Service Placement Considering Robustness and Dynamic in Edge Computing

2021 IEEE 6th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA) | 978-1-6654-2311-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICCCBDA51879.2021.9442568

Yang Liu
School of Software Engineering
University of Science and Technology of China
Hefei, China
lyinfo@mail.ustc.edu.cn

Qianpiao Ma
School of Computer Science and Technology
University of Science and Technology of China
Hefei, China
maqiu@mail.ustc.edu.cn

***Abstract*—With the increasing popularity of the Internet of Things, a huge requirement for low-latency in service response has spurred. Many researches have been proposed on service placement in mobile edge computing (MEC) to address the low-latency requirement, which pushes more service functions from the remote cloud to the network edge. However, they do not consider the scenarios with uncontrolled factors, such as disaster relief and battlefield monitoring, the edge server is likely to go down and fail to respond to the service requests from users. In this paper, we defined the service placement problem in the unreliable and dynamic network and developed a novel distributed service placement decision-making solution called RTSO which based on the greedy algorithm. The edge server using the RTSO algorithm will take into account service popularity and resource constraints, and judiciously adjust service placement decisions. We also verify through experiments that our proposed algorithm can converge to the global optimum through iteration. A large number of simulation results show that our algorithm can provide near-optimal performance and effectively improve the robustness of services in the system.**

***Keywords*—Mobile Edge Computing, Service Placement, Robust, Dynamic**

I. Introduction

The increasing popularity of the Internet of Things (IoT) is driving the development of various applications, such as interactive online games [1], face recognition [2] and VR/AR [3]. In traditional application scenarios, a large amount of application data will be transmitted to the remote cloud platform through the core network, which puts stricter requirements on transmission bandwidth and delay [4]. Unfortunately, the current core network is heavily burdened, causing longer delays for application requests. As a new solution, Mobile edge computing (MEC) [5] has proposed to offload application tasks from terminal devices to nearby edge servers to improve response efficiency, privacy security, and increase bandwidth utilization [6] [7].

The application tasks are usually generated and requested by the user side. Since these devices are often resource-constraint, and the battery and computing capacity of the local devices are limited, it will be resource and time consuming if all the tasks are processed on these devices. As an alternative way, these tasks can be offloaded from devices to edge servers. In recent years, joint optimization of computation offloading and system-level resource allocation for MEC systems has attracted significant research interests [8] [9]. While task offloading has been the central theme of most recent works studying MEC [10] [11], what is equally important is service placement and how these services are placed on servers in the first place. Service placement refers to caching application services and their related databases/libraries in the edge servers, thereby the server can respond to requests from users covered by its communication range.

There have been many researches on service placement. For example, the authors of [12] aim to find optimal locations to cache the data that minimize packet transmissions in wireless sensor nodes. The idea of using caching to support mobility has been investigated in [13], where the goal is to reduce latency experienced by users moving between cells. To cope with the unknown and fluctuating service demand, [14] proposed an online learning algorithm to optimize spatial-temporal dynamic service placement decisions among multiple edge servers to minimize the computation delay. Considering parallel computing at both cloud and edge servers, [15] and [16] studied collaborative service placement and computation offloading to minimize the computation latency.

However, the current research has not considered that in some uncontrolled scenarios, such as disaster rescue and battlefield monitoring, the edge server may go down and cause service requests to fail. In the edge computing scenario, server resources are limited, and it is impossible to deploy all services on each server. Given the above issues, the following key questions should be considered:

- Which services to back up for each server to better improve the service robustness?
- Where to place the service under the resource-constrained scenario?
- How the above decisions can be optimized in a joint manner to improve the service robustness of the system as much as possible?

