Journal of Network and Computer Applications 169 (2020) 102781

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

A survey on computation offloading modeling for edge computing

Hai Lin [a,*], Sherali Zeadally [b], Zhihong Chen [a], Houda Labiod [c], Lusheng Wang [d]

[a] Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education. The School of Cyber Science and Engineering, WuHan University, China
[b] College of Communication and Information University of Kentucky, USA
[c] Department INFRES at Telecom Paris, France
[d] School of Computer Science and Information Engineering, Hefei University of Technology, China

ARTICLE INFO

Keywords:
Computation offloading
Edge computing
Modeling

ABSTRACT

As a promising technology, edge computing extends computation, communication, and storage facilities toward the edge of a network. This new computing paradigm opens up new challenges, among which computation offloading is considered to be the most important one. Computation offloading enables end devices to offload computation tasks to edge servers and receive the results after the servers' execution of the tasks. In computation offloading, offloading modeling plays a crucial role in determining the overall edge computing performance. We present a comprehensive overview on the past development as well as the recent advances in research areas related to offloading modeling in edge computing. First, we present some important edge computing architectures and classify the previous works on computation offloading into different categories. Second, we discuss some basic models such as channel model, computation and communication model, and energy harvesting model that have been proposed in offloading modeling. Next, we elaborate on different offloading modeling methods which are based on (non-)convex optimization, Markov decision process, game theory, Lyapunov optimization, or machine learning. Finally, we highlight and discuss some research directions and challenges in the area of offloading modeling in edge computing.

1. Introduction

1.1. Motivation

In the last decade, we have witnessed an exponential growth and diversity of Internet of Things (IoT) devices, such as wireless sensors, wearable devices, new IoT applications are emerging. Applications, such as face recognition, natural language processing, and augmented reality, on the one hand, generate massive amounts of data. Many of these applications require intensive computation and low latency (Hu et al., 2017; Bedi et al., 2018). If the latency becomes low enough to enable a round-trip delay from terminals through the network back to terminals of approximately 1 ms, human tactile to visual feedback control, will change how humans would communicate around the world, as defined by "tactile internet" (Varsha and Shashikala, 2017). Such a technological improvement disrupts today's cloud computing paradigm. This disruption led to the emergence of a new computing paradigm referred to as Edge Computing (EC) (Shi et al., 2016; Chiang and Zhang, 2016).

Edge computing refers to enabling technologies which allow computation to be performed at the edge of the network, on downstream data on behalf of cloud services and upstream data on behalf of IoT services (Shi et al., 2016). Three related concepts (technologies) are proposed as different paradigms of EC: cloudlets of Open Edge Computing (Satyanarayanan et al., 2009, 2014), Multi-access Edge Computing (MEC) of European Telecommunications Standards Institute (ETSI) (Standard, 2016; Hu et al., 2015), and Fog computing of OpenFog Consortium (Group, 2016; Consortium). Ai et al. (2018) surveys the state of art of these three technologies and compares them in terms of standardization efforts, principles, architectures, and applications.

The benefits that EC brings about compared to local computing or cloud computing includes: in contrast to the local computing, the EC can overcome the restrictions of limited computation capacity on End Devices (EDs). Compared with computation offloading toward the remote cloud, EC can avoid high latencies caused by offloading of certain tasks to the remote cloud (Chen and Hao, 2018). This class depends on the following packages for its proper functioning:

To reap all the benefits that EC brings, some challenges should be addressed. These challenges include architecture design (Marjanović

* Corresponding author.
E-mail address: lin.hai@whu.edu.cn (H. Lin).

https://doi.org/10.1016/j.jnca.2020.102781
Received 7 May 2020; Received in revised form 13 July 2020; Accepted 27 July 2020
Available online 12 August 2020

et al., 2018a), mobility management (Waqas et al., 2019; Kim et al., 2018), security (Alrowaily and Lu, 2018; Esposito et al., 2017), Quality of Service (QoS) and Quality of Experience (QoE) compliant services (Zhang and Wang, 2018), content caching (Ale et al., 2019), and computation offloading (Mach and Becvar, 2017). Among these challenges, computation offloading is a 'connection' between EDs and EC. Thus, it is more or less related to most of the issues associated with EC (Cao et al., 2019). Therefore, computation offloading is considered as a critical challenge in EC. To analyze and optimize computation offloading in EC, offloading modeling which determines the overall EC performance plays a crucial role. However, to the best of our knowledge, there is no survey paper which focuses on computation offloading modeling for EC.

Computation offloading, which offloads computation tasks to EC, consists of a transmission procedure, a remote execution procedure, and a result send-back procedure (Fig. 1). The offloading scheme includes the following key components.

- Task partition: If a task is partitionable, we need to optimally partition the task before offloading (Fig. 1b). Otherwise, the whole task should be offloaded to an EC server (Fig. 1a).
- Offloading decision (task placement): to decide which EC server(s) will execute the whole task or the partitioned task. It is worth noting that the task can also be executed locally.
- Resource allocation: to determine the amount of resources needed. These resources include computing resources, communication resources, and energy which should be allocated for the tasks or components.

These components play an important role in modeling the computation offloading process. The objective of offloading modeling is to find an optimal solution for these components, i.e., how to partition the task, what decision should be made, and how much resources should be allocated, so that the offloading performance is optimized.

1.2. Contribution and organization of this survey

This survey discusses computation offloading and cover topics that focus on offloading modeling, which we argue can provide a complete picture of computation offloading and provide a good insight into its principles.

We summarize our key contributions as follows.

- We present a comprehensive survey of fundamental and recent advances in computation offloading in EC by focusing on offloading modeling. We discuss most of the recently proposed important offloading modeling techniques including (non-)convex optimization, Markov Decision Process (MDP), Lyapunov optimization, game theory, and machine learning.
- Furthermore, this survey highlights some of the open research challenges and discusses the future research directions.

For simplicity, we use the abbreviation 'ED' to represent all types of terminals, such as end device, mobile device, end user and user equipment, which appear in different works. The abbreviation 'EC' is used for all types of edge computing, such as MEC, cloudlet, and fog computing. Table 1 lists all acronyms used in this paper.

Fig. 2 presents the organization of this paper. First, we describe the related surveys used in this work and highlight the differences between our contributions and these past surveys in section 2. Then, we present some important EC architectures and frameworks in section 3. Based on these architectures and frameworks, we classify computation offloading in EC from different perspectives such as offloading flow and offloading scenario in section 4. Next, we discuss some related techniques (mode/model) which are used in offloading modeling in section 5. These techniques include offloading mode, channel model, computation and communication model, and energy harvesting mode. Then, in section 6, we discuss and analyze previous works on computation offloading modeling which is the focus of this survey. In section 7, we highlight some open research challenges related to offloading modeling for EC. Finally, we make some conclusion remarks.

2. Related surveys on edge computing

In this section, first we present previous surveys on EC in general. Then, we discuss surveys which focus computation offloading.

Several surveys (Chiang and Zhang, 2016; Satyanarayanan, 2017) related to EC have been published in the last few years. An early survey on EC (Chiang and Zhang, 2016) presents the motivations behind EC. Similarly, the emergence of EC is discussed in Satyanarayanan (2017). The authors point out that this emergence coincides with three important trends in computing and communication, i.e., Software Defined Networks (SDN), ultra-low latency communication (5G), continuing improvement in the computing capability of EDs. In Shi et al. (2016), the authors present a survey on EC's user cases and challenges. In Roman et al. (2018), the authors focus EC security issues. In Yu et al. (2018) and Porambage et al. (2018), the authors present a comprehensive survey which analyzes how EC improves the performance of IoT networks. In Zhao et al. (2019b), the authors discuss EC from point of view of networking, infrastructures and applications.

Besides the above surveys, there have been several other research works (Hu et al., 2017; Yu, 2016; Taleb et al., 2017; Mukherjee et al.,

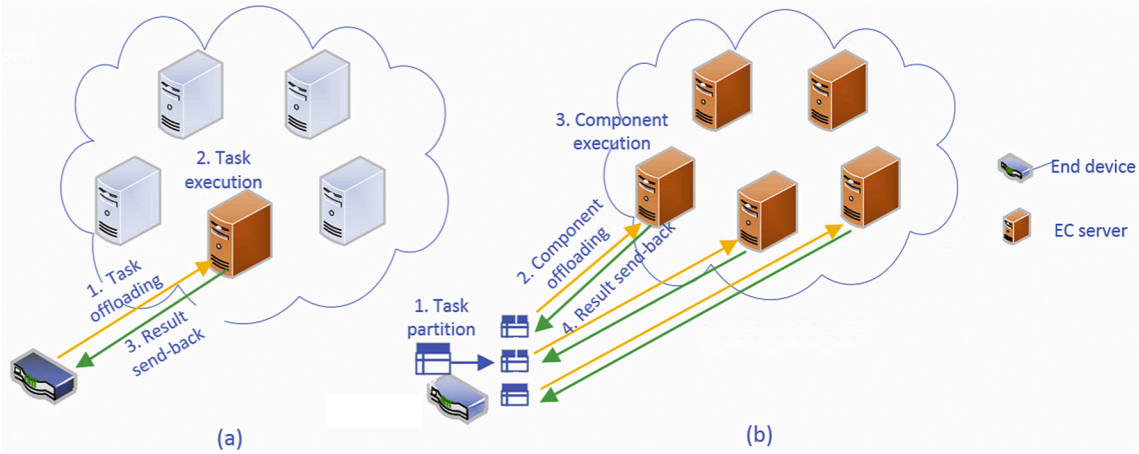

Fig. 1. Offloading procedure. (a) without task partition. (b) with task partition.

Table 1
List of acronyms.

Acronym	Description
ADMM	Alternating Direction Method of Multipliers
AI	Artificial Intelligence
AP	Access Point
ASIC	Application Specific Integrated Circuit
BPSO	Binary Particle Swarm Optimizers
BS	Base Station
CDMA	Code Division Multiple Access
CNN	Convolutional Neural Network
D2D	Device-to-Device
DNN	Deep Neural Network
DQN	Deep Q-Network
EC	Edge Computing
ED	End Device
EH	Energy harvesting
EI	Edge Intelligence
FPGA	Field Programmable Gate Array
GA	Genetic Algorithm
IPM	Interior Point Method
KKT	Karush-Kuhn-Tucker
MBS	Macro Base Station
MCC	Mobile Cloud Computing
MCS	Mobile Crowd Sensing
MDP	Markov Decision Process
MEC	Multi-access Edge Computing
MINLP	Mixed Integer Non-Linear Programming
NFV	Network Function Virtualization
OFDMA	Orthogonal Frequency Division Multiple Access
PDS	Post-Decision State
PSO	Particle Swarm Optimization
QoE	Quality of Experience
QoS	Quality of Service
RL	Reinforcement Learning
SDN	Software Defined Network
SBS	Small cell Base Station
TDMA	Time-Division Multiple Access
UAV	Unmanned Aerial Vehicle
VCTS	Virtual Continuous Time System
VM	Virtual Machine
WPT	Wireless Power Transfer

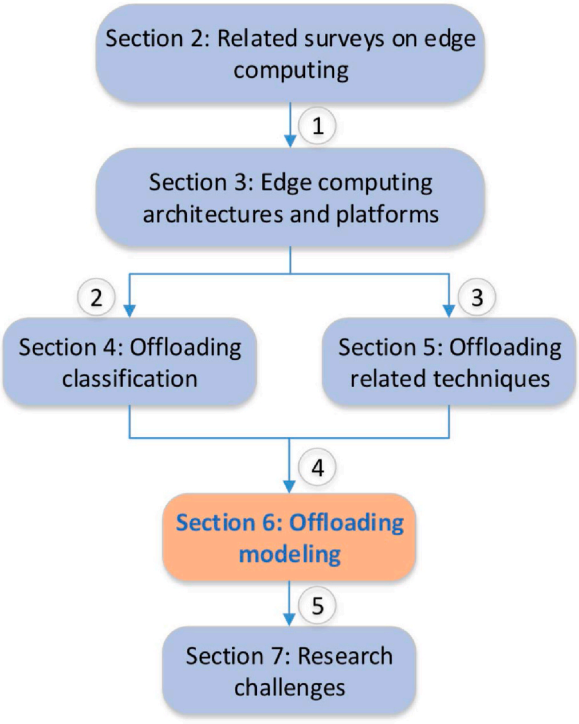

Fig. 2. Paper organization.

2018) which have focused on specific EC technologies such as cloudlets, MEC, and Fog computing. For MEC related surveys, Yu (2016) presents the architectural description of the MEC platform as well as its key functionalities. He also discusses some open research challenges. Taleb et al. (2017) introduce MEC in 5G network focusing on a MEC reference architecture (Sabella et al., 2016) and MEC's fundamental key enabling technologies, such as Virtual Machines (VM), containers, Network Function Virtualization (NFV), and SDN. MEC related research and future directions are highlighted in Abbas et al. (2018). Service migration, which has great potential to address the service continuity issue in MEC, is surveyed in Wang et al. (2018b). Tanaka et al. (2018) focus on the issues of the partitioning and resource management. In Mehrabi et al. (2019), the authors focus on EC mechanisms for computation offloading and mechanisms for caching. Similarly, in Hu et al. (2017) and Mukherjee et al. (2018) the authors present an overview of fog computing architecture, describe some related key technologies and fog computing applications. They also discuss some open research challenges and future directions.

As a key challenge for EC, computation offloading has been discussed in the literature previously. Shan et al. (2018) briefly discuss some offloading algorithms by classifying them according to different goals, such as execution delay minimization, energy consumption minimization, and the tradeoff between energy consumption and execution delay. Wu (2018) presents a related survey for Mobile Cloud Computing (MCC) focusing on the tradeoff between energy consumption and response time based on different offloading decision criteria. Mehrabi et al. (2019) discuss the computation offloading according to the targeted performance goals, which include throughput maximization, latency minimization, energy conservation, and so on. Jiang et al. (2019) analyze the computation offloading from the perspectives of energy consumption minimization, QoS guarantee, and QoE enhancement. In addition, all the above three surveys explore the answers for when, what, where, and how to offload the computation tasks. In Mach and Becvar (2017), the authors discuss the computation offloading works from three perspectives, offloading decision, resource allocation, and mobility management, while the authors in Lin et al. (2019b) discuss these works in terms of application partitioning, task allocation, resource management, and task execution.

All these previous works are important to understand computation offloading in EC, but to the best of our knowledge, there is no survey work that focuses on modeling for computation offloading. This survey highlights the importance of offloading modeling and explore future research trends in this area. Table 2 presents a list of EC related contributions including ours for offloading modeling.

3. Edge computing architectures and frameworks

The previous EC architectures and frameworks include proposed standards such as European Telecommunications Standards Institute (ETSI) MEC (Standard, 2016), OpenFog (Group, 2016), and research proposals such as mobile Edge computing for the Internet of Things (EdgeIoT) (Sun and Ansari, 2016), FLexible Edge Computing (FLEC) (Suganuma et al., 2018). We classify these works into two categories: two-tier and three-tier (or hierarchical) architectures/frameworks. The two-tier architecture (ED-EC) enables EC alone to handle computation tasks while in the three-tier architecture (ED-EC-Cloud) EC works as a complement to cloud computing. Additionally, we overview some EC architectures/frameworks which are designed for specific applications.

3.1. *Two-tier architectures/frameworks*

The early proposed EC framework is MAUI (Cuervo et al., 2010) which is designed for smartphone code offloading. In this framework, three components are defined, a decision engine which determines the cost of offloading, a proxy which handles control and data transfer for offloaded methods, and a profiler which collects measurements of the

Table 2
Comparison of EC surveys.

Survey	Focus		Contributions
Survey (Hu et al., 2017)	Fog computing	• Fog architecture• Application cases	• Key technologies such as computing, communication, and storage
Survey (Shi et al., 2016)	EC in general	• EC definition • Case study	• Challenges and opportunities
Survey (Chiang and Zhang, 2016)	EC in general	• Motivation and benefits • EC use cases	• Architecture and relationship between EC and cloud • Research challenges
Survey (Satyanarayanan, 2017)	EC in general	• Analysis on EC emergence	• Benefits: scalability, privacy-policy enforcement, and so on.
Survey (Roman et al., 2018)	EC security	• Challenges and potential synergies • Security issues	• Security mechanisms
Survey (Yu et al., 2018)	EC for IoT	• Benefits and challenges of EC for IoT	• Architecture, scheduling, and security and privacy for EC based IoT
Survey (Porambage et al., 2018)	MEC for IoT	• MEC for IoT applications • MEC integration to IoT	• Scalability, computation offloading, mobility, security, and so on
Survey (Zhao et al., 2019b)	MEC for IoT	• Networking • Infrastructures	• Applications
Survey (Yu, 2016)	MEC for 5G	• MEC platform description • Key functionalities of MEC	• Content deliver acceleration, content optimization, computation offloading • Open challenges
Survey (Taleb et al., 2017)	MEC for 5G	• key enabling technologies • MEC framework and architecture	• MEC deployment scenarios
Survey (Mukherjee et al., 2018)	Fog Computing	• Network application • Service and resource allocation	• Overview and fundamental of fog computing architecture • Open challenges and research trend
Survey (Sabella et al., 2016)	MEC	• Framework and architecture • Use cases	• Benefits and challenges
Survey (Abbas et al., 2018)	MEC	• Advantages, architecture and application areas • Research direction	• Security and privacy
Survey (Wang et al., 2018b)	Service migration in MEC	• Technologies for hosting services • Service migration in MEC	• Live migration for data centers and handover in cellular networks
Survey (Mehrabi et al., 2019)	Computation offloading and caching	• Offloading performance • Offloading algorithm	• Caching mechanism
Survey (Shan et al., 2018)	Computation offloading in MEC	• Offloading parameter evaluation	• Offloading algorithm description
Survey (Wu, 2018)	Computation offloading in MCC	• Offloading decision making	• Offloading strategies
Survey (Mach and Becvar, 2017)	Architecture and computation offloading	• EC architectures	• Offloading decision, resource allocation and mobility management
Survey (Jiang et al., 2019)	Computation offloading in MCC	• Energy consumption minimization • QoS guarantee	• QoE enhancement
Our work	Computation offloading modeling	• First elaborated analysis on offloading modeling (convex optimization, MDP, Lyapunov optimization, …) • Future research direction on computation offloading modeling	

program's energy and data transfer requirements. These components are implemented on both EDs and EC servers. A similar framework called User-Level Online Offloading Framework (ULOOF) is proposed in Neto et al. (2018). The authors improve the performance of the decision engine component to achieve accurate execution time and energy consumption estimations.

The most important EC architecture/framework is the ETSI MEC (Standard, 2016; Sabella et al., 2016) which defines, from top to bottom, a mobile edge system level, a mobile edge host level and a network level (Fig. 3). The mobile edge host level is the fundamental part of the MEC framework which hosts and manages all edge applications. The mobile edge system level provides an abstraction of the underlying MEC system for end users and third parties whereas the underlying networks level offers connectivity to a variety of accesses including 3GPP mobile networks, local access network and external network such as the Internet.

