needs of users for real-time services. In the field of edge computing, offloading decision and resource scheduling are the hot-spot issues. As for offloading decision and resource scheduling problems of single-cell multi-user partial offloading, the system model is also firstly established from four aspects: network architecture, application type, local computing and offloading computing. Based on the system model, the optimization problem of resource scheduling is modeled, where the solution is hard to be found. Thus, the deep reinforcement learning method based on policy gradient is selected to establish the SPBDDPG algorithm that can solve the problem. Then, in order to solve the practical problems, the SPBDDPG algorithm is set up with the state and action for iteration, as well as the environment for generating new state and feedback reward value. Finally, an appropriate iteration step is written for the edge computing resource scheduling problem by combining with the original deep reinforcement learning algorithm. We also evaluate the proposed approaches by relevant experiments. The complexity and effectiveness of the results are validated.

Keywords: Edge computing · Offloading decision · Resource scheduling · Deep reinforcement learning

Y.-B. Lin and D.-J. Deng (Eds.): SGIoT 2020, LNICST 354, pp. 3–13, 2021.
https://doi.org/10.1007/978-3-030-69514-9_1

1 Introduction

Since the concept of edge computing was proposed, many optimization algorithms on energy consumption and delay have been proposed for edge computing offloading decision and resource scheduling. In order to improve network performance, some studies (e.g. [1–4]) focus on completing offloading strategy to minimize latency or energy consumption. Partial offloading is more suitable for applications with more stringent latency requirements than full offloading because it takes advantage of the parallelism between smart mobile devices and the cloud. In addition, since wireless networks have limited bandwidth, it makes more sense to migrate some applications rather than all.

Therefore, much work (e.g. [5–9]) has been devoted to partial offloading. Munoz et al. [5] reduced the energy consumption of intelligent mobile devices to the maximum by jointly optimizing the uplink time, downlink time and data size of intelligent mobile devices and the cloud. Huang et al. [6] proposed a dynamic offloading algorithm based on Lyapunov optimization to achieve energy saving. Lorenzo et al. [7] jointly optimized the transmission power and modulation mode to minimize the energy consumption of intelligent mobile devices under delay constraints. Yang et al. [8] jointly studied the division of computing and scheduling of offloading computing on cloud resources to achieve the minimum average completion time of all users. Cao et al. [9] proposed a framework for partitioning and executing data-flow applications to achieve maximum speed.

The combination of dynamic voltage regulation technology and computational offloading provides more flexibility for strategy design. Kilper et al. [10] considered the computing speed of intelligent mobile devices and the optimization of the transmission rate under the Gilbert-Elliott channel to make offloading decision between local execution and full offloading. In order to adapt the environment dynamics more intelligently, many researches will apply deep reinforcement learning to optimize the offloading decision. Chen et al. [11] studied a decentralized dynamic computing loading strategy based on deep reinforcement learning and established an extensible mobile edge computing system with limited feedback. Zeng et al. [12] introduced a model-free method of deep reinforcement learning to effectively manage resources on the edge of the network and followed the design principles of deep reinforcement learning, then designed and implemented a mobile aware data processing service offloading management agent. Based on the additive structure of utility functions and combined the Q function decomposition technique with double DQN, Chen et al. [13] proposed a new learning algorithm for solving the calculation of random loads.

In addition to this, there is a survey in which the computation task models considered in existing research work are divided into deterministic and stochastic [14]. By using Lyapunov optimization method, a solution considering general wireless network and optimizing energy consumption is given in [15]. In [16], a perturbed Lyapunov function is designed to maximize the network utility, balance the throughput and fairness, and solve the knapsack problem on each slot to obtain the optimal unloading plan. On the other hand, the previous research based on Markov decision process model is mainly limited to single user MEC system [17, 18]. And in [19], the paper focuses on the problem of offloading and resource allocation under deterministic task model, in which each user needs to process a fixed number of tasks locally or offloaded to the edge server.

In this paper, we mainly focus on the following works: In the next section, combining the application scenario of deep reinforcement learning and edge computing, a new architecture is proposed. Section 3 introduces an optimization scheme based on deep reinforcement learning. Then, a single-cell multi-user partial-offloading based on deep deterministic policy gradient algorithm is proposed in Sect. 4, followed by simulation and performance analysis in Sect. 5. Concluding remarks and the research prospect are illustrated at the end.

