ScalEdge: A framework for scalable edge computing in Internet of things–based smart systems

Mohammad Babar and Muhammad Sohail Khan

Abstract
Edge computing brings down storage, computation, and communication services from the cloud server to the network edge, resulting in low latency and high availability. The Internet of things (IoT) devices are resource-constrained, unable to process compute-intensive tasks. The convergence of edge computing and IoT with computation offloading offers a feasible solution in terms of performance. Besides these, computation offload saves energy, reduces computation time, and extends the battery life of resource constrain IoT devices. However, edge computing faces the scalability problem, when IoT devices in large numbers approach edge for computation offloading requests. This research article presents a three-tier energy-efficient framework to address the scalability issue in edge computing. We introduced an energy-efficient recursive clustering technique at the IoT layer that prioritizes the tasks based on weight. Each selected task with the highest weight value offloads to the edge server for execution. A lightweight client–server architecture affirms to reduce the computation offloading overhead. The proposed energy-efficient framework for IoT algorithm makes efficient computation offload decisions while considering energy and latency constraints. The energy-efficient framework minimizes the energy consumption of IoT devices, decreases computation time and computation overhead, and scales the edge server. Numerical results show that the proposed framework satisfies the quality of service requirements of both delay-sensitive and delay-tolerant applications by minimizing energy and increasing the lifetime of devices.

Keywords
Computation offloading, clustering, edge computing, Internet of things, scalability, virtual machine

Date received: 30 July 2020; accepted: 29 June 2021

Handling Editor: Lyudmila Mihaylova

Introduction
Edge computing is an emerging computing paradigm, which has completely reshaped the technological landscape of modern-day interactive applications and Internet of things (IoT)-based smart systems. User interest in performing data, computation, and communication using handheld devices such as mobile phones, tablet PCs, and laptops is rising. The computational power of these devices is improving over time, but they are yet to compete with their desktop counterpart. In addition, these devices are resource-limited and their

Creative Commons CC BY: This article is distributed under the terms of the Creative Commons Attribution 4.0 License (https://creativecommons.org/licenses/by/4.0/) which permits any use, reproduction and distribution of the work without further permission provided the original work is attributed as specified on the SAGE and Open Access pages (https://us.sagepub.com/en-us/nam/open-access-at-sage).
performance while working with the traditional cloud environment, acquired high latencies.¹

The IoT plays a crucial role in the design of state-of-the-art infrastructure. These include smart cities, health, intelligent transportation, and smart home, and so on. However, applications related to these systems are data, compute, and communication intensive, which require more power, high processing, and abundant storage. IoT devices are resource-limited, but edge computing overcomes these weaknesses, offering low latency, high bandwidth, and a desirable quality of service.² Edge computing sought a viable solution for IoT devices.³ Figure 1 illustrates cloud-based edge computing architecture.

Computation offloading is one of the main characteristics of edge computing, where the IoT devices off-load a compute-intensive task to the edge server for execution. Nonetheless, computation offloading is a complex task and requires information on what, when, and how to offload. Deploying a computation offloading scheme requires serious attention because it incurs computational, and processing latencies, in addition to network latencies. The IoT-based systems strictly adhere to low latency. It requires serious consideration while designing an efficient computation offloading scheme.

These smart systems are built using a large number of IoT devices. These devices tend to send multiple offloading requests to the edge server. That originates congestion over the edge server that leads to the scalability problem. Scalability is the capability of an edge server to execute a large number of computation offloading requests simultaneously. However, scalability poses several challenges such as overloading back-haul links, misuse of resources, and hinders the performance of an edge server.⁴ The proposed solution comprises the following contributions:

- We have proposed an energy-efficient framework for Internet of things (EEFI) to scale the edge server for IoT-based smart systems. The framework spans over the IoT layer, edge layer, and cloud layer. The inclusion of the cloud strengthens the edge layer to utilize its resource-rich features.
- At the IoT layer, we have introduced the energy-efficient recursive clustering (EERC) for clustering IoT devices that limit the number of offloading tasks sent to the edge server. The offloading decision is made based on the weight value assigned to each offloading request. The EERC clusters the IoT devices when an event occurs and elects a cluster head (CH) following the criteria. The recursive clustering keeps the CH selection dynamic. This approach saves the energy of IoT devices because no CH is permanently designated for the computation offloading process.
- The proposed two-stage computation offloading algorithm (EEFI) performs a seamless offloading process following strict energy and latency the requirement of for each task. First, at the IoT layer, the algorithm ensures whether offloading to the edge server is beneficial or local execution by observing the criteria. Second, at the edge layer, the algorithm ascertains to process delay-sensitive and delay-tolerant tasks over the edge server, under the conventional workload. However, under high workload, the algorithm offloads the delay-tolerant task to the cloud for execution. This approach protects the edge server from congestion and scales it to perform optimally.

The forgoing technique produces desirable results because it considering multiple aspects while dealing with scalability. The framework proposed tackles the scalability both at the IoT layer via the EERC clustering and at the edge layer through the EEFI computation offloading algorithm. Along with the implementation of an efficient resource management technique at the edge layer. However, several studies do not consider the cloud in their edge-based architecture and could not benefit from the limitless resources of the cloud. However, most of the research studies for granted the importance of an effective clustering technique at the IoT layer are discussed in Table 1.

The rest of the article is composed as follows: section “Related work” comprises a literature study, section “EEFI” focuses on the proposed EEFI, the edge–cloud
orchestration, and the computation offloading technique. The discussion and result analysis are presented in section “Results and discussion,” and section “Conclusion” concludes the research.

Related work

Computation offloading is a salient feature of edge computing systems, and resource-constrained IoT devices are likely to reduce latency and energy consumption. The issue overcomes with a computation offloading facility. This facility is acquired via three different techniques:

1. Application offloading, where the complete application or process offloads to the edge server. It is the most common and simplest method. However, it incorporates the complexity of edge server discovery, process migration, context gathering, and remote execution control.

2. Using the component offloading method, an application divides into several portions or methods, whichever is compute-intensive, are offloaded to the edge server. Component offloading encompasses the complexity of partitioning, scheduling, synchronization, and remote execution control.

3. In virtual machine (VM) migration, a VM employs over an edge server that provides resources to users. This model scales the edge server to implement as many VM-based edge servers as demanded and does not encapsulate the complexity of architecture and software.

On a commercial scale, medium to small businesses is growing at enormous speed. They cannot scale their technological infrastructure to meet the needs of tomorrow. Mostly, they cannot afford a separate, centralized data center that requires up-front installation cost, maintenance cost, and up-gradation cost. These businesses are transforming into smart systems, and their infrastructure deployments require a large number of IoT devices to interact. The authors present a component offloading technique based on a deep Q-Learning to handle massive computation offloading requests, to reduce latency and energy consumption in large-scale offloading.

Intelligent systems such as smart cities, health, education, and many more are emerging. However, applications associated with these systems are complex and demand high-performance computing, whereas IoT devices offload compute-intensive tasks to the edge server. Offloading these tasks at random might get the desired quality of service (QoS), latency, and reliability, but receiving simultaneous jobs, especially in peak time, overpowers the edge server with many requests, which results in the scalability issue.

When IoT devices in abundance opt for offloading, the edge server will be overwhelmed with many

| Serial number | Article | RMM | CO | FR/OT | RMoE | CT |
|---------------|---------|-----|----|-------|------|----|
| 1             | Shakarami et al. | CPU, execution time, latency | Yes | Brand and bound optimization | Yes | SIoT |
| 2             | Liu et al. | CPU cycles, latency | Yes | Particle swarm optimization | Yes | No |
| 3             | Budrienė and Zalieckaitė | Execution time, energy consumption, latency | No | Supply chain marketing model | Yes | No |
| 4             | Satyanarayanan | Response time, latency, communication overhead | Yes | SDN-based MEC architecture | Yes | Hierarchical |
| 5             | Rodrigues et al. | Response time, energy consumption, latency | No | Task-specific learning schemes | Yes | Yes |
| 6             | Katsaragakis | Latency, communication overhead | No | MOOP | Yes | Service-based |
| 7             | Babar et al. | Latency, offloading overhead | Yes | NDK and RenderScript app | Yes | No |
| 8             | Benedetto et al. | Offloading overhead, number of tasks, latency | Yes | Markov approximation | Yes | Dynamic |
| 9             | Zhou et al. | Execution time, energy consumption, frequency | Yes | TOFFEE | Yes | No |
| 10            | Ortiz et al. | Number of tasks, latency, energy consumption | No | Scalable architecture | Yes | Energy-aware |