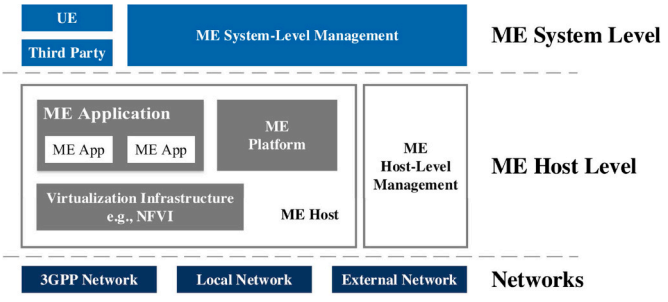

Fig. 3. ETSI MEC framework; ME (Mobile Edge), NFVI (Network Function Virtualization Infrastructure), UE (User Equipment).

Human-driven Edge Computing (HEC) (Bellavista et al., 2018) is designed to facilitate provisioning and to extend the coverage of traditional MEC. In HEC, a traditional EC called Fixed MEC (FMEC) works together with a Mobile MEC (M^2EC) to provide computing services. First, some hot areas of a zone are identified where FMECs are deployed in these areas. In some areas that are populated only during shorter and different timeslots, M^2EC is implemented in the mobile phone of the users who are currently on the site and are willing to host the EC server. With this hybrid deployment, reliable EC service can be achieved in the mobility scenario.

The EC architecture proposed in RDNA (Lorenzo et al., 2018) leverages the advances of mobile devices (e.g. phone, laptop) to provide computing services for users (e.g. sensors), i.e., mobile devices serve as edge servers. These edge servers connect to each other via a cellular base station or via a Device-to-Device (D2D) connection to form an EC platform. The main services provided by the platform include data sharing through which the subscribers that need the same data may collaborate by sharing their data using D2D communications and serve as a backup for each other's connections, and support distributed computing through which mobile devices can share their computing capabilities.

Mobile Ad hoc Cloud (MAC) (Qi and Gani, 2012; Yaqoob et al., 2016) proposes an ad-hoc EC architecture which consists of five components: an application manager, a resource manager, a context manager, a communication manager, and a task offloading manager. Mobile EDs act as both users and EC servers. When a mobile ED wishes to execute a computation-intensive task while its CPU resources are being heavily used, it offloads the task to another mobile ED (or others). Due to the platform's mobility and highly distributed nature, it faces challenges such as resource management, user incentives, security issues. The D2D

fogging platform proposed in Pu et al. (2016) is designed for mobile EDs with a Base Station (BS). Similar to MAC, the computing resources are shared among EDs, but with the assistance of the BS. In this case, the BS schedules the EDs' offloading.

3.2. Three-tier architectures/frameworks

Cloudlets (Satyanarayanan et al., 2014) is considered as the middle tier of a three-tier hierarchy and was designed primarily for cognitive assistance applications. A cloudlet node includes three types of VMs. An ensemble of *cognitive* VMs consists of application VMs which provide computing services such as face recognition, augmented reality; a single *control* VM is responsible for all interactions with sensor devices and assigns computing tasks to different cognitive VMs; a single *user guidance* VM integrates outputs of the cognitive VMs and triggers output for user assistance. The framework with VM has the advantage of flexibility.

The OpenFog consortium defines a hierarchical architecture (Group, 2016; Consortium) (Fig. 4), in which the multiple edge cloud platform introduces a logically hierarchical architecture from the information processing perspective, considering a federation of cloud platforms with the objective to specify fog-to-fog and fog-to-cloud interfaces (Taleb et al., 2017).

In EdgeIoT (Sun and Ansari, 2016), the authors proposed a three-tier EC architecture for IoT based on mobile networks. Computing services are provided by an EC platform which consists of several EC servers. Two functionalities, proxy VM and application VM, are defined. The proxy VM is implemented on the nearby EC severs (connected directly to a BS) serving as a proxy for EDs. The proxy VM collects the raw data streams from its registered EDs and classifies them. The classified data is either sent to a local application VM (EC) or a remote application VM (cloud computing) for each computing service.

In the traditional three-tier hierarchy, the application layer and the service management layer are located at the edge and the cloud. FLEC (Suganuma et al., 2018) extends both layers to 3-tier, and therefore is able to provide flexible EC services. The flexibility is characterized by the environment adaptation ability and user orientation ability of the system. Environment adaptation ability autonomously determines the allocation of processing to an edge or a cloud according to the quality and quantity of works and their fluctuation whereas user orientation ability provides services suitable for each user in real-time by considering the detailed information (such as a user's behavior, intention, and preference) collected by IoT objects.

The hierarchical architecture proposed in Tong et al. (2016) organizes EC servers into different tiers. The lowest tier is called tier one and includes the closest servers to ED. The next tier is called tier two and its next is tier three, until the highest tier *n* which could be traditional cloud computing. The authors argue that the hierarchical architecture can better handle peak loads. In particular, this architecture can opportunistically aggregate and serve the peak loads that exceed the capacities of lower tiers of EC servers to higher tiers.

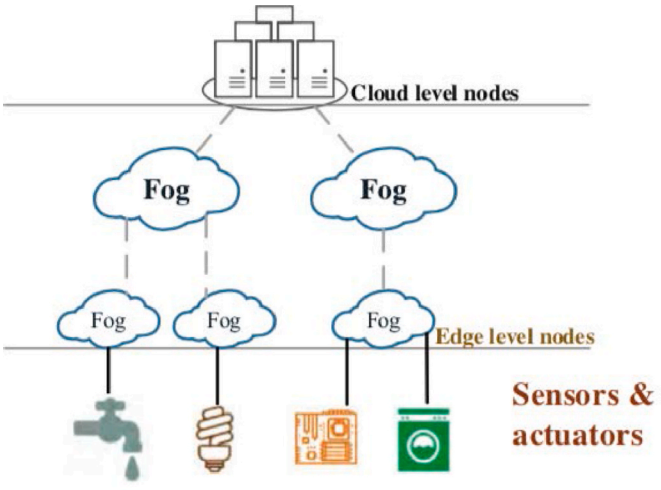

Fig. 4. OpenFog fog computing architecture.

3.3. Architectures/frameworks for specific applications

In the literature, there are also some edge computing architectures/frameworks designed for specific applications. The Stack4Things EC platform (Bruneo et al., 2016) provides users with cloud-based virtualized networking, contextualization, and complex event processing for smart city applications. Al Faruque and Vatanparvar (2016) built an energy management platform over edge computing to provide services for home energy management or microgrid-level energy management. The authors of Mukherjee et al. (2017) propose a fog computing-based framework for industrial applications. The main functionality of this platform is to pre-process the data from sensor devices in order to reduce the traffic overhead and data processing at the remote data center. Consequently, the overall performance is improved.

Docker is an application container. In Ismail et al. (2015), the authors evaluate the performance of Docker when it is used as an EC platform. The authors evaluate functions such as resource and service management, fault tolerance, and caching. They argue that Docker provides fast deployment, elasticity and good performance over VM based EC platform. Waggle (Beckman et al., 2016) developed an open sensor platform capable of leveraging advances in machine learning to support EC. The authors claim that Waggle can address three important needs for wireless sensor platforms: resilience, performance isolation, and data privacy. In Marjanović et al. (2018b), the authors present an EC architecture for hierarchical and large-scale deployments of Mobile Crowd Sensing (MCS) (Guo et al., 2016) services. The goal of such an architecture is to simplify service execution and increase the quality of service, primarily by reducing the latency and data processing complexity.

3.4. Discussion

Two-tier and three-tier architectures can be applied in different scenarios. A two-tier architecture handles all tasks at the edge, so it works well for time-sensitive applications. However, since the devices (servers) at the edge are much less powerful than cloud computing servers, edge processing often requires that several edge servers work collaboratively to provide computing services for EDs. In contrast, the design of the two-tier architecture focuses more on edge server management, performance optimization (e.g. load balancing, execution latency), and application management. A three-tier architecture is suitable for applications which have both time-sensitive tasks and computation-intensive tasks. The time-sensitive tasks are handled at the edge whereas the computation-intensive tasks are executed at the cloud. Therefore, the design of the three-tier architecture focuses more on the interface between the edge and the cloud, and the task assignment (i.e., which tasks are executed at the edge and which are executed at the cloud).

To support EC services into the existing infrastructure, one solution is to integrate them into an existing entity, such as the one described in Lobillo et al. (2014) wherein the authors exploit the computing resources of small cells (SCeNBs) to serve as EC. This solution minimizes deployment cost. However, it would probably impact the entity's original functions, because partial computing/storage resources should be allocated to EC. Hence, this solution is not well suited for the case where the EC needs to provide heavy computing services. Alternatively, some EC architectures introduce a new entity to run EC services (Wang et al., 2013). This way, the EC can provide more powerful computing services, but the deployment cost increases in this case.

The framework of EC is usually layered. For instance, MEC (Sabella

et al., 2016) defines three layers: network layer, host layer and system layer; EdgeIoT (Sun and Ansari, 2016) designs the user data layer and the application layer. The layered framework simplifies development and updating complexity. It also has the advantages of robustness and reliability.

To provide EC services, EC architectures/frameworks include at least two components (or VM): a component for ED interaction (e.g. control VM of Cloudlets) and a component for task execution (e.g. cognitive VM of Cloudlets). Besides these basic components, some EC architectures/frameworks also include a user interaction component (Satyanarayanan et al., 2014), data pre-processing component (Marjanović et al., 2018b), mobility management component (Marjanović et al., 2018b), and an offloading decision component (Cuervo et al., 2010). Components can be added dynamically, which makes EC architectures/frameworks more flexible.

To fully and smoothly integrate EC into the existing infrastructure and provide better a computing performance, a control entity is required. For instance, in Lobillo et al. (2014), the control entity called Small Cell Manager (SCM) is introduced to control the EC platform. Inspired by SDN technology, Jararweh et al. (2016) use SDN-like mechanism to control the edge entities.

Recently, a new computing paradigm called serverless computing emerges (Baldini et al., 2017). Serverless computing refers to the concept of building and running applications that do not require server management. Due to the diversity of EC servers, the server management in EC is more complex than that in the cloud. With serverless computing, the development of EC application becomes a lot easier and therefore, we expect more EC applications will emerge in the future. Moreover, serverless computing describes a deployment model where applications, bundled as one or more tasks, are uploaded to a platform and are then executed. These tasks may be executed on a variety of servers (e.g., X86, ARM) without a tight coordination (Hendrickson et al., 2016). This characteristic matches well with EC wherein various edge devices (e.g., X86, ARM) collaboratively provide computing services. Recently, some works on integrating serverless computing into EC have been undertaken (Baresi and Filgueira Mendonça, 2019; Sarkar et al., 2020). We believe that serverless computing will further promote the development of edge computing.

4. Computation offloading classification

In this subsection, we classify previous works on computation offloading into different categories. In Shan et al. (2018), the computation offloading work is classified based on offloading goals. Shi et al. (2018) classify the computation offloading into static offloading and dynamic offloading. In this paper, we classify computation offloading from two other perspectives (Fig. 5): one is from offloading flow in which the previous offloading schemes could be classified into four categories: 1) offloading from ED to EC; 2) offloading from EC to cloud computing; 3) offloading from one EC server to another (others); 4) hierarchical offloading. The second one is based on offloading scenario, i.e., offloading based on one-to-one scenario, one-to-many scenario, many-to-one scenario, and many-to-many scenario.

4.1. Classification based on offloading flow

4.1.1. From EDs to EC

In the first category, EDs along with EC servers form a whole system, where the EDs execute computation tasks locally or offload them to EC. The authors of Wang et al. (2016) studied a simple scenario with a single ED and an EC server where the ED makes a decision on whether to offload a partial computation task to the EC server. The authors of Chen et al. (2016) and Wang and Zhang (2018) extend the above simple scenario to multi-user computation offloading in the multi-channel wireless contention environment.

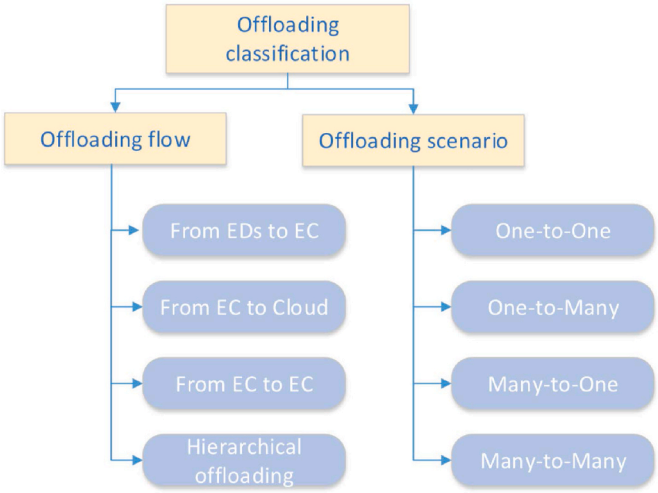

Fig. 5. Offloading classification.

4.1.2. From edge computing to cloud computing

In the second category, EC is considered as a complement to cloud computing. EDs always send their tasks to a nearby EC server which decides whether to execute the received tasks itself or to offload them to the cloud. Although there exists a benchmark that latency-sensitive tasks should be processed by the EC and delay-tolerant tasks should be forwarded to the cloud, optimization of task distribution can still bring performance improvement. In Deng et al. (2015) and Liu et al. (2017), the computation tasks are assigned to cloud/edge computing by minimizing the total power consumption on both sides, while keeping the delay requirement satisfied.

4.1.3. From one EC server to another (others)

In the third category, multiple EC servers form an edge system (or edge cluster). When one EC server receives a computation task, it decides to execute it locally or offload it to other EC server(s) within the same cluster. Since computation offloading is done within a cluster, cluster formation (how to group EC servers) has a direct impact on the offloading performance. Cluster formation in a single ED scenario is carried out in Oueis et al. (2014) to optimize both the execution delay and the power consumption. This work has been extended to multiple-ED scenario in Oueis et al. (2015). Offloading in mobile cloudlet wherein mobile EDs serve as EC servers can be classified into this category (Chen et al., 2015; Hu and Wang, 2015).

4.1.4. Hierarchical offloading

In hierarchical offloading, a computation task could be offloaded to different layers of the underlying computing system. For example, in a three-tier hierarchy, an ED can decide to accomplish a task locally or offload it to EC or to cloud (Guo et al., 2018b; Liu et al., 2018b). If a task is partitionable, it can be divided into four parts: one part for local execution, one part for EC, one part for cloud, and the last part is rejected (Liu et al., 2018a). Hierarchical offloading can also occur in a two-tier system consisting of EC and EDs. An ED could execute computation tasks locally, or offload to another ED, or offload to the EC server, thus forming hierarchical offloading (Jošilo and Dán, 2019; Zhang et al., 2018d).

4.2. Classification based on offloading scenarios

4.2.1. One-to-one

In one-to-one offloading, only one entity (e.g. ED) decides whether to offload a computation task to another entity (e.g. EC server). In Mao et al. (2016a), the authors developed a system consisting of one ED and one EC server to demonstrate how to optimize the offloading

performance. One-to-one offloading has also been studied in Zhang et al. (2018b), but in this case the ED runs multiple applications and each application can offload its data separately, which makes this kind of offloading similar to many-to-one offloading.

4.2.2. One-to-many

In one-to-many offloading, many EC servers are available. The ED decides whether to offload and also decides to which server(s) the task should be offloaded. If the task can be partitioned, the ED needs to distribute the components to multiple EC servers (Shi et al., 2018; Dinh et al., 2017). The one-to-many scenario has also been studied in Le and Tham (2017) where a particular mobile ED can offload its tasks to an ad-hoc mobile EC system consisting of multiple mobile EDs (acting as EC servers).

4.2.3. Many-to-one

In many-to-one offloading, multiple EDs offload their tasks to one server. Modeling this kind of scenario should consider all entities, i.e., the decision should be made with the goal of optimizing the whole system. Since there is only one server, it is quite natural that the server is the decision maker for all EDs. In You et al. (2017), a BS serves as the decision maker. It collects the necessary information from the environment and from all its EDs, and then makes decisions for them.

4.2.4. Many-to-many

The many-to-many offloading scenario is more complex. It is a combination of one-to-many offloading and many-to-one offloading scenarios. If a centralized offloading model is used for the many-to-many offloading scenario, the information from both EDs and EC servers is required for decision making (Chen and Hao, 2018). Though centralized method can achieve the global optimization, it is usually difficult to solve the model. Hence, a distributed method is more suitable to solve the many-to-many offloading problem (Chen et al., 2016).

4.3. Discussion

We classify different computation offloading scenarios to better discuss offloading modeling later. In fact, not all offloading modeling approaches are suitable for all scenarios. For instance, a centralized modeling approach is better for the one-to-one or one-to-many offloading scenarios while a distributed modeling fits well with the many-to-one or many-to-many scenarios. The decision maker is important for offloading modeling. In the one-to-one scenario, it is better for the left 'one' side (i.e., ED) that needs to offload its tasks to make the decision. In either the one-to-many or the many-to-one scenario, it is better for the 'one' side to make the offloading decision for all users if a centralized modeling method is applied. In the many-to-many scenario, it depends on the offloading modeling, i.e., if distributed modeling is applied, the individual users are decision makers, while a central entity should act as a decision maker if the centralized modeling is applied.

5. Computation offloading related techniques

When modeling the offloading problem, we need to consider the following aspects.

- *Offloading mode*: depending on whether a task is partitionable, two offloading modes are defined. In binary mode, the whole task should be offloaded, whereas in partial mode, a partial task can be offloaded.
- *Channel model*: depending on multiple access mode, the channel model is classified into interference or interference-free model with various associated transmission rates.
- *Computing model*: depending on the definition of the computation task and the queue model, the computations of the latency and energy consumption for task execution and task transmission can be different.
- *Energy harvesting model*: depending on the knowledge about energy arrivals, energy harvesting models can be classified as deterministic or stochastic.

We discuss some of the basic techniques (models) related to these aspects. Fig. 6 shows a hierarchical architecture for the techniques discussed.

5.1. Offloading mode

Two computation offloading modes have been defined in EC: binary and partial offloading (Mao et al., 2017). For binary mode, the data set of a task has to be executed as a whole either locally or remotely on an EC server. In the case of partial offloading, task partition is allowed. Hence, a task is first partitioned into several components. These components are then offloaded to EC servers or some components are executed locally while others are offloaded. In practice, binary offloading is easier to implement and it is suitable for simple tasks that cannot be partitioned while partial offloading is favorable for some complex tasks composed of multiple parallel segments (Bi and Zhang, 2018b).