2 System Architecture

Since cellular networks can only manage the communications and not computing, a new computational control entity is required for the computational offloading architecture known as Small Cell Manager (SCM) under project TROPIC proposed on EU FP7 [20].

SCM consists of three modules: operation module, optimization module and offloading module. Based on the offloading of mobile computing architecture, the multiple user equipment (UE) through the calculation of base station, mobile edge server and SCM are linked together in a single-cell multi-user scenario. Base stations use TDMA mode to enable communication and the time is divided into several time slots. Suppose the maximum delay that users can tolerate is set to constraint L_{max}, and the set of devices that the user connects to base station k is as follows:

$$\mathcal{K} = \{1, 2, \ldots, k, \ldots, K\}$$

When UE has a compute-intensive application to process, it sends resource request to the SCM through base station (BS). Afterwards, SCM makes smart decision about whether to migrate an application, and which parts need to be migrated to the edges. Once the SCM has decided to migrate, three phases need to be performed in sequence.

a. UE sends data to BS via uplink channel.
b. MEC receives the calculated offloading data of UE from BS.
c. the results are sent back to UE through the downlink channel.

The specific system architecture is shown in Fig. 1:

The rate at which user device K uploads data to the base station is as follows:

$$R_k = Blog_2(1 + P_{t,k}{h_k}^2/N_0) \tag{1}$$

Where B is the bandwidth of the complex Gaussian white noise channel, N_0 is the variance of the complex Gaussian white noise channel, $P_{t,k}$ is the transmitting power of the user device K, and h_k is the channel gain of the user device K.

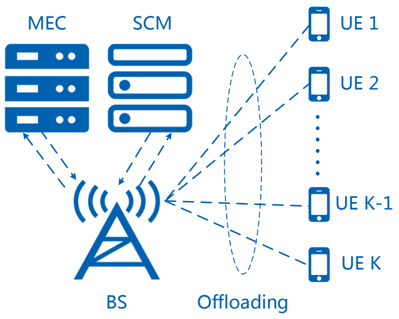


Fig. 1. Single-Cell Multi-User Complete Offloading System Architecture

3 Deep Reinforcement Learning Optimization Scheme

In a single-cell multi-user edge computation offloading scenario, suppose that all user equipment between the base station and the same channel. The application is abstract to a two-parameter profile ($I_{k,}$ C_k), where I_k represents the number of input data bits that user device K computes, and C_k represents the number of CPU cycles required for each bit of input data. The ratio of the calculated offloading data of user device K to the total input data is defined as λ_k ($0 \leq \lambda_k \leq 1$). Therefore, the calculated offloading ratio of all user devices in the cell constitutes the decision vector:

$$\Lambda = [\lambda_1, \lambda_2, \ldots, \lambda_k, \ldots, \lambda_K]$$

The Local Model
Assuming that the local CPU computing capacity of user's device K is $F_k(bits/s)$ and the local energy consumption per CPU cycle is P_k, so user's device needs to meet:

$$(1 - \lambda_k)I_k/F_k \leq L_{max} \tag{2}$$

Therefore, the energy consumption calculated locally by user device K as follows:

$$E_{l,k} = (1 - \lambda_k)I_k C_k P_k \tag{3}$$

The Offloading Model
The time of device K calculates offloading data is t_k, and the total time of all devices in the same time slot needs to less than L_{max}. And the energy consumption by the user device K offloading is as follows:

$$E_{o,k} = \frac{N_0(2^{\frac{\lambda_k I_k/t_k}{B}} - 1)}{{h_k}^2} t_k \tag{4}$$

Cost Model
Considering the parallelization of local and offloading computing, the total energy consumption cost is:

$$E_k(\lambda_k, t_k) = E_{l,k} + E_{o,k} = (1 - \lambda_k)I_k C_k P_k + f(\lambda_k I_k/t_k)t_k \tag{5}$$