CPU: central processing unit; SDN: software-defined networking; MEC: multi-access edge computing; MOOP: multi-objective optimization problem; NDK: native development kit; TOFFEE: task offloading and frequency scaling for energy efficiency; SIoT: social Internet of Things; RMM: resource management metrics; CO: computation offload; FR/OT: framework/optimization technique; RMoE: resource management over edge; CT: clustering technique.

Table 1. Summary of related work.
offloading requests. These concurrent requests create congestion at the edge server and result in scalability problems in an edge server environment.\(^7\) Scalability is the ability of the system, a network, or a process to handle the growing amount of work.\(^5\) Scalability in edge computing is quite achievable through computation offloading. However, offloading computationally intensive tasks incurs extra energy and latency while interacting with other devices and servers.

The scalability problem is addressed in several ways by researchers. From the perspective of a smart system, the system generates a massive amount of data and requires a quick response to make timely decisions. However, the author implemented a VM migration technique to balance the workload over the edge server. Besides, edge server resources such as central processing unit (CPU), memory, bandwidth, and latency have managed to avoid bottleneck situations over edge servers via distributed market-based resource management algorithm\(^9,10\) and flexible resource management for data and communication-intensive applications.\(^20\)

Edge computing meets the evolving needs of the market. Researchers implemented a low-cost framework for large-scale IoT devices to achieve heterogeneity, scalability, and resiliency.\(^21,22\) The studies facilitate resource-constrained IoT devices to offload complex tasks to the edge server to achieve low latency and energy consumption. Despite this, computation offload incurs communication delay among IoT devices and consumes more energy, and this is where the study lacks in proposing an efficient solution.

Heuristic offloading decision algorithm (HODA) introduced different optimization techniques to achieve low latency and minimize the energy consumption of IoT devices.\(^23\) The selective offloading scheme proposed ensures scalability by reducing network latency and energy consumption.\(^16\) However, it did not consider the time and energy-consume to send back the computation result to the user, which we contemplate in our work.

Clustering is another technique used to deal with the scalability problem. These techniques in several studies show phenomenal results. Adaptive Particle Swarm Optimization Technique clusters IoT devices to reduce their energy consumption by adjusting the inertia weight parameter to find the optimal solution.\(^24,25\) Social Internet of Things (SIoT),\(^14\) a clustering technique, where the devices in IoT cluster are grouped based on co-location, co-services, and co-ownership to limit the number of requests sent to the edge server. It also avoids the bottleneck at the edge.

Geo-clustering,\(^26\) fuzzy clustering (FCBWTS),\(^27\) and multi-clustering\(^28\) scale the edge server architecture, decrease the cost of energy consumption, and handle a large pool of IoT devices effectively. These techniques scale the edge servers but consume extra energy and add more communication overhead.

Emerging computing paradigms such as transparent computing, and opportunistic edge computing (OEC), were proposed and the stakeholders are encouraged to make available their edge resources to deal with scalability. However, there is a trade-off between energy consumption to make computation offload and reducing latency.\(^29,30\) The proposed work scalable edge computing\(^22\) considers the implementation of multi-objective optimization over edge servers for efficient resource allocation.

Our proposed solution spans over a framework. We introduced a recursive clustering technique at the IoT layer presented in section “Clustering of IoT devices.” A resource scheduling technique over an edge server is mentioned in section “The edge–cloud orchestration architecture,” and a two-stage hierarchical computation offloading algorithm that exploits the effectiveness of edge computing and uses cloud in conjunction with an edge for better scalability. A summary of the relevant work is highlighted in Table 1.

### EEFI

This section illustrates the proposed methodology for addressing the scalability problem. It includes how large-scale IoT devices are clustered and how the edge computing provisions the available resources of the edge–cloud orchestration framework. A two-stage computation offloading algorithm performs low latency and minimizes energy consumption. We tried to achieve scalability at both the cluster forming level and the edge orchestration level.