It seems that using exhaustive search can solve the binary offloading problem. For example, in N EDs with one EC server scenario where the EDs need to make a decision on whether to offload a task to the EC server, enumerating all possible decisions can achieve the global optimal objective. However, the number of possible decisions is 2^N. It is hard to solve it when N is large. To reduce the computational complexity, some optimization algorithms or methods should be applied. For example, the Alternating Direction Method of Multipliers (ADMM) decomposition technique (Boyd et al., 2011) is used in Bi and Zhang (2018b) for binary offloading.

There are three types of partitionable tasks (Muñoz et al., 2015): data-partitioned-oriented tasks, code-partitioned-oriented tasks, and continuous-execution tasks. In the context of offloading modeling, we distinguish these partitionable tasks into two classes: parallel or sequential. In the former case, the task is partitioned into components which can be executed in parallel. Thus, the offloading strategy only needs to decide where to place the components. In the latter case, there are dependencies among the components, i.e., some components cannot be executed until the completion of others, so we should define the components' placement as well as their scheduling.

In Wang et al. (2016) and Ren et al. (2017), the authors discuss offloading with parallel components where an ED partitions the task and offloads a part of the components to an EC server. The authors define $\lambda (0 \leq \lambda \leq 1)$ as the ratio of locally executed amount of bits to the total input data bits, i.e., $(1 - \lambda)$ is the ratio of offloaded bits to the total bits. In order to simplify the analysis, they assume full granularity in data partition (Muñoz et al., 2015), i.e., the task could be partitioned into components of any size. To optimize the offloading performance, one critical objective is to find the optimal λ. The authors of You et al. (2017) also performs offloading based on full granularity in data partition and parallel execution of components. An interesting observation is that if Time-Division Multiple Access (TDMA) is used for ED access and the EC server has infinite capability, the partial offloading becomes binary offloading. That is, EDs with priorities above or below a given threshold will perform complete or minimum offloading.

In the sequential case, the relationship of components can be described by a graph (Ryder, 1979), in which each vertex denotes a component and the direct edge represents the size of data migration from one component to another. In Huang et al. (2012) and Deng et al. (2016c), the computation offloading with sequential components is modeled as an optimization problem which aims to maximize the energy efficiency while satisfying the execution time requirement. The dependence on components limits the execution start time of a component,

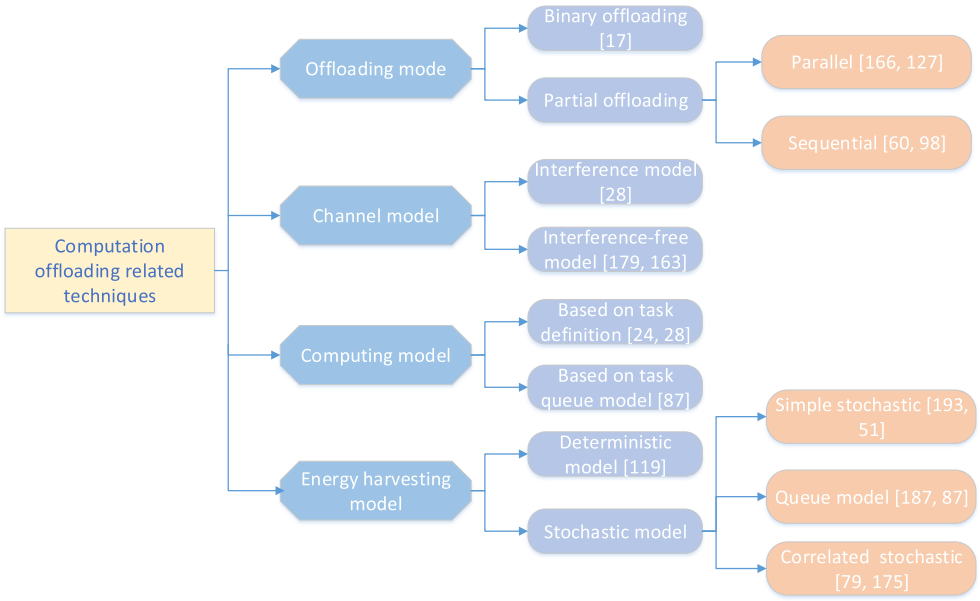

Fig. 6. Offloading techniques.

which serves as a constraint in the optimization model.

5.2. Channel model

Interference and interference-free channel models have been studied for computation offloading. For the interference-free model, e.g. using TDMA or Orthogonal Frequency Division Multiple Access (OFDMA), the transmission rate is calculated as (You et al., 2017; Wang and Zhang, 2018):

$$r_k = B \log_2\left(1 + \frac{p_k h_k^2}{N_0}\right) \tag{1}$$

where B and N_0 are the bandwidth and the variance of complex white Gaussian channel noise respectively; p_k and h_k are ED k's transmission power and its channel gain respectively. In this model, if h_k and N_0 are considered as constant, the function (1) is convex (concave indeed, but we use convex for both convex and concave) in transmission power. With this advantage, we can deduce that the energy consumption function (due to transmission) is convex (You et al., 2017). This enables an optimal solution to be found. If they are not constant, the random processes of white Gaussian channel noise and channel gain should be considered. Therefore, the complexity of the optimization model increases dramatically.

The interference model, e.g. using Code Division Multiple Access (CDMA), is more complex. The transmission rate is calculated as (Chen et al., 2016):

$$r_k = B \log_2\left(1 + \frac{p_k h_k^2}{N_0 + \sum_{i \neq k} p_i h_i^2}\right) \tag{2}$$

where $\sum_{i \neq k} p_i h_i$ denotes the interference from other EDs.

In a hybrid channel model, both interference and interference-free could be used. For instance, in Guo et al. (2018a), an ED can connect to a Macro Base Station (MBS) or a Small cell Base Station (SBS). The communication between the ED and MBS is considered to be interference-free, and therefore (1) is used for the transmission rate calculation. When interference is taken into account for the communication between the ED and SBS, (2) is used for the calculation of the transmission rate.

5.3. Computing model

Energy consumption optimization and latency minimization are two important objectives of computation offloading. They are mainly caused by task execution and transmission. In this subsection, we discuss how execution and transmission models determine these two criteria.

Different definitions of computation task affect the calculation of the two criteria. In Chen and Hao (2018) and Chen et al. (2016), the computation task is defined as (ω_k, s_k) for ED k, where ω_k denotes the total number of CPU cycles required to accomplish the computation task and s_k denotes the size of computation input data (or instructions). The computation latency and energy consumption of local execution of ED k are then defined as a function of ω_k and s_k:

$$t_k^L = \frac{\omega_k}{f_k} \tag{3}$$

$$e_k^L = \rho \omega_k \tag{4}$$

where f_k is ED k' computation capability (i.e., CPU cycles per second) of the executing machine, and ρ is the coefficient denoting the energy consumed per CPU cycle. The transmission latency and the energy consumption of ED k due to offloading are defined as:

$$t_k^T = \frac{s_k}{r_k} \tag{5}$$

$$e_k^T = p_k \cdot t_k^T \tag{6}$$

where p_k is the transmission power of ED k.

Alternatively, Wang et al. (2016) define the computation task as (s_k, τ_k), where τ_k is the latency requirement. The computation latency and the energy consumption of local execution are defined as:

$$t_k^L = \frac{\alpha s_k}{f_k} \tag{7}$$

$$e_k^L = \alpha s_k \cdot \beta f_k^2 \tag{8}$$

where α ($\alpha > 0$) denotes the factor that converts the data size into CPU cycles. βf_k^2 (identical to ρ) represents the energy consumed per CPU cycle which is proportional to f_k^2. Then, t_k^L should satisfy the condition $t_k^L \leq \tau_k$. The two criteria of transmission are calculated identically by using (5)

and (6).

The task description in Zhang et al. (2016a) combines the above two definitions as (ω_k, s_k, τ_k). The latency and energy consumption for execution and transmission are calculated by (3) (4) and (5) (6) respectively, and both latencies should satisfy the constraint $\leq \tau_k$.

In Guo et al. (2018a), the number of CPU cycles required to process one-bit data ϕ_k varies for different tasks. Hence, a task is modeled as (s_k, ϕ_k, τ_k). The total number of CPU cycles ω_k is then calculated by $\omega_k = s_k\phi_k$. Using (3) (4) and (5) (6) we can obtain the latency and the energy consumption respectively for execution and transmission operations.

In contrast to the above work where the calculation of the latency and the energy consumption is based on the definition of task, Liu et al. (2018a) use a queue model for this calculation. The queue in an ED is modeled as $M/M/1$ with an average task generation rate λ and serving rate μ, so the expected local serving latency (waiting time plus computing time) is expressed as:

$$t_k^L = \frac{1/\mu_k}{1-\lambda_k/\mu_k} \tag{9}$$

The local energy consumption is defined in a similar way except that it is calculated for each time slot instead of for each task.

$$e_k^L = \gamma\lambda_k\tau s_k \tag{10}$$

where γ is the computing energy consumption per bit and $\lambda_k\tau$ is the total number of tasks that have arrived during a time slot which has fixed length τ.

The transmission time is also defined for each time slot as:

$$t_k^T = \frac{\lambda_k\tau s_k}{r_k} \tag{11}$$

The energy consumption for transmission is identical to (6).

5.4. Energy harvesting model

Various types of energy sources can be utilized to supplement energy supplies such as solar, wind, vibration, motion, and radio frequency. Energy harvesting (EH) (Ku et al., 2016; Shaikh and Zeadally, 2016) which can capture ambient recyclable energy is a promising technology to address the energy issues which could arise in EC, where grid power supply and battery recharge are considered as unfavorable solutions, or even impossible (Mao et al., 2016a). By integrating EH techniques into EC, satisfactory and sustained computation performance can be achieved. EH models play vital role in designing computation offloading schemes. Based on the availability of non-causal knowledge about energy arrivals, the EH models can be primarily divided into two categories: deterministic (Ozel et al., 2011) and stochastic models (Ku et al., 2016). The deterministic model requires the full knowledge of energy arrival instants and amounts in advance. As a result, this model is only suitable for the applications with the energy sources whose power intensities are predictable. Most studies of energy harvesting in EC focus on the stochastic model (He et al., 2018; Liu et al., 2018a).

A simple stochastic model is a uniform distribution (Zhao et al., 2018a; He et al., 2018). The harvested energy during a timeslot t is uniformly distributed with a maximum value of E_{max}. The advantage of this model is that it is easy to calculate the expectation or variance of the harvested energy which is often required to establish the offloading model.

To develop a stochastic process for the computation offloading system, EH is often modeled as a successive energy packet arrival following a Poisson process (Zhang et al., 2016b; Liu et al., 2018a). In Liu et al. (2018a), the arrival of energy packets is assumed to follow a Poisson process with an average arrival rate λ_e which is constrained by the maximum arrival rate λ_e^{max}, i.e., $\lambda_e < \lambda_e^{max}$. To analyze the relationship between the renewable energy supply and consumption, Zhang et al. (2016b) model the battery as an energy queue, i.e., the harvested energy enters the queue, while the consumed energy leaves the queue. By assuming that the energy arrival follows a Poisson process, together with the deterministic consumption rate, the battery queue is then modeled as a M/D/1 queue for offloading analysis.

The models above are simple but they are inadequate in capturing the temporal correlation properties of the harvested energy for some energy sources. Thus, the correlated stochastic model has been proposed for the EH scenario. A correlated two-state process is proposed in Li et al. (2011), where energy from ambient sources is modeled by two states of 'on' and 'off'. Energy is harvested with a constant rate in the 'on' state and no energy is generated in the 'off' state, and the state probability is correlated with its previous state. In a more complex correlated model, a stochastic process, instead of constant rate, is defined for the 'on' state. For example, the energy arrival is defined as a Bernoulli random process in 'on' state in Michelusi et al. (2013) and Wang et al. (2012). The two-state model reflects some real EH phenomena, e.g. the weather states of solar power harvesting may be cloudy ('off' state) and clear ('on' state). This model is extended to more than two states in Del Testa et al. (2016) and Michelusi et al. (2014).

A similar idea can be found in Xu et al. (2017), where the amount of harvested energy $H(t)$ for each epoch is deduced from some observable environment states $e(t)$. The harvested energy $H(t)$ is then modeled as an i.i.d random variable given $e(t)$, which obeys a conditional distribution $P_H(H(t)|e(t))$.

Radio-frequency EH (also known as Wireless Power Transfer (WPT)), which harvests energy from the ambient electromagnetic waves, has attracted a lot of attention from people working on EC. It is considered as one of the most favorable technologies for supplying continuous power to EDs (Kim et al., 2014). In Wang and Zhang (2018), Bi and Zhang (2018b) and Ji and Guo (2019), each time slot is divided into two phases. During the first phase, ED harvests energy from RF signals emitted by a BS, and then utilizes the harvested energy to execute local tasks and performs partial computation offloading in the second phase. The total energy harvested is modeled to be proportional to the harvesting interval T, transmit power p, and channel gain h (or the square of channel gain):

$$H_t \propto (T, p, h^{(2)}) \tag{12}$$

Zhao et al. (2018c) have investigated radio-frequency EH for resource allocation in EC. But in contrast, besides the harvesting interval, the transmit power, and the channel gain, the mobility of ED is also taken into account. The total energy harvested is modeled as inversely proportional to the square of distance and the model is defined as:

$$H_t \propto (T, p, h, 1 \,/\, d_t^2) \tag{13}$$

where d_t^2 is the distance between the ED and the radio emitter which depends on the mobility of the ED.

No matter which stochastic model is used to capture the characteristics of EH, the goal is to facilitate offloading analysis. Hence, if the established offloading model (e.g. Lyapunov optimization based model) does not require any specific assumption on harvesting model, there is no need to define EH model. This is why some previous work such as (Mao et al., 2016a) which is based on Lyapunov optimization, only requires the process of EH to be identically and independently distributed (i.i.d.).

6. Computation offloading modeling

When making offloading decisions, we need to decide not only whether to offload, but some offloading related actions should be taken into account as well (Chen and Hao, 2018; Wang et al., 2016; Mao et al., 2017; Aazam et al., 2018). We list some main actions for computation offloading below.

1. *Offloading decision*: we need to decide whether to process the task locally or offload it to an edge/cloud computing in the case of binary offloading. If the task can be partitioned, how to partition the task should also be considered.
2. *Server selection*: we select an appropriate edge/cloud server when multiple servers are available (Chen and Hao, 2018). We also consider BS selection as server selection. In Zhang et al. (2016a), an ED takes action on selecting a SBS (or MBS) for task transmission.
3. *Wireless resource allocation*: the amount of wireless resources (e.g. frequency or time) which should be allocated for task transmission should be determined (You et al., 2017). Alternatively, we need to select an appropriate channel which can maximize the data transmission or minimize the interference (Zhang et al., 2016a).
4. *Transmission power setting*: an appropriate transmission power level should be set for task transmission (Guo et al., 2018a).
5. *Computation resource allocation*: we need to allocate computation resources (e.g. CPU cycles, time) for computation tasks, which could be local computation resource allocation in the case of local processing or edge computation resource allocation in the case of offloading (Wang et al., 2016; Bi and Zhang, 2018a).
6. *Slot partition*: it usually occurs in the WPT scenario wherein a time slot in WPT needs to be divided into two phases, one for EH and the other for offloading. Optimal slot partitioning can improve the energy efficiency (Sun et al., 2019; Ji and Guo, 2019).

In addition to the defined actions listed above, there are also some actions which are application dependent. For example, the offloading in Zhou et al. (2018) makes use of Unmanned Aerial Vehicle (UAV)-enabled EC systems wherein the UAV's trajectory management is considered as an action.

By executing some of the actions listed above, one or more computation offloading objectives should be addressed. These objectives include:

1. *Latency minimization*: minimize the average task latency which includes transmission latency and execution latency in the case of offloading (Chen and Hao, 2018).
2. *Energy consumption minimization*: minimize the overall consumed energy, which includes the energy consumed by transmission and execution in the case of offloading (You et al., 2017).
3. *Task dropping minimization*: minimize the dropping of tasks due to a lack of resources. For instance, the task generated by an ED has to be dropped if the ED has no available energy for executing it locally or offloading it to the EC (Liu et al., 2018a).
4. *Computation rate maximization*: maximize the task computation rate under limited resources such as energy and computing resources (Bi and Zhang, 2018a).
5. *Computation efficiency (or energy efficiency) maximization*: maximize the computation efficiency which is defined as the number of total computed bits divided by the energy consumed (Sun et al., 2019).
6. *Payment minimization*: some scenarios assume that the ED has to pay for the resources used in EC or cloud computing, so this payment should be minimized (Liu et al., 2018b).

The offloading problem is essentially an optimization problem, in which the objective functions include one or more objectives defined above, while the variables for the objective functions are the defined actions. For example, EPCO (Wang et al., 2016) formulates the energy consumption minimization as its objective:

$$\min_{f_l, P_t, \lambda} E(f_l, P_t, \lambda) \tag{14}$$

where f_l represents how many computing resources should be allocated, P_t represents the transmission power setting, and λ represents the ratio of locally executed task. The objective is to minimize energy consumption by determining optimal values for these variables.

Next, we study the main previous work related to computation offloading modeling. Most of them include (non-) convex optimization, MDP, Game theory, Lyapunov optimization, and machine learning.

6.1. Convex and non-convex optimization

Convex optimization is a powerful tool for optimization problems because it is solvable. In this model, the objective(s) of offloading is(are) formulated as an objective function and the limitations of offloading are formulated as constraint functions. If the formulated optimization model is convex, classical methods such as the Lagrange duality method and Karush-Kuhn-Tucker (KKT) conditions can be applied to solve the model and achieve the global optimization objective.

If the offloading model is a non-convex optimization problem, a usual way is to transform it to a convex optimization. To do this, we can usually fix some variables to make the objective function over the remaining variables a convex optimization problem. This multiple-phase solution (decoupled solution) is based on the following formula (Wang et al., 2016):

$$\min_{x,y} f(x,y) = \min_{x} \widetilde{f}(x) \tag{15}$$

where $\widetilde{f}(x) = min_y f(x,y)$.

The (non-)convex optimization model consists of one or more objective(s) and some constraints. By determining some variables, the objective(s) is (are) optimized. Table 3 presents most of the offloading schemes discussed in this subsection, where the schemes are compared in terms of offloading mode, optimization parameters (variables), main constraints, and optimization objective(s).

MULTIUSER MECO (You et al., 2017) applies partial offloading aiming at minimizing the energy consumption of EDs. The energy consumption consists of two parts, one resulting from local computing and the other part resulting from data transmission due to offloading. In MULTIUSER MECO, the energy consumption for local computing is a linear function of data volume (local execution data), while the energy consumption for data transmission is an exponential function of data volume (transmission data) and transmission duration. Since both parts are convex, the sum of the two parts keeps convexity. To solve this convex problem, the authors first assume that the EC server has infinite computation capacity, and then apply KKT conditions to obtain the optimal value. To reduce the complexity arising in the case of finite server capacity, the authors propose an algorithm based on the approximated offloading priority order.