As we all know, place service on the edge servers allows tasks requiring to be processed at the network edge, thereby reducing computation latency and improving user quality of experience. However, optimal service placement faces many challenges, due to the limited storage space of servers, not all services can be cached at the same time [5]. Therefore, the server has to judiciously decide which services to place and where to place. For example, services are heterogeneous in terms of not only required resources (*e.g.* face recognition and AR/VR services have different resource requirements [6]) but

978-0-7381-0533-8/21/$31.00 ©2021 IEEE

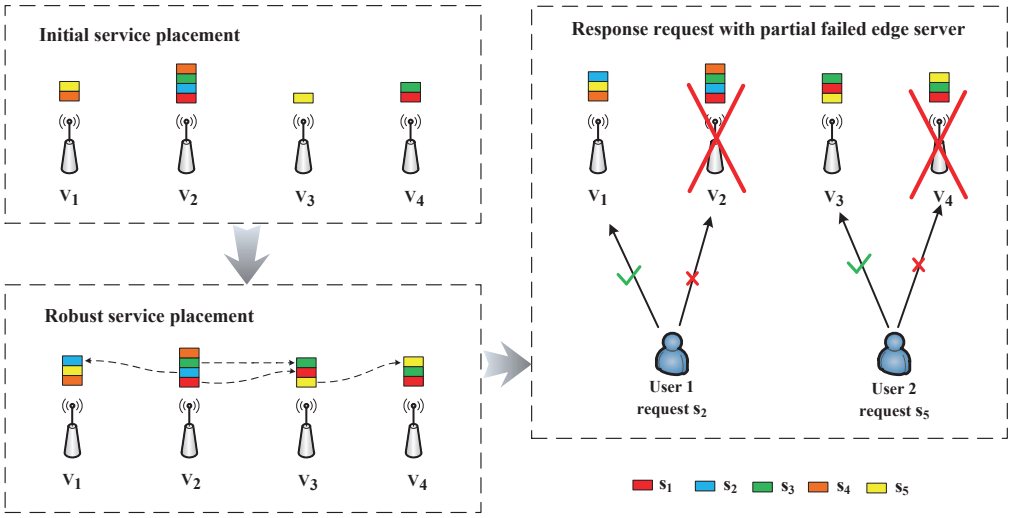

Fig. 1: An example of robust service placement scene.

also popularity among users. Therefore, we define the service placement problem under dynamic and unreliable networks in MEC, and develop an efficient solution that jointly considers service popularity and resource-constraint.

There are several significant superiorities of our algorithm. While service placement mainly concerns with storage capacity constraints, the cost of loss when servers are damaged should not be ignored either. Furthermore, in order to make the deployed services more fully used and improve resource utilization, jointly optimized with task offloading should be considered to maximize the overall system performance. In this algorithm, service request frequency can measure the efficiency of task offloading, cause the more frequently the service is requested, the more frequently the task is offloaded, and the more valuable the deployed service.

The main contributions of this paper are summarized as follows.

1) We propose and formalized the problem of service placement under dynamic and unreliable networks, aiming to improve the robustness of services and enhance the disaster tolerance of the services in the system in the face of emergencies.
2) To solve this problem, we develop a novel service replica placement algorithm, called RTSO (RobusT Service placement for mobile edge cOmputing), which is based on the greedy algorithm, and verify through experiments that our proposed algorithm can converge to the global optimum through iteration.
3) Extensive and practical simulations are carried out to verify the effectiveness of our proposed solutions and algorithms.

The rest of this paper is organized as follows. Section II presents the system model, the concept of robust service placement, an example for motivation and problem formulation. In Section III, we describe the proposed algorithm and its workflow. After that, we give simulation evaluation in Sections IV. Finally, we conclude this paper in Section V.

II. Preliminaries

This section begins with an overview of the system model. Subsequently, we will introduce the concept of robust service placement. Finally, we will formulate the problem.

A. System Model

We consider a MEC system consisting of a set of base stations (BSs), denoted as by $\mathcal{V} = \{v_1, v_2, ..., v_N\}$, with $N = |\mathcal{V}|$. Each BS $v_i \in \mathcal{V}$ has a storage capacity C_i, which can be used to store deployed applications and application data.