### Clustering of IoT devices

IoT architecture consists of a multitude of devices of heterogeneous types. The instantaneous provision of resources to these devices challenges the resource-limited edge servers. To address the scalability challenge, we have divided these large-scale devices into smaller manageable groups called clusters. These are based on EEFI algorithm. EERC is an event-driven clustering algorithm that clusters the IoT devices when an event occurs. It offers a two-stage clustering. In stage 1, the distance between the IoT devices computed using Euclidean Distance Forming “K” clusters. The recursive nature of the algorithm further divided the “K” cluster into a “j” number of clusters using distance and interval between nodes. Using this criterion, a CH is elected.

The election of CH is made using round-robin (RR) scheduling, and the energy level of the devices is also taken into account. Every node in the cluster receives data for two rounds. The device with the minimum turn-around time, high energy level, and computational power as compared to other nodes inside the cluster is
selected as a CH. The CH will keep on changing whenever re-clustering takes place, and the next node that fits the criteria will be selected as CH. Each device in the cluster sends data to CH. The CH aggregates the data, assigns a weight value to each task, makes an offloading decision, and offloads the task to base station (BS). Figure 2 shows a two-stage clustering design.

The primary advantage of EERC is scalability because it reduces the amount of traffic forwarded to the edge server, which protects the edge server from the bottleneck. The secondary advantage is reducing the energy consumption of the cluster’s devices, as the CH head changes every time. Furthermore, this approach also avoids a single point of failure and increases the lifetime of IoT devices. Data aggregation at CH allows only high-priority tasks to be offloaded to the edge server which efficiently utilizes the back-haul links, reduces communication overhead, and improves reliability.

The edge–cloud orchestration architecture

The edge–cloud orchestration architecture spans over three tiers. Tier-1 consists of smart IoT devices discussed in section “Clustering of IoT devices.” In tier-2, edge servers of limited capability to compute, store, and process are hierarchically placed, and tier-3 consists of a resource-rich cloud based on multiple servers that provide a variety of services in abundance. A client–server communication model is deployed both at smart devices and at the edge server, where smart devices such as mobile phones act as a client and edge acts as a server. A client–server model deployment reduces the communication overhead, and the inclusion of edge server leads to energy efficiency, enhances throughput, high bandwidth, and reduces network overhead.

The workflow of computation offload is illustrated in Figure 3.

The workflow completes in three stages. First, the CHs from different clusters send a request to get the BS information for a successful offloading process. In response, the BS provides the best channel by considering the loss factor, channel overhead, and bandwidth. Second, the smart device prepares for offloading by first, evaluating if, offloading is beneficial or not. If the task requirements are beyond the capability of the device and have a high weight value, the task will be offloaded using an offloading algorithm, discussed in section 4.3. In our work, we have evaluated task requirements based on execution time, bandwidth, energy consumption, and latency that provide enough knowledge for making computational offloads successful.

Sun and Ansari considered only latency, and Nan et al. focused more on energy consumption that only...
provides limited information for such a complex task as offloading. Third, the BS receives the offloading request, forwards it to the edge server to process the compute-intensive task, and sends the results back to the end-user device using the same channel, and then releases the channel as shown in Figure 4.

The CH aggregates the data and makes computational offloading decisions based on equations (1) and (2)

\[
\Omega_{d,j} = \begin{cases} 
1, & \text{if } (M_j < M_d) \text{ and } (w_j < E_{d,j}) \\
0, & \text{otherwise}
\end{cases} \quad (1)
\]

where \( \Omega_{d,j} \) is the computation offload decision parameter, \( \omega_j \) is the time delay of job \( j \), \( M_j \) is the memory required by job \( j \), \( M_d \) is the available memory on device \( d \), and \( E_{d,j} \) is the estimated execution time of job \( j \) on device \( d \). Once the job \( j \) is offloaded to the nearest edge \( e \), then the edge \( e \) makes a further decision whether to offload the given job to the cloud, in case, the server is overloaded or the latency of the task fears to violate, otherwise, the task is locally executed over the edge. The offloading decision is given by

\[
\Omega_{e,j} = \begin{cases} 
1, & \text{if } (w_j < E_{e,j} + Q_{e,j}) \\
0, & \text{otherwise}
\end{cases} \quad (2)
\]

where \( E_{e,j} \) is the estimated execution time of job \( j \) on edge \( e \), and \( Q_{e,j} \) is the queuing delay of job \( j \) on edge \( e \).