EPCO (Wang et al., 2016) focuses on partial offloading. However, compared with MULTIUSER MECO (You et al., 2017), more parameters (as shown in Table 3) are used for optimization modeling. This makes the formulated model a non-convex optimization problem. To transform this model into a convex problem, first the optimal value for local computation capability f_l is determined by noting that the energy consumption increases monotonically with f_l. Then, the authors demonstrate that rest of the problem is a convex optimization problem, and it is therefore solvable.

SDTO (Chen and Hao, 2018) achieves binary offloading by using multiple EC servers and the goal is to minimize the overall latency. So, the objective function is the sum of the latencies of all tasks, which is formulated as the latency of local processing ($x_i = 1$) or offloading ($x_i = 0$):

$$\sum_{task\ i} \left[x_i t_i^L + (1 - x_i) t_i^E\right] \tag{16}$$

where t_i^L is the local processing latency (fixed) and t_i^E is the offloading latency (which varies). The formulated offloading problem is a Mixed Integer Non-Linear Programming (MINLP) problem which is NP-hard. To solve it, the original problem is divided into two sub-problems. First, the authors only consider the optimization problem of

Table 3
Computation offloading schemes based on (non-)convex optimization.

Offloading schemes	Offloading mode	Optimization parameters	Main constraints	Optimization objective(s)
MULTIUSER MECO (You et al., 2017)	Partial	Task partition Offloading duration	EC computing capability constraint Total offloading time constraint	Energy consumption minimization
EPCO (Wang et al., 2016)	Partial	Task partition Transmission power setting Computing resource allocation	Latency constraint ED transmission power constraint ED computing capability constraint	Energy consumption minimization
SDTO (Chen and Hao, 2018)	Binary	Task placement (multiple EC servers) Computing resource allocation	ED computing energy constraint ED transmission energy constraint EC server computing capability constraint	Latency minimization
HGPCA (Guo et al., 2018a)	Binary	Task placement Channel assignment Transmission Power setting EC server computing resource allocation	Latency constraint EC server computing capability constraint Channel assignment constraint	Energy consumption minimization
Scheme with EC-WPT in Ji and Guo (2019)	Binary	Time slot partition	Time allocation constraint Transmission rate constraint Energy harvesting constraint	Energy efficiency maximization
Scheme with EC-WPT in Hu et al. (2018)	Partial	Time slot partition Transmission power setting	Time allocation constraint Transmission rate constraint Energy harvesting constraint	Transmission energy minimization
Scheme with EC-WPT in Wang et al. (2018a)	Partial	Task partition Energy transmission Computing resource allocation Time slot partition	Latency constraint Energy harvesting constraint	Energy consumption minimization
Scheme with EC-WPT in Bi and Zhang (2018a, 2018b)	Binary	Task placement Time slot partition ED computing resource allocation	Time allocation constraint Energy harvesting constraint	Computation rate maximization
Scheme with UAV (Zhou et al., 2018)	Binary/ Partial	Task placement (for Binary case) ED computing resource allocation Transmission power setting UAV trajectory management Time slot partition	Energy harvesting constraint Time allocation constraint UAV speed and location constraint	Computation rate maximization
Scheme in Xiao and Krunz (2017)	Partial	Task partition	Power efficiency constraint	Response-time minimization
Scheme in Liu et al. (2018b)	Partial	Offloading probability Transmission power setting	ED computing capability constraint EC server computing capability constraint Transmission rate constraint	Energy consumption minimization Latency minimization Payment cost minimization
Scheme in Dinh et al. (2017)	Partial	Task placement ED computing resource allocation	ED computing capability constraint Task placement constraint	Energy consumption minimization Latency minimization

computing resource allocation by fixing all other parameters. This sub-problem is a convex problem and can be solved by the KKT condition. Then, with the optimal resource allocation obtained, the sub-problem of offloading decision is formulated as a 0–1 integer programming. Similar to Liu et al. (2016b), 0–1 integer programming is translated into a convex problem to have an approximate optimal solution.

HGPCA (Guo et al., 2018a) also utilizes binary offloading. The formulated model is still a MINLP problem. HGPCA considers more parameters than SDTO (as shown in Table 3), which makes it a large-scale MINLP problem and impossible to solve it with a determined algorithm. Therefore, the authors combine two heuristic algorithms, Genetic Algorithm (GA) (Deep et al., 2009) and Particle Swarm Optimization (PSO) (Yiqing et al., 2007). This is because GA is powerful in searching the global domain while PSO is more accurate when searching locally.

EC with WPT (EC-WPT) has been attracting a lot of attention from academic community recently (Ji and Guo, 2019; Hu et al., 2018; Wang et al., 2018a; Bi and Zhang, 2018a, 2018b). The works in Ji and Guo (2019) and Hu et al. (2018) focus on the scenario where two EDs can offload computation tasks to an EC server. In such scenario, the ED closer to power beacon (access point) experiences better energy harvesting and information transfer than the further one, which is called doubly near-far problem. To address this problem, cooperative communications in the form of relaying via the nearer mobile device is used for offloading. In the WPT scenario, since wireless frequency is used for both energy harvesting and data transmission, an optimal time slot partition is the main factor which affects the performance. Ji and Guo (2019) demonstrate that the established offloading model is a quasi-concavity optimization problem. By applying Dinkelbach's method, the quasi-concavity optimization is transformed into a convex optimization problem. Finally, using classical Lagrangian method, the optimal solution can be obtained. Hu et al. (2018) develop an offloading model to minimize the Access Point's (AP) transmit energy (an ED offloads computation tasks to an AP connected to an EC server), and transform the model into an equivalent min-max optimization problem which is solved by a two-phase approach.

The authors of Wang et al. (2018a) study Multiuser EC-WPT and its objective is to minimize the energy consumption on EC. To solve the

formulated non-convex energy minimization problem, they use the Lagrange duality method and obtain an optimal solution in a semi-closed form.

In Bi and Zhang (2018a, 2018b), the authors address the computation rate maximization problem by selecting an offloading decision and partitioning time slot. To solve the established non-convex model, the authors decouple the formulated model. The decision selection is assumed to be given and they proposed a simple bi-section search algorithm to obtain the optimal time partition. Additionally, they developed a coordinate method to optimize the decision selection. To address the complexity issue in large-scale networks, they proposed an ADMM-based method, in which the principal idea is to decompose the original problem into parallel small-scale integer programming sub-problems, one for each ED.

The authors of Zhou et al. (2018) propose an offloading scheme with UAV which is also based on WPT, but in a special scenario where the UAV serves as an EC server and a WPT supplier. Besides CPU frequency and transmission power, the work also optimizes the UAV trajectory and offloading time. This makes the optimization model they have developed non-convex. Likewise, they use multiple-phase optimization. First, the variable of UAV trajectory is set to be fixed and the model over all other variables is then a convex optimization problem. Thus, optimal values for CPU frequency, transmission power, and offloading time are obtained. However, the objective function over the remaining variable, UAV trajectory, is still non-convex. But since only one variable needs to be solved, the complexity is significantly reduced. The authors use the Successive Convex Approximation (SCA) method to solve the problem and the solutions can be guaranteed to satisfy the KKT conditions of the original problem.

In Xiao and Krunz (2017), the authors propose a distributed ADMM-based method where an EC server can offload the received tasks to the cloud computing or other EC servers (hierarchical offloading). The formulated non-convex optimization problem is decomposed into sub-problems each of which can be solved by each EC server. The authors argue that with, the distributed property, each EC server only needs its own information for the solution, so the privacy and performance improvement are achieved.

Liu et al. (2018b) jointly optimize energy consumption, delay, and payment cost. This multi-objective optimization is first transformed into a single-objective optimization problem (non-convex). Then, the Interior Point Method (IPM) is applied to solve the transformed optimization problem. Dinh et al. (2017) address the one-to-many offloading aiming to minimize both the total tasks' execution latency of all the tasks and the ED's energy consumption. The formulated model is NP-hard and SemiDefinite Relaxation (SDR) based algorithm is proposed to find the near optimal solution.

Next, we summarize other offloading works based on (non-) convex optimization. The computation offloading problems in Tran and Pompili (2018) and Du et al. (2018) are modeled as MINLP, in order to jointly optimize the offloading decision along with power allocation and computing resource allocation. In Deng et al. (2016b), the computation offloading aims to minimize the energy consumption of EDs. They solve the formulated non-convex problem using an approximate approach by decomposing it into three sub-problems. Sardellitti et al. (2015) also formulate a non-convex optimization problem in order to minimize energy consumption for computation offloading, and use a distributed method called successive convex approximation techniques (Scutari et al., 2017) to solve the problem. Tong et al. (2016) proposed an offloading scheme for a hierarchical edge cloud architecture with multiple tiers. A non-linear integer programming problem is formulated to select a tier for task offloading and determine the amount of computing capacity needed to process the tasks. In Samanta and Chang (2019) and Samanta and Tang (2020), the authors formulate the computation offloading as non-convex optimization problem with the objective of service utilization maximization and use gradient descent method to solve its Lagrangian optimization problem.

(Non-)convex optimization is a classic method for optimization problem and it can obtain a (near) global optimal solution. However, since it is a centralized method, a powerful entity is required to support intensive calculations. Moreover, due to its static calculation, (non-) convex optimization modeling is not well-suited for the dynamic of the environment.

6.2. Markov decision process

MDP (Puterman, 2005) is suitable for offloading optimization where dynamic decision making is a requirement in light of changing environmental parameters, such as wireless channel conditions and computation load. A MDP system normally consists of five-tuple $(\mathcal{K}, \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$, where $\mathcal{K}$ represents decision epochs; $\mathcal{S}$ denotes a set of states; $\mathcal{A}$ is the set of actions; $\mathcal{P}$ represents the state transition probabilities; and $\mathcal{R}$ is rewards. In this model, it is critical to define the five-tuple according to the offloading procedure. Afterward, we can formulate the following Bellman's equation:

$$V^*(s) = \max_{a \in A} \left\{ R(s,a) + \lambda \sum_{s' \in S} p(s'|s,a) V^*(s') \right\} \tag{17}$$

where R is the immediate reward, p is state transition probability, λ is a discount factor, and V is the expected total rewards. Then, using a classical solution such as value iteration algorithm or policy iteration algorithm, we can obtain an optimal offloading decision, such as the decision on local execution or offloading and the decision on EC server should execute the offloaded tasks. Table 4 represents the offloading schemes which we discuss in this subsection. They are compared based on the following features: decision maker, system state, action, reward, goal, and algorithm.

Most of MDP models define the fixed decision epochs, i.e., equal time slot interval. From this assumption, it is assumed that either the computation tasks are requested at the beginning of each time slot (Mao et al., 2016a) (Fig. 7a) or the incoming requests are accumulated (in a queue) during a slot and the decision is made at the beginning of the next slot (Xu et al., 2017) (Fig. 7b). MDP with fixed time slot simplifies the modeling, but often suffers from additional latency due to accumulated waiting time (Lin et al., 2019a). If the decision is made immediately upon receiving a task (Fig. 7c), this modeling removes the waiting time, but the uncertain time slot interval requires predicting the length of the next epoch, which makes the MDP model less accurate (Lin et al., 2019a). It is worth noting that most of the works on MDP assume that the decision epoch is long enough to finish executing a task (or a component) (Zhang et al., 2015), otherwise the modeling becomes extremely complicated. Next, we conduct an analysis of computation offloading based on MDP.

ST-CODA (Ko et al., 2018) focuses on hierarchical offloading, where an ED could make a decision to: 1) offload to the central cloud; 2) offload to an EC server; 3) process the task by itself; and 4) delay the task processing. The last decision is because the ED anticipates a movement to a better wireless coverage area, and as a result delays the offloading process. The objective of the established MDP model is to maximize the utility which combines the task completion reward, transmission cost, and energy consumption cost. Then, the classic MDP algorithm namely, the value iteration algorithm (Puterman, 2005), is used to solve the formulated model.

The authors of Kamoun et al. (2015) and Labidi et al. (2015) focus on the one-to-one offloading scenario. The ED collects the system state information $s(t) = (b(t), x(t))$, where $b(t)$ is its buffer status and $x(t)$ is the channel quality information. Based on this state information, the EC server makes a decision for ED to offload or not the tasks, in order to minimize the ED's energy consumption while satisfying the delay requirement. This process is then modeled as a constrained MDP (Altman, 1999). To solve the model, the authors propose two algorithms, an online algorithm which is based on the Post-Decision State (PDS)

Table 4
Computation offloading schemes based on Markov Decision Process.

Scheme	Decision maker	System state	Action	Reward(s)	Goal	Algorithm
Scheme in Ko et al. (2018)	ED	s=(t, l, c, r) t: task phase l: location information c: available network information r: task remaining time	Offloading decision • offloading to CC • offloading to EC • local processing • delay the processing	Utility = Task completio - Transmission cost - Energy consumption	Reward maximization	Value iteration algorithm
Scheme in Kamoun et al. (2015)	EC server	s=(b, x) b: buffer status of terminal x: channel quality information	Offloading decision • local processing • offloading • idle	Cost = Energy consumption	Cost minimization	Online solution with PDS Offline solution based on occupation measure
Scheme in Wei et al. (2019)	ED	s=(O, h, z_1, z_c, b) O: generated data h: harvested energy z_1: queue state for end device z_c: queue state for EC server b: residual energy level	Offloading decision • dropping • local processing • offloading	Rewards = Computation profits + Negative of latency cost	Reward maximization	Online solution combining Q-function reconstruction with post decision state
Scheme in Xu et al. (2017)	EC server	s=(λ, e, h, b) λ: computation task arrival rate e: observable environment state h: backbone network state b: battery state	$a = (a_1, a_2)$ a_1: offloading decision a_2: autoscaling	Cost = Delay cost + Battery depreciation cost + Backup power cost	Cost minimization	Online solution with PDS
Scheme in Zheng et al. (2019b)	ED	s=(m, g, q) m: base station state g: channel quality information q: queue state	$a = (a_1, a_2)$ a_1: offloading decision a_2: energy allocation	Utility function of task satisfaction regarding delay, payment, and dropping	Utility maximization	Online solution combining Q-function reconstruction with post decision state
Scheme in Wang et al. (2019)	ED	s=(u, v, H) u: ED state (location, battery and computing capability) v: task information H: other EDs' states	Offloading decision • idle • local processing • offloading to EC • offloading to another ED	Cost = Energy cost + Latency cost + Dropping cost	Cost minimization	Dynamic RL scheduling and deep dynamic scheduling
Scheme in Zhang et al. (2018d)	EC server	s=(η, b, T) η: channel condition b: battery state T: task queue length	$a = (c, p, f)$ c: offloading decision p: transmission power f: computation resource allocation	Utility = Executed task - Energy cost - Delay cost	Utility maximization	Actor-critic based RL
Scheme in Le and Tham (2017)	ED	s=(q, c, d) q: queue state c: channel state d: distance state	Offloading decision • local processing • offloading to EC server$_1$ … • offloading to EC server$_N$	Rewards = Completion function - Cost function	Reward maximization	Q-learning
Scheme in Meng et al. (2018)	ED	s=(c, q_l, q_r) c: channel state q_l: local queue state q_r: remote queue state	$a = (c, p, f)$ c: offloading decision p: transmission power f: computation resource allocation	Cost = Delay cost + Energy cost	Reward maximization	Multi-level water-filling solution

method (Salodkar et al., 2008) and an offline algorithm which is based on occupation measure representing the probability to visit each (state, action) pair (Altman, 1999).

Wei et al. (2019) also use one-to-one offloading, but they consider data priority as an important factor. They define different data priority levels and the system handles the data according to its priority level, which is achieved by letting higher priority data contribute more on rewards. Meanwhile, the authors assume all related probability distributions (e.g. energy harvesting probability) are unknown, so classic algorithms for MDP, such as the value iteration algorithm or the policy iteration algorithm, cannot be used. Instead, they use the Q-learning algorithm (Sutton et al., 1998), where the Q-value $Q(s, a)$ can be obtained by:

$$Q(s,a) = Q(s,a) + \kappa(R(s,a) + \rho \max_{a' \in A} Q(s', a') - Q(s,a)) \quad (18)$$

where κ is the learning rate. This iterative equation eliminates the state transition probability. The optimal decision (action) is then obtained from the optimal Q-value $Q^*(s, a)$. Moreover, to address the issue 'curse of dimensionality' (Sutton et al., 1998), the authors also use the PDS

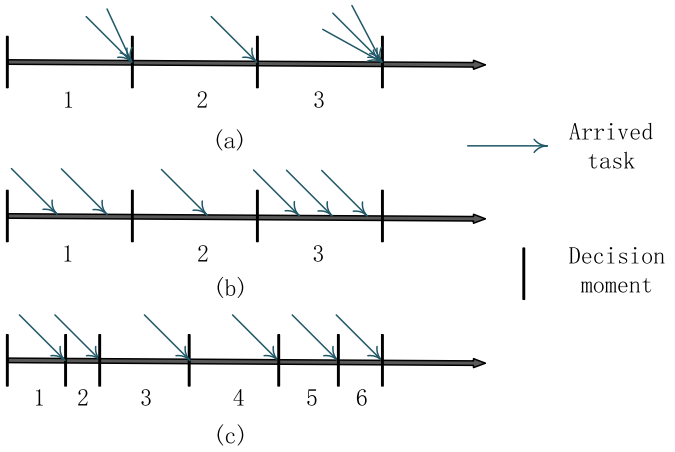

Fig. 7. Task arriving model: (a) fixed decision epochs with the computation tasks being requested at the beginning of each epoch; (b) fixed decision epochs with the incoming requests being accumulated during a slot; (c) varied decision epochs.

method (called *after state* in the paper). The dimension of the PDS defined is significantly reduced compared to the normal state, so the complexity is reduced. Finally, the value function approximation, which uses a classic function or a neural network is adopted to approximate the PDS value.

Xu et al. (2017) consider an energy harvesting scenario. In their work, the EC system consists of a number of EC servers which receive computation tasks from EDs. The system aims to minimize the cost of delay and energy consumption by making decisions on the number of tasks to executed locally (EC) and remotely (cloud computing) as well as the number of servers to be activated if the tasks are executed locally (two types of actions should be taken). To do this, the system state is modeled as $s(t) = (\lambda(t), e(t), h(t), b(t))$, where $\lambda(t)$ is computation task arrival rate, $e(t)$ is observable environment state which is used to calculate the harvested energy, $h(t)$ is the backbone network state which is used to estimate the latency between the edge system and the cloud system, and $b(t)$ is the battery state. Since the defined 'action' and 'state' have large spaces, the model also suffers from the curse of dimensionality issue. Likewise, an online Reinforcement Learning (RL) algorithm with PDS is used to address the issue. Similar work can be found in Zheng et al. (2019b), where an ED can offload its tasks to an EC server via different BSs, so it needs to decide whether to execute a task locally or remotely on an EC server. In the case of offloading, it should further decide via which BS the task should be transmitted. Moreover, the ED also decides how much energy should be allocated for local computing or offloading transmission. All these decisions are based on its state including its allocated BS, channel quality information, and its queue state. The authors use an online solution combining Q-function with PDS to maximize the utility function related to delay, payment, and task dropping.