A service is an abstraction of an application hosted by BSs, the system offers a library $\mathcal{S}$ of M services to the users, denoted as $\mathcal{S} = \{s_1, s_2, ...s_M\}$, with $M = |\mathcal{S}|$. Typical services include video communication, navigation, and augmented reality. Different services have different popularity of kinds in the real scene, which means services with high popularity will be more frequently requested by terminal devices [17]. To run specific services, related applications need to be deployed on the BSs, such as source code, executable programs, and databases. For each server v_i, the set of services which are deployed on v_i is denoted as S_i. We define a boolean variable $y_{i,j} \in \{0, 1\}$ to indicate whether service s_j is initially deployed on edge server v_i ($y_{i,j} = 1$) or not ($y_{i,j} = 0$). Services may have different requirements in terms of the amount of storage. The storage consumption for each service s_j is denoted as c_j.

B. Robust Service Placement

In the MEC system, each server v_i can communicate with a set of reachable servers, which denoted as $\mathcal{V}_i$. Our objective is to deploy the replicas of each edge server v_i's initial services on $\mathcal{V}_i$ to enhance the robustness of the system. Let $x_{i,j} \in \{0, 1\}$ denote whether service s_j or its replica is deployed on edge server v_i ($x_{i,j} = 1$) or not ($x_{i,j} = 0$).

Therefore, if service s_j's replica can be deployed on server v_i, then at least one of the reachable servers of v_i should have

initial deployed service s_j, and the constraint relationship can be expressed as the following inequality

$$x_{i,j} \leq y_{i',j}, \forall v_i \in \mathcal{V}, v_{i'} \in \mathcal{V}_i \tag{1}$$

On the other hand, the storage resource consumption of the service replicas on server v_i should not greater than its idle resources, which can be expressed as follows

$$\sum_{s_j \in \mathcal{S}} x_{i,j} c_j \leq C_i, \forall v_i \in \mathcal{V} \tag{2}$$

Let p_j denote the profit of the placement of service s_j, this metric measures the popularity of the service s_j within the area covered by the server v_i that initially deploy s_j. Then the total profit of the services on v_i is

$$P_i = \sum_{s_j \in S} x_{i,j} p_j \tag{3}$$

The system robustness index [18] of the service profits among edge servers can be expressed as

$$I = \frac{(\sum_{v_i \in \mathcal{V}} P_i)^2}{N \cdot \sum_{v_i \in \mathcal{V}} P_i^2}. \tag{4}$$

Eq. (4) can quantify the balancing of service profits in the system. If $I = 1$, the profits of all edge servers is equal, and the system achieves totally balance, so it have strong robustness. If the value of I decreases, the system robustness level goes down.

C. An Example for Motivation

In this section, we give an example for robust service placement. As shown in Fig. 1, different services are initially deployed on edge servers v_1, v_2, v_3, and v_4, and each edge server can only deploy service replicas on its neighboring servers (*e.g.*, v_2 can deploy the replicas of services s_1, s_2, s_3, and s_4 to v_1 and v_3). In order to improve the service robustness of the system, considering resource constraints and profit of the service placement, edge server v_2 deploys service s_2 to v_1, services s_1 and s_3 to v_3, respectively. Similarly, edge server v_3 deploys service s_5 to v_4.

Now, due to some uncontrolled factors, for example, network attacks or harsh natural environments, some edge servers failed and can not provide services (*e.g.*, v_2 and v_4) for users. If the user fails to request services from the failed servers, it can send service requests to nearby servers instead. For example, User 1 fails to request service s_2 from v_2, then it can send a request to the nearby server v_1. Since v_2 previously has deployed the replica of s_2 to v_1, v_1 can successfully respond to the service request of User 1.

D. Problem Formulation

Our optimization objective is to determine the service placement decision $\mathbf{x} = \{x_{i,j} | \forall v_i \in \mathcal{V}, s_j \in \mathcal{S}\}$ in the system to improve the robustness of services and enhance the disaster tolerance of the the system under storage resource constraints of edge servers. Therefore, the problem can be formulated as follows

$$\textbf{(P1)}: \max I \tag{5a}$$

$$\textbf{s.t.} \quad (1), (2) \tag{5b}$$

$$x_{i,j} \in \{0,1\}, \forall v_i \in \mathcal{V}, s_j \in \mathcal{S} \tag{5c}$$

$$y_{i,j} \in \{0,1\}, \forall v_i \in \mathcal{V}, s_j \in \mathcal{S} \tag{5d}$$