As a result, the formulation of edge computing resource offloading scheduling problem can be deduced by changing the decision vector $\Lambda = [\lambda_1, \lambda_2, \ldots, \lambda_K]$ and $T = [t_1, t_2, \ldots, t_K]$ to minimize the total energy consumption $\sum_{k=1}^{K} E_k(\lambda_k, t_k)$. The optimization problem can be mathematically formalized as:

$$\min_{\lambda_k, t_k} \sum\nolimits_{k=1}^{K} [(1-\lambda_k) I_k C_k P_k + f\left(\frac{\lambda_k I_k}{t_k}\right) t_k] \tag{6}$$

$$s.t. C1 : \sum\nolimits_{k=1}^{K} \lambda_k I_k C_k \le F_c,$$

$$C2 : \max\{0, \left(1 - \frac{L_{max} F_k}{I_k}\right)\} \le \lambda_k \le 1,$$

$$C3 : \sum_{k=1}^{K} t_k \le L_{max}.$$

3.1 Parameter Setting

Reinforcement learning is often used to make decisions. It always observes the surrounding environment state and can perform an action to reach another state according to the decision rules, then get a feedback reward.

State
After each scheduling, the total cost of energy consumption is respectively:

$$E_k(\lambda_k, t_k) = \sum\nolimits_{k=1}^{K} [(1-\lambda_k) I_k C_k P_k + \frac{t_k f\left(\frac{\lambda_k I_k}{t_k}\right)}{{h_k}^2}] \tag{7}$$

And the edge server computing capacity required by the actual scheduling scheme:

$$P_n = \sum\nolimits_{k=1}^{K} \lambda_k I_k C_k \tag{8}$$

The state $s(s_1, s_2)$ can be calculated after each action. The system can judge whether each scheduling scheme is to minimize the energy consumption according to s_1. The size of s_2 is used to determine whether the server is fully utilized.

Action
The offloading decision vector $\Lambda = [\lambda_1, \lambda_2, \ldots, \lambda_K]$ with the base station assigning to the user equipment used to upload data time slot decision vector $T = [t_1, t_2, \ldots, t_K]$ as a deep reinforcement learning action for each turn.

Reward
In reinforcement learning, reward is the feedback of state and action, which reflects the impact of the current decision on the result. Therefore, reward is crucial to the outcome of training.

4 SPBDDPG Algorithm

Based on the deep reinforcement learning approach, this chapter further proposes a Single-cell multi-user Partial-offloading Based on Deep Deterministic Policy Gradient (SPBDDPG) algorithm superior to the traditional policy gradient REINFORCE algorithm, which is specifically designed to solve the optimization of user device partial migration in the single-cell multi-user scenario.

4.1 The Principle of Policy Gradient Algorithm

In the strategy gradient based approach, the action is finally executed according to the probability distribution. The strategy is related to the current state, that is, for each states, the strategy gives $\pi(a|s)$, the probability distribution of an action a, and because the strategy needs to be optimized, it parameterizes it by giving the parameter vector, which is denoted as $\pi(a|s,\theta)$.In order to solve the strategy gradient, the introduction of sampling Trajectory $\tau = (s_0, a_0, s_1, a_1, \dots, s_{T-1}, a_{T-1}, s_T)$, represents the sequence of states and actions.Make $p(\tau|\theta)$ denote the probability of the whole trajectory in the case of parameter vector, and the total benefit of the whole trajectory is:

$$R(\tau) = \sum_{t=0}^{T-1} r_t \tag{9}$$

In the practical application, because the expected value cannot be calculated, a large number of samples must be collected, and the approximate expected value can be obtained by taking the mean value:

$$\nabla_\theta E_\tau[R(\tau)] = \frac{1}{N}\sum\nolimits_{n=1}^{N}\sum\nolimits_{t=0}^{T-1} R\big(\tau^n\big)\nabla_\theta log\pi(a_t^n|s_t^n,\theta) \tag{10}$$

4.2 Improved SPBDDPG Algorithm

The actor-critic algorithm, as its name suggests, has two parts: Actor and Critic.