The offloading technique

We have implemented two popular computational offload algorithms for mobile edge computing, that is, efficient multi-user computation offload (EMU)\(^{34}\) and a dynamic offloading algorithm (DoA),\(^{35}\) using biased randomization. However, EMU performs well in the environment, where we deal with a static number of users, and dynamic offloading works well when the number of users is not stationary but degrades performance in cellular connectivity. We have combined both algorithms for better performance. The proposed algorithm is based on dynamic programming, which is an efficient optimization technique that transforms complex problems into a sequence of general problems, with randomized strategy.

A client–server model is implemented for simulation purposes, where the BS acts as a server and mobile as a client. The EEFI algorithm is implemented on the client station, which decides to offload the task, or not to offload, unlike the centralized algorithms, where the BS is responsible for making the offloading decision. The EEFI algorithm works well with the static number of users, and the biased randomization works better with users on the move.

Computation offloading is a complex process and requires enough knowledge about the time it takes to complete the task and the total energy consumed by the task. The prior knowledge of these will make the computation offloading decision a lot easier.\(^{36,37}\) In our work, we have calculated the time taken to offload a task, given in equation (3) as follows

\[
T_{\text{off}} = T_u + T_e + T_d \quad (3)
\]

where \( T_{\text{off}} \) is the total time required to offload a task, \( T_u \) is the total time to upload the task, \( T_e \) represents the time taken to execute the task, and \( T_d \) is the download time, when the task is completed and the result is downloaded on to the device. Our computation offloading technique shown reasonably improved performance and more accurate results because we have considered the download time, while studies\(^{35,38,39}\) have neglected the downtime effect.

The time required for computation offload is expressed in equation (4)

\[
T_u = \frac{D_n}{W \log_2 (1 + \frac{P_u \cdot \text{Los}}{N})} \quad (4)
\]

where \( D_n \) is the size of data, which is offloaded to the edge for execution. \( W \) represents the bandwidth of the channel. \( P_u \) is the total power of the device that uploads the task. \( \text{Los} \) is the channel gain, and \( N \) represents the Gauss noise power in the channel. The time
to execute a task over an edge server is expressed in equation (5)

\[ T_e = \frac{C_n}{C_e} \]  (5)

where CE is the execution rate of the edge server. The computation offloading time is obtained by combining equations (4) and (5). However, downloading time has a lesser effect on the computation offloading time because the size of the input is much higher than the output produced. Therefore, the total time to offload a task is given in equation (6)

\[ T_{off} = \frac{D_n}{W \log_2 (1 + \frac{P_{max} L_{off}}{N})} + \frac{C_n}{C_e} \]  (6)

We have got the energy and time consumed by the computation offload. In the next stage, we have illustrated how to determine whether the decision to offload is beneficial or not, depends upon the following conditions

\[ T_{off} < T_{local} \]  (7)
\[ E_u < E_{local} \]  (8)
If the time and energy consumed by the task offloading are lesser than the local execution, the offloading is beneficial otherwise not. The offloading decision is made using equations (1) and (2). When the offloading request is received at the edge tier, it is desirable to allocate the minimum resources to the task by the edge server. The computation offloading problem becomes a binary programming problem, which is solved using cross-entropy-based centralized optimization. This optimization algorithm works well in the current scenario as most of the information is known to BS and the edge server. The EEFI Algorithm 1 presents a detailed description of the offloading process.

**Results and discussion**

This section describes the results achieved through a three-tier EEFI-based smart systems. The performance improvement of the computation offloading algorithm and their effects on latency and energy consumption of devices are discussed. The simulation setup spans over, a BS with a radius of 250 m, following a radio communication specification, adopted in the Third-Generation Partnership Project (3gpp). A 10-GHz edge server is placed in the closed proximity of users. A face recognition application, with variant data size, is introduced as a compute-intensive task along with the latency requirements of 1 s for delay-sensitive, and 1.5 s for delay-tolerant applications, respectively. The devices deployed at the IoT layer carry the processing capability of 0.5–1.5 GHz.