In contrast, Le and Tham (2017) focus on the computation offloading at ad hoc cloudlet (Chen et al., 2015), where an ED can decide to process the tasks locally or offload it to one of N special EDs which act as EC servers. Since both ED and EC servers are mobile, the distance between the ED and EC servers determines the communication quality between them. Thus, the distance is defined as an important MDP state parameter. To approximate the optimal solution without prior knowledge of transition probability, Q-learning is used to solve the model.

The work in Wang et al. (2019) can be considered as a combination of two offloading schemes discussed above, i.e., an ED can offload its tasks to an EC server (as scheme of Zheng et al. (2019b)) or another ED acting as an EC server (as scheme of Le and Tham (2017)). To solve the formulated MDP problem, two RL based algorithms are proposed. One is called the Dynamic RL Scheduling (DRLS) algorithm and it is similar to Q-learning, and the other is called Deep Dynamic Scheduling (DDS) algorithm which is based on Deep Q-Network (DQN) (Suh and Hougen, 2017). The simulation results demonstrate that DDS has a slight advantage over DRLS.

Continuous MDP is used in Zhang et al. (2018d), where the authors aim to improve mobile operators' revenues by maximizing the number of the offloaded tasks while decreasing the energy expenditure and time-delays. Since the states and actions defined in the formulated MDP are continuous (e.g. channel state condition and action of transmission power setting are continuous), the traditional value-based methods, such as Q-learning, State Action Reward State Action (SARSA) and so on, are ineffective. Therefore, the authors propose an online actor-critic RL, which includes two different learning processes namely, policy improvement and policy evaluation (Grondman et al., 2012).

The research efforts discussed above use RL to deal with the curse of dimensionality issue that arises in MDP modeling. Alternatively, Meng et al. (2018) reformulate the MDP model as a Virtual Continuous Time System (VCTS) with reflections. Then, an approximate priority functions is derived with dynamic rate estimation. Based on the approximate priority function, a multi-level water filling solution is proposed to address the dependence between ED's queue state and that of the EC server which are considered together in their work.

We summarize other MDP based offloading works as follows. In Li and Huang (2017), hierarchical offloading is modeled as MDP and Ordinal Optimization (OO) (Lau and Ho, 1997) based algorithm is applied to solve the model. Liu et al. (2016a) describe the offloading problem as MDP and model it as a non-convex optimization problem. To solve it, the non-convex optimization problem is simplified as a LP problem. Hong and Kim (2016) propose a semi-MDP framework to jointly control the local CPU frequency, modulation scheme as well as data rates.

MDP is suitable for modeling computation offloading in EC. However, with the curse of dimensionality issue, classic MDP algorithms, such as value iteration policy and policy iteration policy, are unsuitable to handle the MDP problem. Therefore, previous works adopt other tools, such as RL, VCTS, and OO based algorithm to solve the model.

6.3. Game theory

Game theory is a powerful tool for designing distributed mechanisms. It can be used in multi-users offloading scenario, where each ED locally selects an appropriate strategy to achieve a mutually satisfactory offloading solution. In this model, each ED makes offloading decision, receives rewards (utility), and updates the decision. This procedure is repeated until the rewards cannot be improved further, i.e., Nash equilibrium. Therefore, an iterative algorithm is usually used to find the Nash equilibrium. For each ED, this is essentially an optimization problem, i.e., maximization of its own utility. Hence, the classical optimization solution, such as KKT conditions, can be used to obtain the Nash equilibrium. The popular game models used in previous computation offloading works include:

- In the Potential game (Monderer and Shapley, 1996), there exists a potential function $\Phi(\mathbf{x})$ ($\mathbf{x}$ is the strategy vector of $\mathcal{N}$ players) such that for every player i. If a player changes from the strategy x to x', its reward function (or cost function) $R(\mathbf{x})$ maps to the potential function. That is:

$$\begin{aligned} &R(x^{'}, x_{-i}) - R(x, x_{-i}) > 0 \\ &iff\ \Phi(x^{'}, x_{-i}) - \Phi(x, x_{-i}) > 0\ for\ \forall i \in \mathcal{N} \end{aligned} \tag{19}$$

  The attractive property of the potential game is that it always has a Nash equilibrium.
- Generalized Nash Equilibrium Problem (GNEP) (Facchinei and Kanzow, 2007) is a non-cooperative game in which each player's admissible strategy set depends on the other players' strategies.

Hence, its strategy vector is not the Cartesian product of all players' strategy, $\mathbf{x} \neq \mathbf{x}_1 \times \mathbf{x}_2 \times \cdots \times \mathbf{x}_N$, where $\mathbf{x}_i$ is the strategy set of player i.

- Stochastic game (Bowling and Veloso, 2000) is a dynamic game with probabilistic transitions played by one or more players. The multi-player stochastic game can be described by $(\mathcal{N}, \mathcal{K}, \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$, where $\mathcal{N}$ denotes the number of players and other parameters are identical to MDP. In this model, $\mathcal{N}$ players select actions and receive rewards. The state transition thus depends on the joint actions of all players.
- In the Stackelberg game (Fudenberg and Tirole, 1993), two roles for a player are defined: the leader who moves first and the other role is the follower who moves after. The leader maximizes its rewards by anticipating the optimal strategies of the follower(s). This game can be solved to find the subgame perfect Nash equilibrium.
- Coalitional game (Kahan and Rapoport, 2014) is a cooperative game. In this game, it is important to form a coalition to which all players join is always beneficial.

Table 5 presents the offloading schemes based on game theory. In this table, we give a comparison in terms of game model, player, player's strategy, utility/cost function, and the solution for each model.

Potential game is used to model the computation offloading in Chen et al. (2016) and Cao and Cai (2018). In Chen et al. (2016), the authors use on the many-to-many offloading. EDs (players) not only make an offloading decision, but they also decide which channel should be selected for offloading as well. The goal is to find the maximum number of beneficial EC users, i.e., letting the maximum number of EDs offload the computation tasks to EC servers. To do this, the authors develop a potential function for this multi-player computation offloading game, thus demonstrating that this is a potential game. Consequently, the game always has a Nash equilibrium under which the number of beneficial EDs is maximized. Afterward, a distributed computation offloading algorithm is developed to achieve the Nash equilibrium. The key idea of the algorithm is to utilize the finite improvement property of the multi-player computation offloading game and let one ED at a time improve its computation offloading decision.

In Cao and Cai (2018), the multi-user multi-channel computation offloading problem is formulated as a non-cooperative game. Then, the authors prove that this non-cooperative game is the exact potential game, by constructing an exact potential function to which the players' cost function can be mapped ($R(x', x_{-i}) - R(x, x_{-i}) = \Phi(x', x_{-i}) - \Phi(x, x_{-i})$). To achieve Nash equilibrium, they propose a machine learning based Fully Distributed Computation Offloading (FDCO) algorithm in which each player starts with a mixed strategy. This mixed strategy evolves to a pure strategy by user's response to a reaction from the environment. To implement the proposed scheme, two stages are used: a learning stage and an offloading stage. The learning stage is used to find an optimal strategy which is then applied in the offloading stage for task decision and execution.

Game theory is also used in Liu et al. (2018a), where EDs are supposed to have the social relationships in a hierarchical offloading scenario. The strategy of each ED is to partition received tasks during a time slot into four parts, local execution, offloading to EC, offloading to cloud, and rejection. The objective is defined as the task execution cost which is the weighted sum of computing cost, latency cost, and rejection cost. Each ED selects a strategy whose aim is to minimize its social group execution cost which is the total execution cost of all members in the group. Since the offloaded tasks to EC cannot exceed the EC's computation capability, the strategy selection of an ED depends on other EDs' strategies. Therefore, the model formed is a GNEP. To solve this problem, the authors first transform it into a classic Nash equilibrium problem, and then address it with semi-smooth Newton method with Armijo line search.

The potential game offloading model in Chen et al. (2016) is extended in Zheng et al. (2016, 2019a), where the players' activeness (e.g. offloading or not) and the channel condition are dynamic, and therefore unknown when making an offloading decision. That is, the players do not exactly know the system state and the state transmission probability. Therefore, the offloading problem in Zheng et al. (2016, 2019a) is modeled as a stochastic game. The authors prove that the formulated stochastic game is also a potential game by constructing a potential function which is similar to Chen et al. (2016), so a Nash equilibrium exists. To find the Nash equilibrium, the authors propose an iterative algorithm called Multi-Agent Stochastic Learning (MASL) algorithm, where each player independently updates its offloading strategy based on a probability vector over the strategy space, and receives an action-reward from the dynamic environment.

Liu et al. (2017) work on hierarchical offloading where EDs' computation tasks can be offloaded to EC or cloud. The offloading problem is modeled as a Stackelberg game, in which the cloud acts as a

Table 5
Computation offloading schemes based on game theory.

Offloading schemes	Game theory	Player	Player action (strategy)	Utility/cost	Solution/Algorithm
Scheme in Chen et al. (2016)	Potential game	EDs	Offloading decision and channel selection	Offloading maximization	Distributed algorithm
Scheme in Cao and Cai (2018)	Exact potential game	EDs	Offloading decision and channel selection	Execution overhead minimization	FDCO algorithm
Scheme in Liu et al. (2018a)	GNEP	EDs	Task partition	Cost minimization	Semi-smooth Newton method with Armijo line search
Scheme in Zheng et al. (2016, 2019a)	Stochastic game\ potential game	EDs	Offloading decision and channel selection	Cost (processing time and energy consumption) minimization	MASL-algorithm
Scheme in Liu et al. (2017)	Stackelberg game	Cloud server: EC servers:	Computation request payment Offloading decision	Utility (cost reduction-payment) maximization Utility (revenue-cost) maximization	One\Multi round algorithm
Scheme in Aliyu et al. (2017)	Coalition game	EC servers	Coalition formation	Capacity sufficiency and delay minimization	Integer partition based minimum-cost adaptive policy
Scheme in Tanzil et al. (2015)	Coalition game	EC servers	Coalition formation	Gain (revenue-cost) maximization	Dynamic coalition formation algorithm
Scheme in Jošilo and Dán (2017)	Nash equilibrium game	EDs	Offloading decision and AP selection	Cost (processing time and energy consumption) minimization	A distributed algorithm called JPBR
Scheme in Deng et al. (2016a)	Sequential game	EDs	Offloading decision and channel selection	Cost (processing time and energy consumption) minimization	Sequential game algorithm
Scheme in Zhang et al. (2017)	Stackelberg game	EC servers: EDs:	Capacity offering and price Offloading decision	Revenue maximization Utility (latency reduction-payment) maximization	Iterative distributed algorithm
Scheme in Ranadheera et al. (2017)	Minority game	EC servers	Active\Inactive	Energy efficiency	Reinforcement learning based algorithm

leader which specifies the payments to EC for offloaded computations. Each EC server acts as a follower and determines the amount of computation it is willing to accept. Through theoretical analysis, the authors demonstrate that the game is guaranteed to reach a unique Nash equilibrium. Under this Nash equilibrium, the cloud maximizes its utility which is defined as the gain by offloading computation to the EC servers (i.e., cost reduction minus the payment to EC), while each EC server is able to maximize its utility which is defined as the gain by accepting computation from the cloud.

In Aliyu et al. (2017), the authors study resource allocation for offloaded traffic in an EC-enabled IoT network. In this scenario, several Micro Data Centers (MDCs) located in the vicinity of EDs serve as EC servers. The major work is to generate coalition structure for MDCs to optimize resource utilization, which is done by employing a policy-based QoS control framework using the principles of a coalition game. Similarly, Tanzil et al. (2015) study the formation of EC platforms which consist of Femtocell Access Points (FAPs) acting as EC server via the coalitional game theoretic. The core of the formulated game represents an optimal structure for EC platforms such that the maximum computational resources of FAPs are maximally exploited for offloaded tasks.

We summarize other game-based offloading works as follows. Chen (2015) models the offloading problem as a potential game. In fact, the research works discussed above with potential game (Chen et al., 2016; Zheng et al., 2019a) can be considered as an extension of Chen (2015). Jošilo and Dán (2017) apply game theory to coordinate the offloading decisions of mobile devices to ensure EC resource efficiency, and prove that the pure strategy's Nash equilibrium exists for this problem. Deng et al. (2016a) consider a multi-cell, quasi-static environment, and treat the computation offloading problem as a dynamic sequential game. The authors in Zhang et al. (2017) study offloading in a vehicular network with EC. The problem is formulated as a Stackelberg game, where the servers and the offloading vehicles are modeled as the leaders and the followers respectively. In Ranadheera et al. (2017), the authors investigate the EC server activation problem in the offloading system. It is formulated as minority games in which the number of desired active EC servers can be obtained for each offloading round.

The main advantage of game theory is its distributed nature, which matches well the multi-user offloading scenario. However, game theory can only guarantee to achieve Nash equilibrium which might not be the global optimal solution.

6.4. Lyapunov optimization

Lyapunov optimization refers to the use of a Lyapunov function to optimally control a dynamical system (Meyn and Tweedie, 2012). It is widely used in networks to control the queuing system. Lyapunov optimization defines a *drift-plus-penalty expression* as:

$$\triangle L(t) + Vp(t) \tag{20}$$

where $\triangle L(t)$ is the Lyapunov drift, $p(t)$ is the penalty function and V is a non-negative weight that is chosen as desired to achieve a performance tradeoff. The goal is to minimize the drift and penalty simultaneously.

When using Lyapunov optimization in offloading modeling, the key is to define the drift and the penalty. In an offloading scenario, the drift could be the energy queue drift or the task queue drift, whereas the penalty is usually the offloading objective, e.g. the minimization of task dropping or execution latency. Then at each time slot, by minimizing the drift-plus-penalty expression, the optimal offloading decision and other parameters could be determined. It is worth noting that solving the Lyapunov optimization has much lower computation complexity than (non-)convex optimization because the drift-plus-penalty expression is usually a deterministic problem. Moreover, in contrast to MDP, it does not require knowledge of the probability distribution of the random event process.

Even though Lyapunov optimization is a per-time optimal solution, many previous works illustrate that it can behave arbitrarily close to the global optimal performance (Mao et al., 2016a). Table 6 lists the offloading schemes based on the Lyapunov optimization model which we discuss next. They are compared based on Lyapunov drift, penalty function, main constraints, and related parameters.

The authors of Mao et al. (2016a) study a system consisting of an energy harvesting ED with an EC server. They formulate this one-to-one offloading as a Lyapunov optimization problem, in which $\triangle L(t)$ is defined as the battery energy drift which varies by harvested energy and consumed energy, and $p(t)$ is defined as the cost including both delay cost and dropping cost. By minimizing the drift-plus-penalty expression (20), the ED can determine CPU frequency, harvested energy, transmission power, and decide to offload or not. It is worth noting that, since the battery energy level is temporally correlated, which makes the conventional Lyapunov optimization technique inapplicable, the weighted perturbation method is applied to address this issue (Neely and Huang, 2010). The authors of Zhang et al. (2018b) extend the above work, where the EC system also consists of one EC server and one ED, but the ED runs N applications and each application has two corresponding buffer queues, an offloading queue for storing the incoming tasks and a downloading queue for storing the computing results from the EC server. Together with the energy queue, the Lyapunov drift includes three types of queues and each type has N queues. Similar to Mao et al. (2016a), the weighted perturbation method is used to address the Lyapunov optimization problem. The authors decompose the original problem into an energy harvesting sub-problem and a task assignment sub-problem in order to obtain an optimal solution. This work is further extended in Zhang et al. (2018a), where the system consists of one EC server and multiple EDs with each ED running multiple applications. Therefore, the drift and penalty are defined for all EDs and their applications.

Lyu et al. (2017) study many-to-one offloading scenario (a single EC server with multiple EDs). They try to stabilize not only ED's queues but also the queue of EC server, which makes the formulated Lyapunov optimization much more complex. To achieve the queue stability and maximize system utility which is defined as the amount of offloaded data to EC servers, the optimal data admission, the optimal offloading schedule, and the optimal energy harvesting budget should be determined. This is done by decoupling them in both the objective function and its constraints.

Chen et al. (2018) work on multi-user (multi-ED) multi-task computation offloading scenario and assume that EDs should pay for the offloaded tasks. The formulated Lyapunov optimization aims to minimize this payment while keeping the energy queue in each EC server stable. Even though the original time-dependent optimization problem is converted to the Lyapunov optimization which is a per-time slot deterministic problem with reduced complexity, it is still NP hard. Therefore, the authors introduce two greedy algorithms, centralized and distributed Greedy Maximal Scheduling Algorithms (GMSA), to address the problem. A multi-user multi-server scenario is also studied in Zhao et al. (2018a). The authors aim to minimize the execution delay and task dropping cost while keeping EDs' energy queues stable. Since there exists a correlation between any two EDs when choosing to offload computation tasks, the solution to the Lyapunov optimization is not trivial. The authors propose a genetic algorithm with greedy policy to address this issue.

Zhang et al. (2019) address the issues of computation offloading in mobile cloudlet, where mobile EDs act as EC servers and provide computing service to IoT EDs. The offloading decision problem is formulated as an auction problem. The mobile EDs estimate the value to process a task and submit bids to compete for the task. An entity called the dispatcher determines the winners and pays for them. A key issue in this auction is to motivate the EDs to minimize the defined objective function which is described by the Lyapunov function consisting of energy spare drift and the negative of overall rewards. To address this issue, a value function which determines the user utility is defined based

Table 6
Computation offloading schemes based on Lyapunov optimization.