Algorithm 1 Edge server v_i using RTSO

Input: Server set $\mathcal{V}_i$, consumptions and profits of services $c_j, p_j, \forall s_j \in S$
Output: Final service placement decision $\mathbf{x}$

```
t = 0
while true do
    Requester Thread
    for each v_{i'} ∈ V_i do
        Send SSI contains C_i and d_{i,j}^{i',(t)} to v_{i'}
    Receiver Thread
    Receive C_{i'} and d_{i,j}^{i',(t)} from v_{i'} ∈ V_i
    Calculate x_{i,j}^{(t)} by Eq. (6)
    Calculate C_i^{(t)} and P_i^{(t)} by Eqs. (7) and (8)
    Solve P2 to obtain x_{i',j}^{(t+1)}
    Obtain d_{i',j}^{i,(t+1)} by Eq. (11)
    t = t + 1
```

III. Distributed Algorithm Design

In this section, we will propose a distributed algorithm called RTSO to solve **P1**. The idea of the distributed algorithm is that a edge server constantly exchanges server-status-information (SSI), which contains its storage capacity C_i and its service deployment decision, with its surrounding servers. Then it can obtain the optimal service deployment decision after multiple iterations.

Let $d_{i,j}^{i',(t)}$ be the service placement decision of a neighboring server $v_{i'} \in \mathcal{V}_i$ of v_i in iteration t. If a $v_{i'}$ decides to deploy the duplicate of service s_j on v_i in iteration t, $d_{i,j}^{i',(t)} = 1$, and otherwise $d_{i,j}^{i',(t)} = 0$. $x_{i,j}^{(t)}$ is a boolean variable which indicates whether s_j is deployed on edge server v_i or not, satisfying

$$x_{i,j}^{(t)} = 1 - \prod_{v_{i'} \in \mathcal{V}_i} (1 - d_{i,j}^{i',(t)})(1 - y_{i,j}). \tag{6}$$

Let $\mathbf{x}_{i,\cdot}^{(t)} = \{x_{i,1}^{(t)}, x_{i,2}^{(t)}, ..., x_{i,M}^{(t)}\}$ be the service deployment decision of edge server v_i in iteration t.

Each edge server v_i is both a requester and a receiver. There are two threads running in parallel: requester thread and receiver thread. As a requester, server v_i sends its SSI message, which contains C_i and $d_{i,j}^{i',(t)}$, to all its neighboring server $v_{i'} \in \mathcal{V}_i$ in iteration t. As a receiver v_i, on receiving a SSI message from a neighboring server $v_{i'}$, it calculates the estimated storage consumption of iteration t by

$$C_i^{(t)} = \sum_{s_j \in S} x_{i,j}^{(t)} p_j, \tag{7}$$

and the total estimated deployment profit by

$$P_i^{(t)} = \sum_{s_j \in S} x_{i,j}^{(t)} p_j. \tag{8}$$

Then v_i calculates its *system robustness index* of iteration t by

$$I_i^{(t)} = \frac{(\sum_{v_{i'} \in \mathcal{V}_i \cup \{v_i\}} P_{i'}^{(t)})^2}{N \cdot \sum_{v_{i'} \in \mathcal{V}_i \cup \{v_i\}} (P_{i'}^{(t)})^2}. \tag{9}$$

Accordingly, v_i can obtain its service placement decision of the next iteration by solving **P2** as follows:

$$\textbf{(P2)}: \max I_i^{(t)} \tag{10a}$$

$$\textbf{s.t.} \quad \sum_{s_j \in \mathcal{S}} x_{i',j}^{(t+1)} c_j \leq C_{i'}, \forall v_{i'} \in \mathcal{V}_i \cup \{v_i\} \tag{10b}$$

$$x_{i',j}^{(t+1)} \in \{0,1\}, \forall v_{i'} \in \mathcal{V}_i \cup \{v_i\}, s_j \in \mathcal{S} \tag{10c}$$

If s_j has not deployed on $v_{i'}$in iteration t, and $x_{i',j}^{(t+1)} = 1$, v_i will deploy the duplicate of s_j for $v_{i'}$ in iteration $t+1$, *i.e.*,

$$d_{i',j}^{i,(t+1)} = x_{i',j}^{(t+1)}. \tag{11}$$

The algorithm will run for multiple iterations until a stable service deployment decision is obtained.