The simulation setup was designed to analyze the performance of EEFI by finding average latency, energy consumption, and the maximum number of computation offloading requests entertained by the edge server. Then, it is compared with the state-of-the-art solutions. Furthermore, the performance improvement of EEFI in local execution and total computation offloading is observed, when the number of...
offloaded requests are increasing from 0 to 50. The offloading tasks are generated in a single-user multitasking manner presented in Figure 5, and the multi-user multitasking manner is shown in Figure 6.

The results exhibit significant improvement in average latency per user in both delay-sensitive and delay-tolerant applications, shown in Figures 5 and 6. A client–server architecture has been introduced in the IoT tier and edge tier, which eases the synchronization in both tiers, and reduces the communication overhead, to meet the latency requirements of delay-sensitive and delay-tolerant tasks. The average latency achieved by EEFI is 0.8 s for delay-sensitive tasks and 1.12 s for delay-tolerant tasks. The average latency in local execution remains under control 0.9 s in normal conditions, satisfying the strict latency requirements for delay-sensitive applications. However, the average latency recorded as 1.1 s in peak hours violates the strict latency requirements. The average latency of the total offloading goes up to 1.6 s when the offloading requests reach 35. The total number of requests entertained by the edge increased by 65% due to recursive clustering at the IoT tier, which protects the server from the bottleneck, by limiting the number of requests sent to the edge server. However, the performance degrades when the number of requests exceeds 35, which is due to the scalability problem. The EEFI yields better results than selective offloading.

Energy consumption per user request and energy consumed in total offloading are described in Figure 7. We observed that EEFI works well with delay-tolerant and consumes less energy (0.070 J), which is 27% lesser than energy consumption of local execution. Although, in the case of delay-sensitive applications, energy

---

**Figure 7.** Energy consumption of selective offloading and EEFI and edge computing trade-off.

**Figure 8.** The comparison of time reduced and energy saved with a different number of devices.

**Figure 9.** Energy and time saved under different data sizes with 75% battery level.

**Figure 10.** Energy and time saved under different battery levels with 30 MB data.
consumption remains low when the number of requests is less than 27, and the energy consumption rises 10% when the number of requests increases to 35. It is because the resource-constrained devices are unable to save energy and offloading some of the requests from the edge to the cloud, in an attempt to protect the edge server from the bottleneck. It also points to the inherent problem of edge computing, where energy and latency cannot be reduced at the same time.

Figure 8 highlighted the energy saved and time reduced with the different number of devices, and Figure 9 shows energy and time reduced with different data sizes. We have observed that the EEFI saves energy and reduces time to an extent, and the deployment of recursive clustering at the IoT layer scales the edge server.

Data sizes range from 13 to 55 MB were applied to test the performance of EEFI. Significant improvement has been noticed under different battery levels of the devices using 30 MB of data size as shown in Figure 10. EEFI outperforms both the computation offloading schemes. However, the reduced computation overhead tends to be steady, as the number of devices increases, which is the trade-off of the proposed EEFI framework.

A total of 7.902% energy was conserved. 18.4 s of computation time reduced, 6.9 s of data transfer time, and a 3.331% transfer of energy cost recorded. Moreover, EEFI reduces computation offloading overhead by 0.554%. That is because the client–server architecture utilizes for offloading compute-intensive tasks to the edge. The architecture abridges synchronization in the offloading process. Furthermore, it increases the lifetime of the devices by 12.58% while offloading a task, which is beyond the capability of IoT devices to the edge.

Conclusion

We have introduced a three-tier energy-efficient framework comprised of cloud, edge, and IoT devices. To solve the scalability problem in edge computing, a recursive clustering technique was deployed at the IoT layer that limits the number of requests offloaded to the edge server and protects the edge from the bottleneck. We propose a two-stage hierarchical computation offloading algorithm that satisfies the latency and energy requirement of the task while performing computation offloading. To ensure the effective utilization of edge resources, the state-of-the-art resource scheduling technique is employed. That assigns minimum edge resources to each offloaded task. This technique scales the edge server further. EEFI has shown improved performance in reducing energy consumption, saves computation time, and extends the battery life of resource-constrained IoT devices. The framework also reduces the computation offloading overhead by adopting client–server architecture for the computation offloading process. The results depict that the EEFI satisfies the requirements of delay-sensitive and delay-tolerant applications and scales the edge server to entertain the maximum number of user requests.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) received no financial support for the research, authorship, and/or publication of this article.