Offloading schemes	Lyapunov drift	Penalty function	Main constraints	Parameters to be determined
Scheme in Mao et al. (2016a)	Energy queue drift (ED)	Delay cost + Dropping cost	Energy harvesting constraint Energy consumption constraint Transmit power constraint CPU frequency constraint	Offloading decision Transmit power allocation CPU frequency allocation Energy harvesting budget
Scheme in Zhang et al. (2018b)	Offloading queue drift (ED) Downloading queue drift (ED) Energy queue drift (ED)	Energy consumption + Delay cost	Energy harvesting constraint Energy consumption constraint Transmit power constraint CPU frequency constraint	Offloading decision Transmit power allocation CPU frequency allocation Energy harvesting budget
Scheme in Zhang et al. (2018a)	Task queue drift (ED) Virtual energy queue drift (ED)	Energy consumption	Energy harvesting constraint Energy consumption constraint Transmit power constraint CPU frequency constraint	Offloading decision Transmit power allocation CPU frequency allocation Energy harvesting budget bandwidth allocation
Scheme in Lyu et al. (2017)	Energy queue drift (ED) Data queue drift (ED) Virtual queue drift (ED) Task queue drift (EC server)	Negative of system utility	Transmission time constraint Energy harvesting constraint Energy consumption constraint	Offloading schedule Data admission control Energy harvesting budget
Scheme in Chen et al. (2018)	Energy queue drift (EC server)	User payment (Operator revenue)	Offloading efficiency constraint Computing latency constraint Energy consumption constraint	Energy harvesting strategy Offloading strategy
Scheme in Zhao et al. (2018a)	Energy queue drift (ED)	Delay cost + Dropping cost	Energy harvesting constraint Energy consumption constraint Transmit power constraint CPU frequency constraint	Offloading decision Transmit power allocation CPU frequency allocation Harvested energy budget
Scheme in Zhang et al. (2019)	Energy spare drift (EC server)	Negative of overall rewards	Task assignment constraint Energy consumption constraint	Task assignment CPU frequency allocation
Scheme in Wu et al. (2018)	Energy queue drift (ED)	Dropping cost + Computation cost	Task latency constraint CPU frequency constraint Delay constraint Energy harvesting constraint Energy consumption constraint	Offloading decision Traffic admission control Energy harvesting budget
Scheme in Pu et al. (2016)	Energy queue drift Computation resource queue drift D2D bandwidth resource queue drift Cellular bandwidth resource queue drift	Energy consumption	Incentive constraint Energy budget constraint Scheduling constraint	Offloading decision (task scheduling)

on computation reward. To guarantee the truthfulness of the auction, a Vickrey-Clarke-Groves (VCG) (Fudenberg and Tirole, 1993) auction-based scheme is proposed, so that bidding with the true value becomes a dominant strategy.

We summarize other Lyapunov-based offloading works as follows. In Wu et al. (2018), the formulated Lyapunov optimization aims to minimize the system cost which consists of both dropping cost and computation cost by making both offloading decision and traffic access control decision. Power-delay tradeoff in multi-user scenario is analyzed by Lyapunov optimization in Mao et al. (2016b). D2D offloading is formulated as Lyapunov optimization in Pu et al. (2016) to minimize the energy consumption of all EDs. FEMOS (Zhao et al., 2018b) uses Lyapunov optimization to jointly optimize node assignments at control tier and resource allocation at access tier. DCOA (Chen et al., 2019) uses Lyapunov optimization techniques to simplify the stochastic offloading problem into a deterministic optimization problem. The deterministic problem is decomposed into a series of subproblems which are solved concurrently in an online and distributed way.

Lyapunov optimization has been wildly used for offloading modeling. Computation offloading problem is usually a complex problem. With Lyapunov optimization, the complexity can be greatly reduced. Therefore, Lyapunov optimization is a powerful tool for offloading optimization even though it cannot grantees a global optimization. In Lyapunov optimization works, the energy queue is usually chosen as Lyapunov drift. However, the energy queue is normally time-dependent, which makes the allowable action sets not i.i.d., so the perturbation parameter is introduced to define a virtual energy queue. Previous works also illustrate that there often exists a tradeoff between the drift and the penalty function. It obeys a $[O(1/V), O(V)]$ tradeoff with V as a control parameter.

6.5. Machine learning

Machine learning (reinforcement learning) is a promising approach to deal with the high complexity in practice (Mnih et al., 2015). Reinforcement learning has the same methodology as MDP, i.e., an optimal offloading decision is determined according to the system's state. However, it obtains this optimal decision using learning techniques, i.e.,

by training a neural network or Q-table.

In this subsection, we discuss computation offloading in EC from the point of view of machine learning, especially RL. Table 7 presents the offloading schemes using machine learning methods which we discuss next. In this table, 'RL method' indicates the RL method used for offloading modeling. RL usually describes a problem as a MDP problem, so 'system description' describes the system in terms of state, action, and reward. In RL method, there exists a 'core model'. The input data and output results are important to machine learning, which are described by 'input' and 'output'.

An important RL method is Q-learning. In Sen and Shen (2019), the authors study three-tier hierarchical offloading, where the key problem is to make an offloading decision. To do this, the Q-learning based method is used to obtain a decision table, where the best action for each state is stored. Then, this table is distributed to all EDs which use it for making decision about offloading. Q-learning is also used in Dab et al. (2019) to find the task assignment strategy that minimizes the total energy consumption of EDs, while considering the latency constraint. We have discussed, in section 6.2, that with Q-learning, we do not need knowledge about the transition probability, so the issue of 'curse of dimensionality' can be addressed to some degree. However, Q-learning is still not suitable for a large-scale offloading scenario where the Q-table (store the Q-value for each state) size can be immense and it is impossible to visit all of them multiple times to learn a good policy (Li et al., 2018b; Vita et al., 2018). To address this issue, researchers have consider the use of deep RL, where, instead of training the Q-table, a neural network is trained to obtain the Q-value.

Tan and Hu (2018) work on both computation offloading and content caching for vehicular networks, where a vehicle can offload its tasks or require contents from Road Side Units (RSU) or other vehicles. To jointly optimize offloading and caching, the authors propose a deep Q-learning with multi-timescale framework (ChangFard et al., 2003) in which the large timescale model aims to select a set of best RSUs and/or vehicles to serve as EC servers, while the goal of the small timescale model is to maximize the immediate reward by executing caching and offloading actions. Both models are solved by deep Q-learning, in which two deep Q-networks, main and target Q-networks, are trained. Finally, a mobility-aware reward estimation is proposed to address the complexity issue caused by the large action space.

Munir et al. (2019) work on EC with micro-grid, where an EC platform with multiple BSs and multiple EC servers are defined. This platform is powered by three types of energy: renewable energy (e.g. solar), non-renewable energy (e.g. diesel generator), and grid-powered energy. The goal is to minimize the total energy consumption by optimizing task assignment and energy supply plan. The formulated model is a mixed integer nonlinear optimization coupled with the dependencies of uncertainty for both energy consumption and generation. The problem is decomposed into two sub-problems. The sub-problem of task assignment is formulated as a community discovery problem and it is solved distributively for each BS. To solve the sub-problem of energy supply plan, a supervised learning technique is used, where back-propagation method updates the model of DQN (Suh and Hougen, 2017).

In the scenario of UAV based multiple-user EC in Cheng et al. (2019), each ED could execute its tasks locally, or offload them to the UAV (EC server) or the cloud which is accessed by satellite communication. Due to the system' dynamic, continuous, and large-scale characteristics, Q-learning, which is based on value function iteration, becomes inefficient. Therefore, the authors relay on actor-critic RL (Mnih et al., 2016), where 'actor' is a reference to the learned policy, and 'critic' refers to the learned value function which guides the update of the policy. The actor-critic RL updates the policy per time slot instead of per episode (a number of time slots), so that the offloading algorithm converges fast and achieves a lower total cost.

Min et al. (2019) work on one-to-many offloading scenario. An ED needs to decide to which EC server and how much data should be offloaded. The system is described as state-action pair and the goal is to determine the best action for each state. The authors study two cases for computation offloading. First, the system is viewed as MDP, i.e. the next system state depends only on the current state and action, not on the past history of the process. An offloading scheme called RL-based Offloading (RLO) (which is based on Q-learning) is proposed to determine the best action for each state. Second, history dependence is taken into account and the authors propose Deep RL-based Offloading (DRLO) for it. DRLO utilizes a deep Convolutional Neural Network (CNN) (Mnih et al., 2015) to determine an action for a sequence of state-action pairs. The input is a sequence of previous system state-action pairs and the output is Q value of each action for this sequence. With these Q values, a best action can be determined. The simulation results show that the DRLO scheme generates the best policy and outperforms the DRL scheme.

We summarize other computation offloading works using machine learning as follows. Cao et al. (2019) study machine learning based

Table 7
Computation offloading with machine learning.

Scheme	RL method	System Description	Core model	Input	Output
Scheme in Sen and Shen (2019)	Q-learning	*State*: all node resource characteristics, task characteristics *Action*: offloading decision *Reward*: latency and energy consumption	Q-table	s_t(system state)	$Q^*(s_t, a_t)$(Q value)
Scheme in Dab et al. (2019)	Q-learning	*State*: task description, channel information, computing resources *Action*: task placement *Reward*: energy consumption	Q-table	s_t(system state)	$Q^*(s_t, a_t)$(Q value)
Scheme in Tan and Hu (2018)	Deep Q-learning	*State*: vehicle's mobility, channel information, caching contents, computing resources *Action*: server selection, caching/computing resource allocation *Reward*: communication, storage, computing	DNN	s_t(system state)	$Q^*(s_t, a_t)$(Q value)
Scheme in Munir et al. (2019)	Deep Q-learning	*State*: energy demand, energy generation, energy store *Action*: energy store decision, buying decision *Reward*: energy consumption	DQN	s_t(system state)	(s_t, a_t, s_{t+1}) (state-action sequence)
Scheme in Cheng et al. (2019)	Actor-critic RL	*State*: task information, ED information, pathloss information *Action*: offloading decision *Reward*: latency, energy, server usage cost	Critic network Actor network	ED information EC server information Cloud information Task information	Optimal decision
Scheme in Min et al. (2019)	Q-learning/Deep Q-learning	*State*: link transmission rate, harvested energy, battery level *Action*: EC server selection, offloading rate *Reward*: data sharing gains, task drop, energy consumption, and computation latency	Q-table/Deep CNN	s_t(system state)	$Q^*(s_t, a_t)$(Q value)

approaches for offloading. By comparing with convex optimization approach, the authors conclude that machine learning based approaches outperform the convex optimization approach in terms of payoff and running time. Li et al. (2018b) and Vita et al. (2018) work on multi-user EC system by resorting to Deep Neural Network (DNN) or DQN to address the scalability issue. Xu et al. (2019) work on machine learning application offloading, such as virtual speech recognition and intelligent cognitive assistants. In Ranadheera et al. (2017), different learning methods (e.g. exponential learning, adaptive strategy, win-stay lose--shift strategy and Q-learning) are analyzed using simulations.

The offloading decision problem is usually described using a MDP-like model. However, due to its scalability issue (curse of dimensionality), traditional MDP algorithms fail to solve the model. RL based methods are thus used to address large-scale computation offloading. In addition to scalability issue, RL method can be used to address other aspects such as dynamic and continuous time issues as well.

6.6. Other modeling methods

MAGA (Shi et al., 2018) uses genetic algorithm for offloading modeling, in which the chromosome is encoded in a tuple of n (number of tasks) integers and each integer denotes the offloading strategy of a corresponding task. The goal is to improve the offloading success rate and decrease energy consumption of EDs while satisfying the job completion time requirements.

In Deng et al. (2016c), the offloading problem is formulated as a 0–1 integer programming model with delay constraint. To solve this problem, the authors proposed an algorithm called Binary Particle Swarm Optimizers (BPSO) in which each individual called as a 'particle' represents a potential solution. Each 'particle' follows two solutions. One is its own best solution and the other is the best solution of the group. With evolution, an optimal offloading solution can be determined.

Minimizing latency is critical in computation offloading. To accurately analyze the offloading latency, Li (2019) uses queueing models for an ED and multiple heterogenous EC servers (one-to-many offloading). Both ED and EC servers are modeled as M/G/1 queueing systems, so that the average task response time of the ED and each EC server can be obtained accurately and analytically and the average response time of all offloadable and non-offloadable tasks generated on the UE can be optimized.

6.7. Comparison of offloading modeling methods

In this subsection, we analyze the benefits and limitations of each modeling method and we discuss the scenario for which these models are well suited.

We compare the modeling approaches shown in Table 8. In this table, 'centralized/distributed' indicates whether the modeling method is preferred for centralized mode or distributed mode. 'Scalability' describes whether a method is scalable or not. 'Static/dynamic' indicates if a method is suitable for static or dynamic scenario. The achieved objectives of each method are described by 'Achieved objectives'.

Table 8
Computation offloading modeling method analysis.

Modeling method	Centralized/ distributed	Scalability	Static/ dynamic	Achieved objectives
(Non)convex optimization	Centralized	Not scalable	Static	Global offloading optimization
MDP	Centralized/ distributed	Not scalable	Dynamic	Local offloading optimization
Game theory	Distributed	Scalable	Dynamic	Nash equilibrium
Lyapunov optimization	Centralized	Relatively scalable	Static	Offloading optimization Lyapunov drift
Machine learning	Centralized	Scalable	Dynamic	Intelligent decision

(Non-)convex optimization has the advantage of achieving global or near global optimization. However, it requires a centralized entity to be introduced to collect all the necessary information and make the decision for all users, which makes this modeling not scalable. As a result, this method is not suitable for the large-scale offloading scenario. However, it works well for a small-scale scenario, especially for the case where the offloading problem can be modeled as convex optimization problem (global optimization can be achieved). (Non-)convex optimization is kind of static modeling, so it is not suitable for dynamic offloading scenario where the environment such as channel condition and network topology changes frequently.

The benefit of MDP is that it can applied in both centralized and distributed scenarios. In the centralized scenario, a centralized entity makes the decision for all users whereas in the distributed scenario each user makes the offloading decision individually. To achieve the optimized offloading, the decision maker (the centralized entity or each user) needs to know the system state when making the decision, so information collection is still a problem. Together with the curse of dimensionality issue, MDP is not scalable neither. Nonetheless, the dynamic decision making of MDP makes well suited for the dynamic offloading scenario where a user can improve the decision when the environment changes.

The advantage of game theory modeling is its inherent distributed characteristic. It is often used in multi-user offloading scenario where each user competes for computing or wireless resources. Similar to MDP, it can be applied in dynamic scenario. With game theory, each user only requires to maximize his/her own utility, which also makes it work well in the large-scale offloading scenario. However, game theory modeling can only guarantee to achieve the Nash equilibrium which might not be the global optimal solution.

In contrast to the above modeling methods which only focus on offloading optimization, Lyapunov optimization can achieve offloading optimization and Lyapunov drift optimization simultaneously. Therefore, it can be applied in the offloading scenario where, for example, load balancing and energy harvesting balancing should be addressed. Even though Lyapunov optimization is also a centralized modeling method, it could be applied to relatively large-scale offloading scenario because of its relatively low computing complexity. However, Lyapunov optimization is one-step optimization, it could only achieve local optimization.

Reinforcement learning can address the curse of dimensionality issue raised in MDP, so it is naturally applied to a high complexity offloading scenario, such as the large-scale offloading scenario. Moreover, reinforcement learning is an online method, which makes it suitable for a dynamic scenario. However, supporting reinforcement learning is usually beyond the capacity of EDs, so a powerful entity should be introduced to implement the reinforcement learning for all EDs.

7. Research challenges

7.1. Offloading modeling in edge intelligence

With the proliferation of EC and IoT, a recent trend is to integrate artificial intelligence into EC, which gives rise to a new edge paradigm namely, the emergence of Edge Intelligence (EI) (Zhou et al., 2019; Plastiras et al., 2018). In a three-tier EI architecture, a DNN which has multiple layers is deployed at both the EI (EC) server and the cloud server, i.e., the front layers of the DNN are deployed at the EI server; the remaining layers are deployed at the cloud server. The output of EI is offloaded to the cloud, but the challenge here is deciding on the output of which layer should be offloaded to the cloud, i.e., how to deploy the DNN. In a more typical scenario where multiple EI servers works together to provide Artificial Intelligence (AI) service for EDs, the offloading modeling for AI tasks should not only take into account of the

criteria discussed in this paper, but also consider the AI performance, i.e., offloading to an EI server which has already been trained with similar tasks. To make multiple EI servers cooperate to provide AI service is another challenge. This involves how to deploy DNN and how to train the network. Even though some works (for instance, the authors of Li et al. (2018a) discuss the issue of DNN deployment between EI and cloud) have started to contribute to this area, offloading modeling in EI is still an open issue.

7.2. Offloading modeling with heterogenous tasks and computing units

Most previous works on offloading modeling have assumed that the computation tasks are homogeneous. This assumption simplifies the offloading modeling process. However, in practice there are diverse tasks. For example, some tasks are preemptive (i.e., the tasks could preempt other tasks) whereas others are not. This heterogeneity of tasks significantly increases the complexity of modeling. The heterogeneity could also come from the computing units (hardware). To accelerate the computing speed, especially for the AI case, CPU + heterogeneous architectures have been proposed such as CPU + Graphics Processing Unit (GPU), CPU + Application Specific Integrated Circuit (ASIC), CPU + Field Programmable Gate Array (FPGA), even CPU + any combination of GPU, ASIC, and FPGA (Hosseinabady et al., 2019; Biookaghazadeh et al., 2018; Zhu et al., 2019). These different computing units have different computing performances. CPU has a good performance for serial computation whereas GPU is better for parallel computation and ASIC or FPGA is used for specific tasks. Although the heterogeneous architectures make the EC server more powerful, they also make the offloading modeling more challenging.

7.3. Offloading modeling with SDN technology

Edge device has limited resources, so it requires multiple EC servers to work together to provide satisfactory EC service. However, EC interoperability or inter-cooperation raises a challenge of management. SDN (Kreutz et al., 2015) which has proved its success in the networking domain is considered as a good candidate for EC inter-cooperation management (Aliyu et al., 2017). Baktir et al. (2017) argue that to realize the envisioned pervasive EC scenarios, a solution that hides all internal complexities from the users, especially from the application developers and the service providers is needed, and SDN is such a solution. Recently, the integration of EC and SDN has attracted a lot of attention. In Amadeo et al. (2019) and Rafique et al. (2020), the authors discuss how this integration can enhance EC services; EC for 5g-enabled SDN based vehicular networks is studied in Huang et al. (2017); a SDN-enabled architecture deployed at EC servers to manage IoT devices is proposed in Mavromatis et al. (2020). However, there is still a shortage of works focusing on computation offloading with SDN support. Chen and Hao (2018) have started to work on offloading with SDN, where computational and control functionalities are decoupled. Nonetheless, computation offloading in SDN integrated EC is still in its infancy and more research works are needed in order to address some open issues, especially for offloading modeling. Offloading models can be formulated for both centralized SDNs as well as distributed SDNs considering new parameters such as the timeliness of collected information.

7.4. Offloading modeling in large-scale networks

Even though we have shown that game theory, Lyapunov optimization, and machine learning can be applied to large-scale EC (Zhang et al., 2018c; Cicirelli et al., 2018), the challenge in this case still exists. The first challenge is that the large-scale network increases dramatically the complexity of all types of offloading models which in turn increases the offloading decision delay thereby causing an increase in the whole offloading delay. Both Lyapunov optimization and machine learning are centralized methods which lead to the challenge of information collection. The information collection in a large-scale network could overload the network and violate the real-time requirements of offloading decision. This is because the decision is made after receiving all the required information, and the time consumed by this process is not negligible. Moreover, highly real-time constrained large scale dynamic networks such as intelligent transportation systems or e-health infrastructures represent complex systems where the computation offloading model should also capture the heterogeneity of the architectures and their associated data.