The critic network uses the deep intensive learning algorithm based on value functions, which can output the corresponding $Q^\pi(a_t^n, s_t^n)$ and $V^\pi(s_t^n)$ according to the state s_t and action a_t, while the random variable $A^\theta(a_t, s_t)$ can be expressed as:

$$Q^\pi\left(a_t^n, s_t^n\right) - V^\pi(s_t^n)$$

But it is inconvenient to construct two networks for output $Q^\pi(a_t^n, s_t^n)$ and $V^\pi(s_t^n)$, respectively, and:

$$Q^\pi\left(a_t^n, s_t^n\right) = E[r_t^n + V^\pi(s_{t+1}^n)] \tag{11}$$

Therefore, $A^\theta(a_t, s_t)$ can be approximately expressed as:

$$r_t^n + V^\pi(s_{t+1}^n) - V^\pi(s_t^n)$$

At this point, the critic network can provide actor network with $A^\theta(a_t, s_t)$ weight function for gradient rise and update.

The specific flow of the algorithm is as follows:

a. Expanding scope of action to explore the noise distribution of N and making the neural network output value into the scope of the actual scheduling problem in first initialization.;
b. Then initialize the Actor Evaluate network with the parameter of θ^{π}, and select the action by the strategy gradient; and the Critic Evaluate network with parameters of θ^{Q} to output the Q value corresponding to state actions through DQN;
c. The parameters of the two estimated networks were passed to the target network (Actor Target and Critic Target), and the two groups with same parameters constitute differential action network and evaluation network respectively;
d. Generate action a_1 according to initial state s_1 and action estimation network. Then generate the next state s_2 and value r_1 according to environmental feedback. Take $\{s_1, a_1, r_1, s_2\}$ as a set of data, and store them in memory bank R;
e. Take s_2 as the next initial state and repeat step d;
f. When the memory bank R is full, cover storage is carried out. And after each network output, a large amount of data is randomly sampled from the memory bank R to start training and update the two estimation networks;
g. Generate estimated value q_1 according to s_1, a_1 and evaluate real network; action a_2 is generated according to the next state s_2 and action real network. Then the real value q_2 is generated from s_2, a_2 and evaluation real network. $r_1 + \gamma q_2$ is taken as the comparison real value $q_1{}'$, the mean square deviation of estimated value q_1 and real value $q_1{}'$ is minimized by training the network parameter θ^{Q}:

$$TD\_error = \frac{1}{N}\sum_{N}\left(q_1^{'} - q_1\right)^2 \tag{12}$$

h. The negative number of the estimated value q_1 is defined as the loss function, then update the network parameter θ^{π} through the gradient increase to minimize the loss function. Make the network select the highest probability of Q action;
i. Start the loop from Step c.

5 Simulation and Performance Analysis

This section is based on the Python edge structures in the network environment simulation to achieve the edge computing resource scheduling and tack-offloading system network architecture. In the simulation model, SPBDDPG algorithm is used to optimize the results. For comparison, the REINFORCE algorithm and the minimum offloading scenario are chosen as baselines. Specific simulation parameters are shown in Table 1.

5.1 Complexity

The size of simulation setup memory database is 5000. During the training process, the maximum iteration step is 1000 per round and the number of iterations is 30 rounds. And the sixth round began using memory data to train neural networks, then the new data will

Table 1. Simulation parameters

Parameters	Description	Value
L_{max}	A delay constraint that can be tolerated	100 ms
F_c	Upper limit for edge computing servers	6×10^9 cycles/slot
K	Total number of user devices	30
B	Bandwidth of a complex Gaussian white noise channel	10 MHz
N_0	Variance of complex Gaussian white noise channel	10^{-9} W
h_k	Channel gain for user device K	10^{-3}
F_k	Local CPU capacity of user device K	$\sim U[0.1, 1.0]$ GHz
P_k	Local CPU cycle power consumption of user device K	$\sim U\left[0, 20 \times 10^{-11}\right]$ J/cycles
I_k	Input data for user device K	$\sim U[100, 500]$ KB
C_k	Number of cycles required to compute data for user device K	$\sim U[500, 1500]$ cycles/bit

cover the original database. According to Fig. 2, the reward value curve began to change since the 16th round and the algorithm completed convergence when it comes to the 19th round. In a word, SPBDDPG algorithm has good training effect and convergence rate is rapid.