ORCID iD

Mohammad Babar https://orcid.org/0000-0002-7822-0148

References

1. Buyya R and Srirama SN. Fog and edge computing: principles and paradigms. Hoboken, NJ: John Wiley & Sons, 2019.
2. Shi W, Cao J, Zhang Q, et al. Edge computing: vision and challenges. *IEEE Internet Things J* 2016; 3: 637–646.
3. Ai Y, Peng M and Zhang K. Edge computing technologies for Internet of things: a primer. *Digital Commun Netw* 2018; 4: 77–86.
4. Jiang C, Cheng X, Gao H, et al. Toward computation offloading in edge computing: a survey. *IEEE Access* 2019; 7: 131543–131558.
5. Shakarami A, Ghobaei-Arani M and Shahidinejad A. A survey on the computation offloading approaches in mobile edge computing: a machine learning-based perspective. *Comput Netw* 2020; 182: 107496.
6. Liu Y, Yu H, Xie S, et al. Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks. *IEEE Trans Veh Technol* 2019; 68: 11158–11168.
7. Budrienė D and Zalieckaitė L. Cloud computing application in small and medium-sized enterprises.  https://www.thefreelibrary.com/Cloud+computing+application+in+small+and+medium-sized+enterprises-a0347405450
8. Satyanarayanan M. The emergence of edge computing. *Computer* 2017; 50: 30–39.
9. Rodrigues TG, Suto K, Nishiyama H, et al. Cloudlets activation scheme for scalable mobile edge computing with transmission power control and virtual machine migration. *IEEE Trans Comput* 2018; 67: 1287–1300.
10. Katsaragakis E. Distributed market-based resource management of edge computing systems,  https://dspace.lib.ntua.gr/xmlui/bitstream/handle/123456789/48237/katsarakis-diplomatiki.pdf?sequence=1&isAllowed=y
11. Babar M, Khan MS, Ali F, et al. Cloudlet computing: recent advances, taxonomy, and challenges. *IEEE Access* 2021; 9: 29609–29622.
12. Benedetto JI, Valenzuela G, Sanabria P, et al. MobiCOP: a scalable and reliable mobile code offloading solution. *Wirel Commun Mobile Comput* 2018; 2018: 8715294.
13. Zhou W, Fang W, Li Y, et al. Markov approximation for task offloading and computation scaling in mobile edge computing. *Mobile Inform Syst* 2019; 2019: 8172698.
14. Ortiz AM, Hussein D, Park S, et al. The cluster between Internet of things and social networks: review and research challenges. *IEEE Internet Things J* 2014; 1: 206–215.
15. Lin H, Zeadally S, Chen Z, et al. A survey on computation offloading modeling for edge computing. *J Netw Comput Appl* 2020; 169: 102781.
16. Lyu X, Tian H, Jiang L, et al. Selective offloading in mobile edge computing for the green Internet of things. *IEEE Netw* 2018; 32: 54–60.
17. Alam MGR, Hassan MM, Uddin MZ, et al. Autonomic computation offloading in mobile edge for IoT applications. *Fut Gener Comput Syst* 2019; 90: 149–157.
18. Hu J, Jiang M, Zhang Q, et al. Joint optimization of UAV position, time slot allocation, and computation task partition in multi-user mobile-edge computing systems. *IEEE Trans Veh Technol* 2019; 68: 7231–7235.
19. Chen Y, Zhang N, Zhang Y, et al. TOFFEE: task offloading and frequency scaling for energy efficiency of mobile devices in mobile edge computing. *IEEE Trans Cloud Comput*. Epub ahead of print 20 June 2019. DOI: 10.1109/TCC.2019.2923692.
20. Liu J, Wan J, Zeng B, et al. A scalable and quick-response software defined vehicular network assisted by mobile edge computing. *IEEE Commun Mag* 2017; 55: 94–100.
21. Chen T, Barbarossa S, Wang X, et al. Learning and management for Internet of things: accounting for adaptivity and scalability. *Proc IEEE* 2019; 107: 778–796.