7.5. Offloading modeling in mobility scenarios

When an ED performs a handover to another EC domain (for instance, BSs serve as EC servers and a mobile ED moves from one BS to another), it is important to guarantee the service continuity and QoS requirements (Secci et al., 2016). If the offloaded tasks run on a VM container (Tziritas et al., 2017; Sung et al., 2019), a key issue is whether this VM should be migrated to the new EC server. This issue stems from the fact that there is a tradeoff between the cost of migration and the benefit of delay and communication cost reduction. Moreover, a prediction technology is required for dynamic VM migration and optimal offloading decision. All these features make offloading modeling in mobility scenario more challenging.

7.6. Security and privacy

Security in computation offloading concerns two aspects. One is to provide mechanisms such as confidentiality, integrity, availability, access control, and authentication between EDs and EC servers so that the computation offloading process can be protected. Many of the security issues reported in this context are similar to cloud computing. However, the specific features of EC, such as limited resources of EDs and wireless access, make security mechanisms for EC more challenging (Xiao et al., 2019). The other aspect is that the EC server acts as a security agent for EDs. It is believed that the resource-constrained EDs cannot support advanced security algorithms (e.g. group signature) (Hsu et al., 2018). To address this issue, EDs' security functions should be offloaded to an EC server which executes these functions on behalf of the EDs. This mechanism requires comprehensive security protections due to the heterogeneous nature of EDs which includes various types of communication standards, dynamic security configurations (or updates), and so on. Offloading computation tasks or security functions to EC servers opens up several privacy concerns. But protecting EDs' privacy in EC is much more challenging than cloud computing. For example, using Oblivious Random Access Machine (ORAM) (Zhao et al., 2019a) to protect user access pattern in cloud computing cannot be directly applied to EC because it will compromise the latency gain brought by EC. As for the offloading model, it will also be relevant to investigate the possibility of integrating security metrics as well as privacy metrics into the model and analyze the trade-off between performance and security/privacy.

8. Conclusion

Edge computing provides a promising approach to significantly reduce network operational costs and improve QoS of mobile EDs by pushing computation resources to the network edges. Computation offloading as a "connection" between an ED and an EC server is the key challenge for EC. This paper surveys and analyzes the research works related to computation offloading modeling. We have analyzed and discussed different types of offloading modeling models based on various paradigms, such as MDP, game theory, and RL. We observe that most of them describe the computation offloading using a state-action model. Therefore, we argue that the RL-based schemes which use AI methods to address the problem described with state-action model have

some advantages over others for computation offloading for EC. This advantage also arises from the fact that offloading modeling is such a complex problem due to its features such as large-scale and unpredictable whereas the RL-based method is a powerful tool that can deal with high complexity in the real word.

We have also highlighted some research directions and challenges, among which we believe that computation offloading in EI should be the main area that needs further investigations. With the development of AI, EC will be increasingly used for AI services. Just as it is a key part for EC, computation offloading will play an important role in EI as well. Hence, modeling computation offloading in this area is critical.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We thank the anonymous reviewers for their valuable comments which helped us improve the content, organization, and presentation of this paper. This work was supported by the Applied Basic Research Program of Wuhan City, China, under grand 2017010201010117.

References

Aazam, M., Zeadally, S., Harras, K.A., 2018. Offloading in fog computing for iot: review, enabling technologies, and research opportunities. Future Generat. Comput. Syst. 87, 278–289. https://doi.org/10.1016/j.future.2018.04.057. http://www.sciencedirect.com/science/article/pii/S0167739X18301973.

Abbas, N., Zhang, Y., Taherkordi, A., Skeie, T., 2018. Mobile edge computing: a survey. IEEE Internet Things J. 5, 450–465. https://doi.org/10.1109/JIOT.2017.2750180.

Ai, Y., Peng, M., Zhang, K., 2018. Edge computing technologies for internet of things: a primer. Digit. Commun. Netw. 4, 77–86. https://doi.org/10.1016/j.dcan.2017.07.001. http://www.sciencedirect.com/science/article/pii/S2352864817301335.

Al Faruque, M.A., Vatanparvar, K., 2016. Energy management-as-a-service over fog computing platform. IEEE Internet Things J. 3, 161–169. https://doi.org/10.1109/JIOT.2015.2471260.

Ale, L., Zhang, N., Wu, H., Chen, D., Han, T., 2019. Online proactive caching in mobile edge computing using bidirectional deep recurrent neural network. IEEE Internet Things J. 6, 5520–5530. https://doi.org/10.1109/JIOT.2019.2903245.

Aliyu, S.O., Chen, F., He, Y., Yang, H., 2017. A game-theoretic based qos-aware capacity management for real-time edgeiot applications. In: 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS), pp. 386–397. https://doi.org/10.1109/QRS.2017.48.

Alrowaily, M., Lu, Z., 2018. Secure edge computing in iot systems: review and case studies. In: 2018 IEEE/ACM Symposium on Edge Computing (SEC), pp. 440–444. https://doi.org/10.1109/SEC.2018.00060.

Altman, E., 1999. Constrained Markov Decision Processes, ume 7. CRC Press.

Amadeo, M., Campolo, C., Ruggeri, G., Molinaro, A., Iera, A., 2019. Sdn-managed provisioning of named computing services in edge infrastructures. IEEE Trans. Netw. Serv. Manag. 16, 1464–1478.

Baktir, A.C., Ozgovde, A., Ersoy, C., 2017. How can edge computing benefit from software-defined networking: a survey, use cases, and future directions. IEEE Commun. Surv. Tutor. 19, 2359–2391. https://doi.org/10.1109/COMST.2017.2717482.

Baldini, I., Castro, P.C., Chang, K.S., Cheng, P., Fink, S.J., Ishakian, V., Mitchell, N., Muthusamy, V., Rabbah, R., Slominski, A., et al., 2017. Serverless Computing: Current Trends and Open Problems, pp. 1–20.

Baresi, L., Filgueira Mendona, D., 2019. Towards a serverless platform for edge computing. In: 2019 IEEE International Conference on Fog Computing (ICFC), pp. 1–10.

Beckman, P., Sankaran, R., Catlett, C., Ferrier, N., Jacob, R., Papka, M., 2016. Waggle: an open sensor platform for edge computing. In: 2016 IEEE SENSORS, pp. 1–3. https://doi.org/10.1109/ICSENS.2016.7808975.

Bedi, G., Venayagamoorthy, G.K., Singh, R., Brooks, R.R., Wang, K., 2018. Review of internet of things (iot) in electric power and energy systems. IEEE Internet Things J. 5, 847–870. https://doi.org/10.1109/JIOT.2018.2802704.

Bellavista, P., Chessa, S., Foschini, L., Gioia, L., Girolami, M., 2018. Human-enabled edge computing: exploiting the crowd as a dynamic extension of mobile edge computing. IEEE Commun. Mag. 56, 145–155.

Bi, S., Zhang, Y.A., 2018a. An admm based method for computation rate maximization in wireless powered mobile-edge computing networks. In: 2018 IEEE International Conference on Communications (ICC), pp. 1–7. https://doi.org/10.1109/ICC.2018.8422202.

Bi, S., Zhang, Y.J., 2018b. Computation rate maximization for wireless powered mobile-edge computing with binary computation offloading. IEEE Trans. Wireless Commun. 17, 4177–4190. https://doi.org/10.1109/TWC.2018.2821664.

Biookaghazadeh, S., Ren, F., Zhao, M., 2018. Are Fpgas Suitable for Edge Computing arXiv: Distributed, Parallel, and Cluster Computing.

Bowling, M., Veloso, M., 2000. An Analysis of Stochastic Game Theory for Multiagent Reinforcement Learning. Technical Report. Carnegie-Mellon Univ Pittsburgh Pa School of Computer Science.

Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J., et al., 2011. Distributed optimization and statistical learning via the alternating direction method of multipliers. Found. Trends Mach. Learn. 3, 1–122.

Bruneo, D., Distefano, S., Longo, F., Merlino, G., Puliafito, A., D'Amico, V., Sapienza, M., Torrisi, G., 2016. Stack4things as a fog computing platform for smart city applications. In: 2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pp. 848–853. https://doi.org/10.1109/INFCOMW.2016.7562195.

Cao, H., Cai, J., 2018. Distributed multiuser computation offloading for cloudlet-based mobile cloud computing: a game-theoretic machine learning approach. IEEE Trans. Veh. Technol. 67, 752–764. https://doi.org/10.1109/TVT.2017.2740724.

Cao, B., Zhang, L., Li, Y., Feng, D., Cao, W., 2019. Intelligent offloading in multi-access edge computing: a state-of-the-art review and framework. IEEE Commun. Mag. 57, 56–62. https://doi.org/10.1109/MCOM.2019.1800608.

Chang, Hyeong Soo, Fard, P.J., Marcus, S.I., Shayman, M., 2003. Multitime scale markov decision processes. IEEE Trans. Automat. Contr. 48, 976–987. https://doi.org/10.1109/TAC.2003.812782.

Chen, X., 2015. Decentralized computation offloading game for mobile cloud computing. IEEE Trans. Parallel Distr. Syst. 26, 974–983. https://doi.org/10.1109/TPDS.2014.2316834.

Chen, M., Hao, Y., 2018. Task offloading for mobile edge computing in software defined ultra-dense network. IEEE J. Sel. Area. Commun. 36, 587–597. https://doi.org/10.1109/JSAC.2018.2815360.

Chen, M., Hao, Y., Li, Y., Lai, C.F., Wu, D., 2015. On the computation offloading at ad hoc cloudlet: architecture and service modes. IEEE Commun. Mag. 53, 18–24. https://doi.org/10.1109/MCOM.2015.7120041.

Chen, X., Jiao, L., Li, W., Fu, X., 2016. Efficient multi-user computation offloading for mobile-edge cloud computing. IEEE/ACM Trans. Netw. 24, 2795–2808. https://doi.org/10.1109/TNET.2015.2487344.

Chen, W., Wang, D., Li, K., 2018. Multi-user multi-task computation offloading in green mobile edge cloud computing. IEEE Trans. Serv. Comput. https://doi.org/10.1109/TSC.2018.2826544, 11.

Chen, Y., Zhang, N., Zhang, Y., Chen, X., 2019. Dynamic computation offloading in edge computing for internet of things. IEEE Internet Things J. 6, 4242–4251. https://doi.org/10.1109/JIOT.2018.2875715.

Cheng, N., Lyu, F., Quan, W., Zhou, C., He, H., Shi, W., Shen, X., 2019. Space/aerial-assisted computing offloading for iot applications: a learning-based approach. IEEE J. Sel. Area. Commun. 37, 1117–1129. https://doi.org/10.1109/JSAC.2019.2906789.

Chiang, M., Zhang, T., 2016. Fog and iot: an overview of research opportunities. IEEE Internet Things J. 3, 854–864. https://doi.org/10.1109/JIOT.2016.2584538.

Cicirelli, F., Guerrieri, A., Spezzano, G., Vinci, A., Briante, O., Iera, A., Ruggeri, G., 2018. Edge computing and social internet of things for large-scale smart environments development. IEEE Internet Things J. 5, 2557–2571. https://doi.org/10.1109/JIOT.2017.2775739.

Consortium, O.. OpenFog reference architecture for fog computing. OpenFog consortium. https://www.openfogconsortium.org/wp-content/uploads/penFog-Reference-Architecture-2-09-17-FINAL-1.pdf.

Cuervo, E., Balasubramanian, A., Cho, D.k., Wolman, A., Saroiu, S., Chandra, R., Bahl, P., 2010. Maui: making smartphones last longer with code offload. In: ACM MobiSys 2010, Association for Computing Machinery, Inc. https://www.microsoft.com/en-us/research/publication/maui-making-smartphones-last-longer-with-code-offload/.

Dab, B., Aitsaadi, N., Langar, R., 2019. Q-learning algorithm for joint computation offloading and resource allocation in edge cloud. In: 2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), pp. 45–52.

Deep, K., Singh, K.P., Kansal, M.L., Mohan, C., 2009. A real coded genetic algorithm for solving integer and mixed integer optimization problems. Appl. Math. Comput. 212, 505–518.

Del Testa, D., Michelusi, N., Zorzi, M., 2016. Optimal transmission policies for two-user energy harvesting device networks with limited state-of-charge knowledge. IEEE Trans. Wireless Commun. 15, 1393–1405. https://doi.org/10.1109/TWC.2015.2489642.

Deng, R., Lu, R., Lai, C., Luan, T.H., 2015. Towards power consumption-delay tradeoff by workload allocation in cloud-fog computing. In: 2015 IEEE International Conference on Communications (ICC), pp. 3909–3914. https://doi.org/10.1109/ICC.2015.7248934.

Deng, M., Tian, H., Lyu, X., 2016a. Adaptive sequential offloading game for multi-cell mobile edge computing. In: 2016 23rd International Conference on Telecommunications (ICT), pp. 1–5. https://doi.org/10.1109/ICT.2016.7500395.

Deng, R., Lu, R., Lai, C., Luan, T.H., Liang, H., 2016b. Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption. IEEE Internet Things J. 3, 1171–1181. https://doi.org/10.1109/JIOT.2016.2565516.

Deng, Maofei, Tian, Hui, Fan, Bo, 2016c. Fine-granularity based application offloading policy in cloud-enhanced small cell networks. In: 2016 IEEE International Conference on Communications Workshops (ICC), pp. 638–643. https://doi.org/10.1109/ICCW.2016.7503859.

Dinh, T.Q., Tang, J., La, Q.D., Quek, T.Q.S., 2017. Offloading in mobile edge computing: task allocation and computational frequency scaling. IEEE Trans. Commun. 65, 3571–3584. https://doi.org/10.1109/TCOMM.2017.2699660.

Du, J., Zhao, L., Feng, J., Chu, X., 2018. Computation offloading and resource allocation in mixed fog/cloud computing systems with min-max fairness guarantee. IEEE Trans. Commun. 66, 1594–1608. https://doi.org/10.1109/TCOMM.2017.2787700.

Esposito, C., Castiglione, A., Pop, F., Choo, K.R., 2017. Challenges of connecting edge and cloud computing: a security and forensic perspective. IEEE Cloud Comput. 4, 13–17. https://doi.org/10.1109/MCC.2017.30.

Facchinei, F., Kanzow, C., 2007. Generalized Nash Equilibrium Problems. 4or 5, pp. 173–210.

Fudenberg, D., Tirole, J., 1993. Game Theory. MIT Press.

Grondman, I., Busoniu, L., Lopes, G.A.D., Babuska, R., 2012. A survey of actor-critic reinforcement learning: standard and natural policy gradients. IEEE Trans. Syst. Man Cybern. Part C (Applications and Reviews) 42, 1291–1307. https://doi.org/10.1109/TSMCC.2012.2218595.

Group, O.C.A.W., 2016. OpenFog architecture overview white paper. OpenFog consortium architecture working group. https://www.openfogconsortium.org/wp-content/uploads/OpenFog-Architecture-Overview-WP-2-2016.pdf.

Guo, B., Chen, C., Zhang, D., Yu, Z., Chin, A., 2016. Mobile crowd sensing and computing: when participatory sensing meets participatory social media. IEEE Commun. Mag. 54, 131–137. https://doi.org/10.1109/MCOM.2016.7402272.

Guo, F., Zhang, H., Ji, H., Li, X., Leung, V.C.M., 2018a. An efficient computation offloading management scheme in the densely deployed small cell networks with mobile edge computing. IEEE/ACM Trans. Netw. 26, 2651–2664. https://doi.org/10.1109/TNET.2018.2873002.

Guo, H., Liu, J., Qin, H., 2018b. Collaborative mobile edge computation offloading for iot over fiber-wireless networks. IEEE Netw. 32, 66–71. https://doi.org/10.1109/MNET.2018.1700139.

He, X., Chen, Y., Chai, K.K., 2018. Delay-aware energy efficient computation offloading for energy harvesting enabled fog radio access networks. In: 2018 IEEE 87th Vehicular Technology Conference (VTC Spring), pp. 1–6. https://doi.org/10.1109/VTCSpring.2018.8417646.

Hendrickson, S., Sturdevant, S., Harter, T., Venkataramani, V., Arpacidusseau, A.C., Arpacidusseau, R.H., 2016. Serverless Computation with Openlambda, pp. 33–39.

Hong, S.T., Kim, H., 2016. Qoe-aware computation offloading scheduling to capture energy-latency tradeoff in mobile clouds. In: 2016 13th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON). IEEE, pp. 1–9.

Hosseinabady, M., Zainol, M.A.B., Nunezyanez, J.L., 2019. Heterogeneous Fpga+gpu Embedded Systems: Challenges and Opportunities arXiv: Distributed, Parallel, and Cluster Computing.

Hsu, R., Lee, J., Quek, T.Q.S., Chen, J., 2018. Reconfigurable security: edge-computing-based framework for iot. IEEE Netw. 32, 92–99. https://doi.org/10.1109/MNET.2018.1700284.

Hu, H., Wang, R., 2015. User-centric local mobile cloud-assisted d2d communications in heterogeneous cloud-rans. IEEE Wirel. Commun. 22, 59–65. https://doi.org/10.1109/MWC.2015.7143327.

Hu, Y.C., Patel, M., Sabella, D., Sprecher, N., Young, V., 2015. Mobile edge computinga key technology towards 5g. ETSI White Paper 11, 1–16.

Hu, P., Dhelim, S., Ning, H., Qiu, T., 2017. Survey on fog computing: architecture, key technologies, applications and open issues. J. Netw. Comput. Appl. 98, 27–42.

Hu, X., Wong, K., Yang, K., 2018. Wireless powered cooperation-assisted mobile edge computing. IEEE Trans. Wireless Commun. 17, 2375–2388. https://doi.org/10.1109/TWC.2018.2794345.

Huang, D., Wang, P., Niyato, D., 2012. A dynamic offloading algorithm for mobile computing. IEEE Trans. Wireless Commun. 11, 1991–1995. https://doi.org/10.1109/TWC.2012.041912.110912.

Huang, X., Yu, R., Kang, J., He, Y., Zhang, Y., 2017. Exploring mobile edge computing for 5g-enabled software defined vehicular networks. IEEE Wirel. Commun. 24, 55–63. https://doi.org/10.1109/MWC.2017.1600387.

Ismail, B., Mostajeran, E., Bazli Ab Karim, M., Ming Tat, W., Setapa, S., Luke, J.Y., Ong, H., 2015. Evaluation of Docker as Edge Computing Platform. https://doi.org/10.1109/ICOS.2015.7377291.

Jararweh, Y., Doulat, A., Darabseh, A., Alsmirat, M., Al-Ayyoub, M., Benkhelifa, E., 2016. Sdmec: software defined system for mobile edge computing. In: 2016 IEEE International Conference on Cloud Engineering Workshop (IC2EW), pp. 88–93.

Ji, L., Guo, S., 2019. Energy-efficient cooperative resource allocation in wireless powered mobile edge computing. IEEE Internet Things J. https://doi.org/10.1109/JIOT.2018.2880812, 11.

Jiang, C., Cheng, X., Gao, H., Zhou, X., Wan, J., 2019. Toward computation offloading in edge computing: a survey. IEEE Access 7, 131543–131558. https://doi.org/10.1109/ACCESS.2019.2938660.