IV. Performance Evaluation

A. Simulation Setting

We simulate an edge computing network with 30 edge servers and 9000 terminal devices, which are all random uniformly distributed in a $1km \times 1km$ area. Each edge server establishes topological connections with its nearby servers within a 100m communication range. Each user will call a random service to edge servers within its 100m communication range. If a server receives a user's service request and it has deployed this service, it will respond to the request.

B. Benchmarks and Performance Metrics

We use the centralized algorithm (CA) and non-algorithm (NA) as the benchmarks.

Centralized Algorithm. We assume that the system contains a scheduler which collects the requests of all users and has the whole knowledge of the network information (*e.g.*, resource usage status, service profits). Then it can solve the problem **P1** to obtain the service placement decisions, aiming to achieve the best robustness of services in the system. The simulation result can be regarded as an upper bound of the metrics of our problem.

Non-algorithm. The system that does not use any algorithm has low robustness of services. Therefore, the simulation result can be used as a lower bound of the metrics of our problem.

In this paper, we design service placement decision solutions so as to improve the system robustness index. From the perspective of quality of service (QoS) [19] and quality of experience (QoE) [20], we use the following metrics in our evaluations to demonstrate the efficiency of our solutions.

Request-response Ratio. We define the request-response ratio as the ratio between the actual number of requests and all requested, then we can obtain the request-response ratio of all tasks in the whole system.

Invalid Service Ratio. Invalid service means that the server is damaged due to natural disasters and other factors, so the server cannot provide the corresponding services. We define the invalid service ratio in the system as the ratio between the number of invalid services and all services.

C. Simulation Results

We run two groups of simulations in our evaluation. First, we verify the convergence of our RTSO algorithm and show that RTSO can perform well in dynamic scenarios. Then, we compare our RTSO algorithm with the CA and NA using two performance metrics. Since CA has the whole knowledge of the network, and solves the problem **P1** to provide the optimal solution, therefore, its simulation result can be regarded as a upper bound of the request-response ratio of our problem.

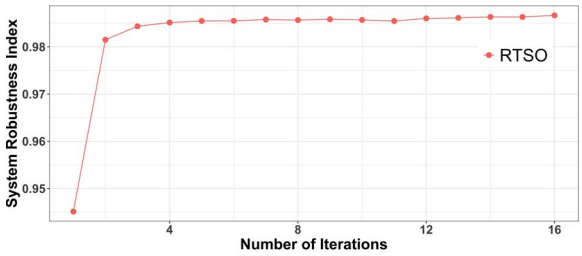

Fig. 2: System Robustness Index vs. Iterations Using RTSO in Stable Network

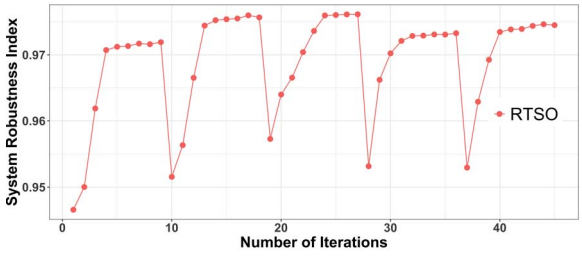

Fig. 3: System Robustness Index vs. Iterations Using RTSO in Dynamic Network

1) Convergence of RTSO: In this section, we simulate the convergence of RTSO algorithm in stable and dynamic networks, respectively.

Fig. 2 shows the changing trend of the system robustness index as the number of iterations increases. In general, with more iterations, the system robustness index of RTSO algorithm will increase and converge to a relatively stable value in a few iterations, because each edge server can adjust service its placement decisions by iterations.

Fig. 3 describes the changes in the system robustness index as the number of iterations increases in a dynamic network environment. To simulate the network dynamic, we changed the number of servers 4 times in 40 iterations. The number of servers increases 20% for in the first and the third times, and reduces 20% in the second and the fourth times. As is shown in the figure, the value of the system robustness index will fluctuate when each the number of servers changes. However, the system robustness index can adaptively converge to a high value again after a few iterations, which indicates that our algorithm is suitable for dynamic networks. For example, the system robustness index falls to 0.951 in the $10th$ iteration because of the change of the number of servers, but the system robustness index bounces back to 0.973 in the $13th$ iteration.