22. Kabir MT and Masouros C. A scalable energy vs. Latency trade-off in full-duplex mobile edge computing systems. *IEEE Trans Commun* 2019; 67: 5848–5861.
23. Lyu X, Tian H, Sengul C, et al. Multiuser joint task offloading and resource optimization in proximate clouds. *IEEE Trans Veh Technol* 2017; 66: 3435–3447.
24. Wu J, Cao Z, Zhang Y, et al. Edge-cloud collaborative computation offloading model based on improved particle swarm optimization in MEC. In: *Proceedings of the 2019 IEEE 25th international conference on parallel and distributed systems (ICPADS)*, Tianjin, China, 4–6 December 2019, pp.959–962. New York: IEEE.
25. Babar M, Khan MS, Din A, et al. Intelligent computation offloading for IoT applications in scalable edge computing using artificial bee colony optimization. *Complexity* 2021; 2021: 5563531.
26. Bouet M and Conan V. Mobile edge computing resources optimization: a geo-clustering approach. *IEEE Trans Netw Serv Manage* 2018; 15: 787–796.
27. Guo F, Yu L, Tian S, et al. A workflow task scheduling algorithm based on the resources’ fuzzy clustering in cloud computing environment. *Int J Commun Syst* 2015; 28: 1053–1067.
28. Bruschi R, Davoli F, Lago P, et al. A multi-clustering approach to scale distributed tenant networks for mobile edge computing. *IEEE J Select Area Commun* 2019; 37: 499–514.
29. Ren J, Guo H, Xu C, et al. Serving at the edge: a scalable IoT architecture based on transparent computing. *IEEE Netw* 2017; 31: 96–105.
30. Olaniyi R, Fadahunsi O, Maheswaran M, et al. Opportunistic edge computing: concepts, opportunities and research challenges. *Fut Gener Comput Syst* 2018; 89: 633–645.
31. Akila IS and Subaselvi S. Energy efficient recursive clustering approach for wireless sensor networks. *Int J Electron Commun (AEU)* 2017; 66: 102781.
32. Sun X and Ansari N. Latency aware workload offloading in the cloudlet network. *IEEE Commun Lett* 2017; 21: 1481–1484.
33. Nan Y, Li W, Bao W, et al. Adaptive energy-aware computation offloading for cloud of things systems. *IEEE Access* 2017; 5: 23947–23957.
34. Chen X, Jiao L, Li W, et al. Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Trans Netw* 2016; 24: 2795–2808.
35. Chen Y, Zhang N, Zhang Y, et al. Dynamic computation offloading in edge computing for Internet of things. *IEEE Internet Things J* 2019; 6: 4242–4251.
36. Labidi W, Sarkiss M and Kamoun M. Joint multi-user resource scheduling and computation offloading in small cell networks. In: *Proceedings of the 2015 IEEE 11th international conference on wireless and mobile computing, networking and communications (WiMob)*, Abu Dhabi, United Arab Emirates, 19–21 October 2015, pp.794–801. New York: IEEE.
37. Ningning S, Chao G, Xingshuo A, et al. Fog computing dynamic load balancing mechanism based on graph repartitioning. *China Commun* 2016; 13: 156–164.
38. Cheng K, Teng Y, Sun W, et al. Energy-efficient joint offloading and wireless resource allocation strategy in multi-MEC server systems. In: *Proceedings of the 2018 IEEE international conference on communications (ICC)*, Kansas City, MO, 20–24 May 2018, pp.1–6. New York: IEEE.
39. Opadere J, Liu Q, Zhang N, et al. Joint computation and communication resource allocation for energy-efficient mobile edge networks. In: *Proceedings of the ICC 2019-2019 IEEE international conference on communications (ICC)*, Shanghai, China, 20–24 May 2019, pp.1–6. New York: IEEE.
40. Salem AOA and Shudifat N. Enhanced LEACH protocol for increasing a lifetime of WSNs. *Pers Ubiquit Comput* 2019; 23: 901–907.