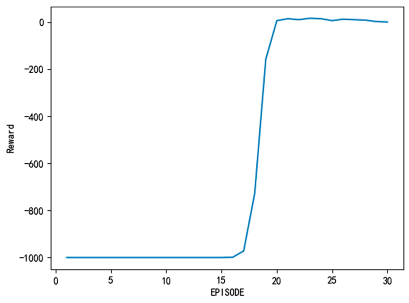
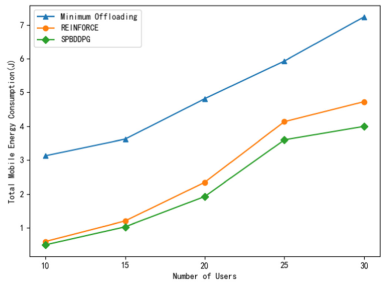


Fig. 2. The convergence of the algorithm

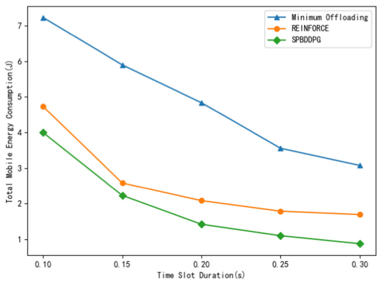
5.2 Effectiveness

With the increase of cell user equipment, the computing power of edge server is limited, and many tasks need to be calculated locally. According to Fig. 3(a), the total energy consumption of the scheduling scheme based on SPBDDPG algorithm is lower than the other two schemes.

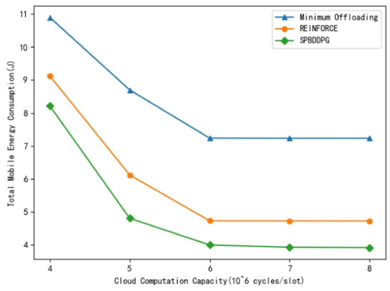
As the maximum of time delay that all users can tolerate increases, many tasks can be calculated by offloading tasks to the edge server. As a result, local computing energy consumption will be reduced. According to Fig. 3(b), compared with the optimization scheme based on REINFORCE algorithm and implement minimum offloading scheme, the total energy consumption of the scheme proposed in this paper is lower.



(a) Impact of user number



(b) Impact of time slot duration



(c) Impact of cloud computation capacity

Fig. 3. The comparison of 5.2 effectiveness between different algorithms

With the increase of the edge server computing capacity limit, many tasks can be offloaded to the edge server for computing, so the total energy consumption of user equipment will be reduced. According to Fig. 3(c), the total energy consumption of the scheduling scheme based on SPBDDPG algorithm is lower than the other two schemes. In addition, the total energy consumption almost stops decreasing and tends to be stable when the edge computing server's upper limit of computing capacity exceeds a certain threshold (about $6 \times 10\hat{}9$ cycles/slot). It indicates that the upper limit of edge computing server's computing capacity has a certain threshold value. If the edge server's computing capacity exceeds the threshold value, the energy consumption of user equipment will not be reduced.

6 Conclusion

In this paper, an offloading decision and resource scheduling optimization algorithm based on deep reinforcement learning is proposed, which can reduce the energy consumption in traditional edge computing problems and improve the resource utilization. Meanwhile, the conventional REINFORCE algorithm combined with DQN was used to model the actual scene of edge computing through SPBDDPG algorithm, and the problem is optimized and solved. Besides, its computational accuracy and complexity are improved compared with other methods. Finally, the advantages of our method are proved by the experimental simulation and services preloading scheme is proved to be effective.

On the other hand, we admit that there are some limitations in this paper. For example, optimization is mainly based on energy consumption and is not applicable to scenarios that are more time-sensitive or integrated. We will concentrate on it in the future. At the same time, some factors should also be considered, such as other deep reinforcement learning algorithms, and it is an important research direction.

Acknowledgement. The authors gratefully acknowledge the support and financial assistance provided by the National Natural Science Foundation under Grant No. 61772064 and 61701019, the National Key R&D Program of China under Grant No. 2018YFC0831900. The authors thank the anonymous reviewers who provided constructive feedback on earlier work of this paper.