2) Performance Comparison with Benchmarks: In this section, we compare our algorithm performance with the benchmarks in three aspects—system robustness index, request-response ratio and invalid service ratio.

Fig. 4 shows the system robustness index of RTSO, CA, and NA. As shown in the figure, the system robustness indexes

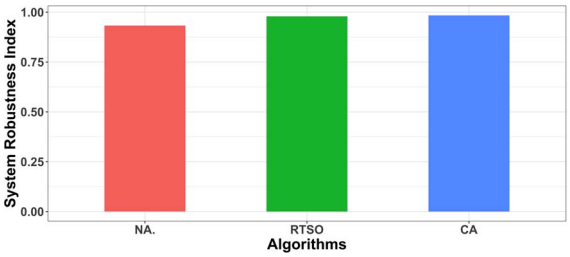

Fig. 4: System Robustness Index vs. Algorithms

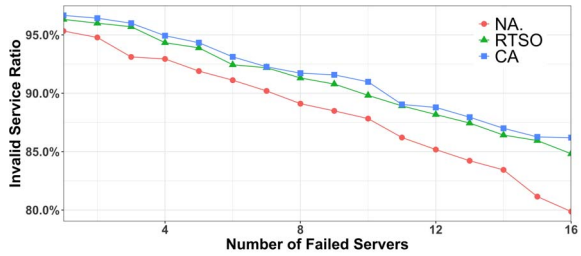

Fig. 6: Invalid Service Ratio vs. Number of Failed Servers

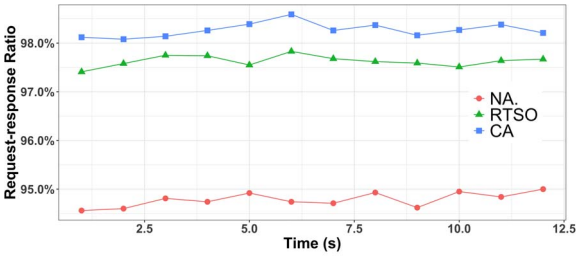

Fig. 5: Request-response Ratio vs. Time

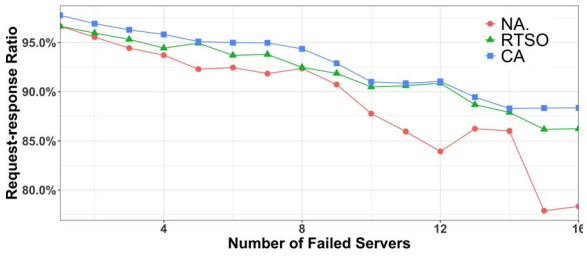

Fig. 7: Request-response Ratio vs. Number of Failed Servers

of CA, RTSO and NA are 0.984, 0.979 and 0.933, respectively. The system robustness index of our RTSO algorithm is approximate close to that of CA.

Fig. 5 shows the change in the request-response ratio of all tasks over time when using RTSO, CA, and NA. As is shown in the figure, there is a clear improvement in the request-response ratio in the network when the algorithm is deployed. In other words, the robustness of the service of the system deployed with algorithms is better, and the users' request-response ratio is higher, which will greatly improve the QoS and QoE of the network. At the same time, our RTSO algorithm has a similar request-response ratio with CA.

Fig. 6 shows the invalid service ratio in the system when servers deploy RTSO, CA, and NA. We test the invalid service ratio when 0-16 edge servers fail, respectively. As is shown in the figure, the invalid service ratio of the system where the algorithm is deployed drops more slowly. We know that the more services deployed in the system, the greater the possibility of users' requests will be responded to. Therefore, the request-response ratio of the system without the RTSO algorithm dropped faster, while this index drops more slowly in networks deployed with the RTSO algorithm and CA, which can be in Fig. 7. At the same time, the results of Figs. 6-7 show that the RTSO algorithm can achieve approximately optimal results compared with the centralized algorithm. In real scenarios, this can give system maintainer and developers more time to maintain and improve the system while ensuring user service experience.

V. Conclusion

In this paper, we have investigated decision-making schemes for service placement considering robustness and dynamic in the MEC network. We propose RTSO, a novel and distributed framework that jointly considers service popularity and resource constraints to improve the service robustness of the system. We verify through experiments that our proposed algorithm can converge to the global optimum through iteration. Extensive simulation results have demonstrated that the proposed solutions can greatly reduce the invalid service ratio and improve the request-response ratio when uncontrollable factors cause damage to the servers, and RTSO can achieve near-optimal service robustness of the system.

References

[1] X. Lyu, H. Tian, W. Ni, Y. Zhang, P. Zhang, and P. R. Liu, "Energy-efficient admission of delay-sensitive tasks for mobile edge computing," *IEEE Transactions on Communications*, pp. 2603–2616, 2018.

[2] Y. Mao, S. Yi, Q. Li, J. Feng, F. Xu, and S. Zhong, "A privacy-preserving deep learning approach for face recognition with edge computing," *Proc. USENIX Workshop Hot Topics Edge Comput.(HotEdge)*, pp. 1–6, 2018.

[3] R.-A. Cherrueau, A. Lebre, D. Pertin, F. Wuhib, and M. J. Soares, "Edge computing resource management system: a critical building block! initiating the debate via openstack," *HotEdge*, 2018.

[4] Q. Fan and N. Ansari, "Application aware workload allocation for edge computing-based iot," *IEEE Internet of Things Journal*, pp. 2146–2153, 2018.

[5] Y. Mao, C. You, J. Zhang, K. Huang, and B. K. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, pp. 2322–2358, 2017.

[6] Y. Mao, J. Zhang, and B. K. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE Journal on Selected Areas in Communications*, pp. 3590–3605, 2016.

[7] R. Roman, J. Lopez, and M. Mambo, "Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges," *Future Generation Computer Systems*, pp. 680–698, 2018.

[8] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, "Mobile-edge computing: Partial computation offloading using dynamic voltage scaling," *IEEE Trans. Communications*, pp. 4268–4282, 2016.

[9] C. You, K. Huang, H. Chae, and B.-H. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Trans. Wireless Communications*, pp. 1397–1411, 2017.

[10] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, pp. 1628–1656, 2017.

[11] J. Ren, H. Guo, C. Xu, and Y. Zhang, "Serving at the edge: A scalable iot architecture based on transparent computing," *IEEE Network*, pp. 96–105, 2017.

[12] S. Prabh and F. T. Abdelzaher, "Energy-conserving data cache placement in sensor networks," *TOSN*, pp. 178–203, 2005.

[13] T. Wang, L. Song, and Z. Han, "Dynamic femtocaching for mobile users," *IEEE Wireless Communications & Networking Conference*, pp. 861–865, 2015.

[14] L. Chen, J. Xu, S. Ren, and P. Zhou, "Spatio-temporal edge service placement: A bandit learning approach," *IEEE Transactions on Wireless Communications*, pp. 8388–8401, 2018.

[15] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," *IEEE INFOCOM*, pp. 207–215, 2018.

[16] L. Chen, C. Shen, P. Zhou, and J. Xu, "Collaborative service placement for edge computing in dense small cell networks," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2019.

[17] G. Li, J. Wu, J. Li, K. Wang, and T. Ye, "Service popularity-based smart resources partitioning for fog computing-enabled industrial internet of things," *IEEE Transactions on Industrial Informatics*, pp. 4702–4711, 2018.

[18] P. N. D. Bukh, "The art of computer systems performance analysis, techniques for experimental design, measurement, simulation and modeling," 1992.

[19] L. Zeng, B. Benatallah, H. A. Ngu, M. Dumas, r. j. kalagnanam, and H. Chang, "Qos-aware middleware for web services composition," *IEEE Transactions on Software Engineering*, pp. 311–327, 2004.

[20] R. Stankiewicz and A. Jajszczyk, "A survey of qoe assurance in converged networks," *Computer Networks*, pp. 1459–1473, 2011.