References

1. Sardellitti, S., Scutari, G., Barbarossa, S.: Joint optimization of radio and computational resources for multicell mobile-edge computing. IEEE Trans. Signal Inf. Process. Over Netw. **1**(2), 89–103 (2014)
2. Kumar, K., Lu, Y.H.: Cloud computing for mobile users: can offloading computation save energy. Computer **43**(4), 51–56 (2010)
3. Wu, H., Wang, Q., Katinka, W.: Tradeoff between performance improvement and energy saving in mobile cloud offloading systems. In: 2013 IEEE International Conference on Communications Workshops (ICC), pp. 728–732 (2013)
4. Barbarossa, S., Sardellitti, S., Lorenzo, P.D.: Joint allocation of computation and communication resources in multiuser mobile cloud computing. In: 2013 IEEE 14th Workshop on Signal Processing Advances in Wireless Communications (SPAWC), pp. 26–30 (2013)
5. Munoz, O., Pascual-Iserte, A., Vidal, J.: Optimization of radio and computational resources for energy efficiency in latency-constrained application offloading. IEEE Trans. Veh. Technol. **64**(10), 4738–4755 (2015)
6. Huang, D., Wang, P., Niyato, D.: A dynamic offloading algorithm for mobile computing. IEEE Trans. Wirel. Commun. **11**(6), 1991–1995 (2012)
7. Di Lorenzo, P., Barbarossa, S., Sardellitti, S.: Joint Optimization of Radio Resources and Code Partitioning in Mobile Edge Computing (2013). https://arxiv.org/abs/1307.3835.
8. Yang, L., Cao, J., Cheng, H., et al.: Multi-user computation partitioning for latency sensitive mobile cloud applications. IEEE Trans. Comput. **64**(8), 2253–2266 (2015)
9. Yang, L., Cao, J., Yuan, Y., et al.: A framework for partitioning and execution of data stream applications in mobile cloud computing. ACM SIGMETRICS Perform. Eval. Rev. **40**(4), 23–32 (2013)

10. Zhang, W., Wen, Y., Guan, K., et al.: Energy-optimal mobile cloud computing under stochastic wireless channel. IEEE Trans. Wirel. Commun. **12**(9), 4569–4581 (2013)
11. Chen, Z., Wang, X.: Decentralized computation offloading for multi-user mobile edge computing: a deep reinforcement learning approach (2018)
12. Zeng, D., Gu, L., Pan, S., et al.: resource management at the network edge: a deep reinforcement learning approach. IEEE Netw. **33**(3), 26–33 (2019)
13. Chen, X., Zhang, H., Wu, C., et al.: Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning. IEEE Internet Things J. **6**, 4005–4018 (2018)
14. Mao, Y., You, C., Zhang, J., Huang, K., Letaief, K.B.: A survey on mobile edge computing: the communication perspective. IEEE Commun. Surv. Tuts. **19**(4), 2322–2358 (2017)
15. Mao, Y., Zhang, J., Song, S.H., Letaief, K.B.: Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems. IEEE Trans. Wirel. Commun. **16**(9), 5994–6009 (2017)
16. Lyu, X., et al.: Optimal schedule of mobile edge computing for Internet of Things using partial information. IEEE J. Sel. Areas Commun. **35**(11), 2606–2615 (2017)
17. Liu, J., Mao, Y., Zhang, J., Letaief, K.B.: Delay-optimal computation task scheduling for mobile-edge computing systems. In: Proceedings of IEEE International Symposium Information Theory (ISIT), pp. 1451–1455 (2016)
18. Hong, S.-T., Kim, H.: QoE-aware computation offloading scheduling to capture energy-latency tradeoff in mobile clouds. In: Proceedings of 13th Annual IEEE International Conference Sensing, Communication Networking (SECON), pp. 1–9 (2016)
19. Li, J., Gao, H., Lv, T., Lu, Y.: Deep reinforcement learning based computation offloading and resource allocation for MEC. In: Proceedings of IEEE Wireless Communications, Networking Conference (WCNC), pp. 1–6 (2018)
20. FP7 European Project. Distributed Computing, Storage and Radio Resource Allocation Over Cooperative Femtocells (TROPIC) [EB/OL]. https://www.ict-tropic.eu.