Joilo, S., Dn, G., 2017. A game theoretic analysis of selfish mobile computation offloading. In: IEEE INFOCOM 2017 - IEEE Conference on Computer Communications, pp. 1–9. https://doi.org/10.1109/INFOCOM.2017.8057148.

Joilo, S., Dn, G., 2019. Decentralized algorithm for randomized task allocation in fog computing systems. IEEE/ACM Trans. Netw. 27, 85–97. https://doi.org/10.1109/TNET.2018.2880874.

Kahan, J.P., Rapoport, A., 2014. Theories of Coalition Formation. Psychology Press.

Kamoun, M., Labidi, W., Sarkiss, M., 2015. Joint resource allocation and offloading strategies in cloud enabled cellular networks. In: 2015 IEEE International Conference on Communications (ICC), pp. 5529–5534. https://doi.org/10.1109/ICC.2015.7249203.

Kim, S., Vyas, R., Bito, J., Niotaki, K., Collado, A., Georgiadis, A., Tentzeris, M.M., 2014. Ambient rf energy-harvesting technologies for self-sustainable standalone wireless sensor platforms. Proc. IEEE 102, 1649–1666. https://doi.org/10.1109/JPROC.2014.2357031.

Kim, Y., An, N., Park, J., Lim, H., 2018. Mobility support for vehicular cloud radio-access-networks with edge computing. In: 2018 IEEE 7th International Conference on Cloud Networking (CloudNet), pp. 1–4. https://doi.org/10.1109/CloudNet.2018.8549365.

Ko, H., Lee, J., Pack, S., 2018. Spatial and temporal computation offloading decision algorithm in edge cloud-enabled heterogeneous networks. IEEE Access 6, 18920–18932. https://doi.org/10.1109/ACCESS.2018.2818111.

Kreutz, D., Ramos, F.M.V., Verssimo, P.E., Rothenberg, C.E., Azodolmolky, S., Uhlig, S., 2015. Software-defined networking: a comprehensive survey. Proc. IEEE 103, 14–76. https://doi.org/10.1109/JPROC.2014.2371999.

Ku, M., Li, W., Chen, Y., Ray Liu, K.J., 2016. Advances in energy harvesting communications: past, present, and future challenges. IEEE Commun. Surv. Tutor. 18, 1384–1412. https://doi.org/10.1109/COMST.2015.2497324.

Labidi, W., Sarkiss, M., Kamoun, M., 2015. Energy-optimal resource scheduling and computation offloading in small cell networks. In: 2015 22nd International Conference on Telecommunications (ICT), pp. 313–318. https://doi.org/10.1109/ICT.2015.7124703.

Lau, T.E., Ho, Y.C., 1997. Universal alignment probabilities and subset selection for ordinal optimization. J. Optim. Theor. Appl. 93, 455–489.

Le, D.V., Tham, C., 2017. An optimization-based approach to offloading in ad-hoc mobile clouds. In: GLOBECOM 2017 - 2017 IEEE Global Communications Conference, pp. 1–6. https://doi.org/10.1109/GLOCOM.2017.8254632.

Li, K., 2019. Computation offloading strategy optimization with multiple heterogeneous servers in mobile edge computing. IEEE Trans. Sustain. Comput. https://doi.org/10.1109/TSUSC.2019.2904680, 11.

Li, S., Huang, J., 2017. Energy efficient resource management and task scheduling for iot services in edge computing paradigm. In: 2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC), pp. 846–851. https://doi.org/10.1109/ISPA/IUCC.2017.00129.

Li, H., Jaggi, N., Sikdar, B., 2011. Relay scheduling for cooperative communications in sensor networks with energy harvesting. IEEE Trans. Wireless Commun. 10, 2918–2928. https://doi.org/10.1109/TWC.2011.070711.100778.

Li, H., Ota, K., Dong, M., 2018a. Learning iot in edge: deep learning for the internet of things with edge computing. IEEE Netw. 32, 96–101. https://doi.org/10.1109/MNET.2018.1700202.

Li, J., Gao, H., Lv, T., Lu, Y., 2018b. Deep reinforcement learning based computation offloading and resource allocation for mec. In: 2018 IEEE Wireless Communications and Networking Conference (WCNC), pp. 1–6. https://doi.org/10.1109/WCNC.2018.8377343.

Lin, H., Chen, Z., Wang, L., 2019a. Offloading for edge computing in low power wide area networks with energy harvesting. IEEE Access 7, 78919–78929. https://doi.org/10.1109/ACCESS.2019.2922399.

Lin, L., Liao, X., Jin, H., Li, P., 2019b. Computation offloading toward edge computing. Proc. IEEE 107, 1584–1607.

Liu, J., Mao, Y., Zhang, J., Letaief, K.B., 2016a. Delay-optimal computation task scheduling for mobile-edge computing systems. In: 2016 IEEE International Symposium on Information Theory (ISIT), pp. 1451–1455. https://doi.org/10.1109/ISIT.2016.7541539.

Liu, Y., Niu, D., Li, B., 2016b. Delay-optimized video traffic routing in software-defined interdatacenter networks. IEEE Trans. Multimed. 18, 865–878. https://doi.org/10.1109/TMM.2016.2538718.

Liu, Y., Xu, C., Zhan, Y., Liu, Z., Guan, J., Zhang, H., 2017. Incentive mechanism for computation offloading using edge computing: a stackelberg game approach. Comput. Network. 129, 399–409.

Liu, L., Chang, Z., Guo, X., 2018a. Socially aware dynamic computation offloading scheme for fog computing system with energy harvesting devices. IEEE Internet Things J. 5, 1869–1879. https://doi.org/10.1109/JIOT.2018.2816682.

Liu, L., Chang, Z., Guo, X., Mao, S., Ristaniemi, T., 2018b. Multiobjective optimization for computation offloading in fog computing. IEEE Internet Things J. 5, 283–294. https://doi.org/10.1109/JIOT.2017.2780236.

Lobillo, F., Becvar, Z., Puente, M.A., Mach, P., Lo Presti, F., Gambetti, F., Goldhamer, M., Vidal, J., Widiawan, A.K., Calvanesse, E., 2014. An architecture for mobile computation offloading on cloud-enabled lte small cells. In: 2014 IEEE Wireless Communications and Networking Conference Workshops (WCNCW), pp. 1–6.

Lorenzo, B., Garcia-Rois, J., Li, X., Gonzalez-Castano, J., Fang, Y., 2018. A robust dynamic edge network architecture for the internet of things. IEEE Netw. 32, 8–15. https://doi.org/10.1109/MNET.2018.1700263.

Lyu, X., Ni, W., Tian, H., Liu, R.P., Wang, X., Giannakis, G.B., Paulraj, A., 2017. Optimal schedule of mobile edge computing for internet of things using partial information. IEEE J. Sel. Area. Commun. 35, 2606–2615. https://doi.org/10.1109/JSAC.2017.2760186.

Mach, P., Becvar, Z., 2017. Mobile edge computing: a survey on architecture and computation offloading. IEEE Commun. Surv. Tutor. 19, 1628–1656. https://doi.org/10.1109/COMST.2017.2682318.

Mao, Y., Zhang, J., Letaief, K.B., 2016a. Dynamic computation offloading for mobile-edge computing with energy harvesting devices. IEEE J. Sel. Area. Commun. 34, 3590–3605. https://doi.org/10.1109/JSAC.2016.2611964.

Mao, Y., Zhang, J., Song, S.H., Letaief, K.B., 2016b. Power-delay tradeoff in multi-user mobile-edge computing systems. In: 2016 IEEE Global Communications Conference (GLOBECOM), pp. 1–6. https://doi.org/10.1109/GLOCOM.2016.7842160.

Mao, Y., You, C., Zhang, J., Huang, K., Letaief, K.B., 2017. A survey on mobile edge computing: the communication perspective. IEEE Commun. Surv. Tutor. 19, 2322–2358. https://doi.org/10.1109/COMST.2017.2745201.

Marjanovi, M., Antoni, A., arko, I.P., 2018a. Edge computing architecture for mobile crowdsensing. IEEE Access 6, 10662–10674. https://doi.org/10.1109/ACCESS.2018.2799707.

Marjanovi, M., Antoni, A., arko, I.P., 2018b. Edge computing architecture for mobile crowdsensing. IEEE Access 6, 10662–10674. https://doi.org/10.1109/ACCESS.2018.2799707.

Mavromatis, A., Colman-Meixner, C., Silva, A.P., Vasilakos, X., Nejabati, R., Simeonidou, D., 2020. A software-defined iot device management framework for edge and cloud computing. IEEE Internet Things J. 7, 1718–1735.

Mehrabi, M., You, D., Latzko, V., Salah, H., Reisslein, M., Fitzek, F.H.P., 2019. Device-enhanced mec: multi-access edge computing (mec) aided by end device computation and caching: a survey. IEEE Access 7, 166079–166108. https://doi.org/10.1109/ACCESS.2019.2953172.

Meng, X., Wang, W., Wang, Y., Lau, V.K.N., Zhang, Z., 2018. Delay-optimal computation offloading for computation-constrained mobile edge networks. In: 2018 IEEE Global Communications Conference (GLOBECOM), pp. 1–7. https://doi.org/10.1109/GLOCOM.2018.8647703.

Meyn, S.P., Tweedie, R.L., 2012. Markov Chains and Stochastic Stability. Springer Science & Business Media.

Michelusi, N., Stamatiou, K., Zorzi, M., 2013. Transmission policies for energy harvesting sensors with time-correlated energy supply. IEEE Trans. Commun. 61, 2988–3001. https://doi.org/10.1109/TCOMM.2013.052013.120565.

Michelusi, N., Badia, L., Zorzi, M., 2014. Optimal transmission policies for energy harvesting devices with limited state-of-charge knowledge. IEEE Trans. Commun. 62, 3969–3982. https://doi.org/10.1109/TCOMM.2014.2359009.

Min, M., Xiao, L., Chen, Y., Cheng, P., Wu, D., Zhuang, W., 2019. Learning-based computation offloading for iot devices with energy harvesting. IEEE Trans. Veh. Technol. 68, 1930–1941. https://doi.org/10.1109/TVT.2018.2890685.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al., 2015. Human-level control through deep reinforcement learning. Nature 518, 529.

Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., Kavukcuoglu, K., 2016. Asynchronous methods for deep reinforcement learning. In: International Conference on Machine Learning, pp. 1928–1937.

Monderer, D., Shapley, L.S., 1996. Potential games. Game. Econ. Behav. 14, 124–143. https://doi.org/10.1006/game.1996.0044. http://www.sciencedirect.com/science/article/pii/S0899825696900445.

Mukherjee, M., Shu, L., Wang, D., Li, K., Chen, Y., 2017. A fog computing-based framework to reduce traffic overhead in large-scale industrial applications. In: 2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pp. 1008–1009. https://doi.org/10.1109/INFCOMW.2017.8116534.

Mukherjee, M., Shu, L., Wang, D., 2018. Survey of fog computing: fundamental, network applications, and research challenges. IEEE Commun. Surv. Tutor. 20, 1826–1857. https://doi.org/10.1109/COMST.2018.2814571.

Munir, M.S., Abedin, S.F., Tran, N.H., Hong, C.S., 2019. When edge computing meets microgrid: a deep reinforcement learning approach. IEEE Internet Things J. https://doi.org/10.1109/JIOT.2019.2899673, 11.

Muoz, O., Pascual-Iserte, A., Vidal, J., 2015. Optimization of radio and computational resources for energy efficiency in latency-constrained application offloading. IEEE Trans. Veh. Technol. 64, 4738–4755. https://doi.org/10.1109/TVT.2014.2372852.

Neely, M.J., Huang, L., 2010. Dynamic product assembly and inventory control for maximum profit. In: 49th IEEE Conference on Decision and Control (CDC), pp. 2805–2812. https://doi.org/10.1109/CDC.2010.5717235.

Neto, J.L.D., Yu, S., Macedo, D.F., Nogueira, J.M.S., Langar, R., Secci, S., 2018. Uloof: a user level online offloading framework for mobile edge computing. IEEE Trans. Mobile Comput. 17, 2660–2674.

Oueis, J., Strinati, E.C., Barbarossa, S., 2014. Small cell clustering for efficient distributed cloud computing. In: 2014 IEEE 25th Annual International Symposium on Personal, Indoor, and Mobile Radio Communication (PIMRC), pp. 1474–1479. https://doi.org/10.1109/PIMRC.2014.7136401.

Oueis, J., Strinati, E.C., Sardellitti, S., Barbarossa, S., 2015. Small cell clustering for efficient distributed fog computing: a multi-user case. In: 2015 IEEE 82nd Vehicular Technology Conference (VTC2015-Fall), pp. 1–5. https://doi.org/10.1109/VTCFall.2015.7391144.

Ozel, O., Tutuncuoglu, K., Yang, J., Ulukus, S., Yener, A., 2011. Transmission with energy harvesting nodes in fading wireless channels: optimal policies. IEEE J. Sel. Area. Commun. 29, 1732–1743.

Plastiras, G., Terzi, M., Kyrkou, C., Theocharidcs, T., 2018. Edge intelligence: challenges and opportunities of near-sensor machine learning applications. In: 2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP), pp. 1–7. https://doi.org/10.1109/ASAP.2018.8445118.

Porambage, P., Okwuibe, J., Liyanage, M., Ylianttila, M., Taleb, T., 2018. Survey on multi-access edge computing for internet of things realization. IEEE Commun. Surv. Tutor. 20, 2961–2991. https://doi.org/10.1109/COMST.2018.2849509.

Pu, L., Chen, X., Xu, J., Fu, X., 2016. D2d fogging: an energy-efficient and incentive-aware task offloading framework via network-assisted d2d collaboration. IEEE J. Sel. Area. Commun. 34, 3887–3901. https://doi.org/10.1109/JSAC.2016.2624118.

Puterman, M., 2005. Markov Decision Processes: Discrete Stochastic Dynamic Programming, second ed., p. 37

Qi, H., Gani, A., 2012. Research on mobile cloud computing: review, trend and perspectives. In: 2012 Second International Conference on Digital Information and Communication Technology and It's Applications (DICTAP). ieee, pp. 195–202.

Rafique, W., Qi, L., Yaqoob, I., Imran, M., Rasool, R.u., Dou, W., 2020. Complementing iot services through software defined networking and edge computing: a comprehensive survey. IEEE Commun. Surv. Tutor. early access.

Ranadheera, S., Maghsudi, S., Hossain, E., 2017. Mobile Edge Computation Offloading Using Game Theory and Reinforcement Learning arXiv preprint arXiv:1711.09012.

Ren, J., Yu, G., Cai, Y., He, Y., Qu, F., 2017. Partial offloading for latency minimization in mobile-edge computing. In: GLOBECOM 2017 - 2017 IEEE Global Communications Conference, pp. 1–6. https://doi.org/10.1109/GLOCOM.2017.8254550.

Roman, R., Lopez, J., Mambo, M., 2018. Mobile edge computing, fog etal.: a survey and analysis of security threats and challenges. Future Generat. Comput. Syst. 78, 680–698. https://doi.org/10.1016/j.future.2016.11.009. http://www.sciencedirect.com/science/article/pii/S0167739X16305635.

Ryder, B.G., 1979. Constructing the call graph of a program. IEEE Trans. Software Eng. SE-5, 216–226. https://doi.org/10.1109/TSE.1979.234183.

Sabella, D., Vaillant, A., Kuure, P., Rauschenbach, U., Giust, F., 2016. Mobile-edge computing architecture: the role of mec in the internet of things. IEEE Consum. Electron. Mag. 5, 84–91. https://doi.org/10.1109/MCE.2016.2590118.

Salodkar, N., Bhorkar, A., Karandikar, A., Borkar, V.S., 2008. An on-line learning algorithm for energy efficient delay constrained scheduling over a fading channel. IEEE J. Sel. Area. Commun. 26, 732–742. https://doi.org/10.1109/JSAC.2008.080514.

Samanta, A., Chang, Z., 2019. Adaptive service offloading for revenue maximization in mobile edge computing with delay-constraint. IEEE Internet Things J. 6, 3864–3872.

Samanta, A., Tang, J., 2020. Dyme: dynamic microservice scheduling in edge computing enabled iot. IEEE Internet Things J. 11.

Sardellitti, S., Scutari, G., Barbarossa, S., 2015. Joint optimization of radio and computational resources for multicell mobile-edge computing. IEEE Trans. Signal Inf. Process. Netw. 1, 89–103. https://doi.org/10.1109/TSIPN.2015.2448520.

Sarkar, S., Wankar, R., Srirama, S.N., Suryadevara, N.K., 2020. Serverless management of sensing systems for fog computing framework. IEEE Sensor. J. 20, 1564–1572.

Satyanarayanan, M., 2017. The emergence of edge computing. Computer 50, 30–39. https://doi.org/10.1109/MC.2017.9.

Satyanarayanan, M., Bahl, P., Caceres, R., Davies, N., 2009. The case for vm-based cloudlets in mobile computing. IEEE Pervasive Comput. 8, 14–23. https://doi.org/10.1109/MPRV.2009.82

Satyanarayanan, M., Chen, Z., Ha, K., Hu, W., Richter, W., Pillai, P., 2014. Cloudlets: at the leading edge of mobile-cloud convergence. In: 6th International Conference on Mobile Computing, Applications and Services, pp. 1–9. https://doi.org/10.4108/icst.mobicase.2014.257757.

Scutari, G., Facchinei, F., Lampariello, L., 2017. Parallel and distributed methods for constrained nonconvex optimizationpart i: Theory. IEEE Trans. Signal Process. 65, 1929–1944. https://doi.org/10.1109/TSP.2016.2637317.

Secci, S., Raad, P., Gallard, P., 2016. Linking virtual machine mobility to user mobility. IEEE Trans. Netw. Serv. Manag. 13, 927–940. https://doi.org/10.1109/TNSM.2016.2592241.

Sen, T., Shen, H., 2019. Machine learning based timeliness-guaranteed and energy-efficient task assignment in edge computing systems. In: 2019 IEEE 3rd International Conference on Fog and Edge Computing (ICFEC), pp. 1–10. https://doi.org/10.1109/CFEC.2019.8733153.

Shaikh, F.K., Zeadally, S., 2016. Energy harvesting in wireless sensor networks: a comprehensive review. Renew. Sustain. Energy Rev. 55, 1041–1054. https://doi.org/10.1016/j.rser.2015.11.010. http://www.sciencedirect.com/science/article/pii/S1364032115012629.

Shan, X., Zhi, H., Li, P., Han, Z., 2018. A survey on computation offloading for mobile edge computing information. In: 2018 IEEE 4th International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing, (HPSC) and IEEE International Conference on Intelligent Data and Security (IDS), pp. 248–251. https://doi.org/10.1109/BDS/HPSC/IDS18.2018.00060.

Shi, W., Cao, J., Zhang, Q., Li, Y., Xu, L., 2016. Edge computing: vision and challenges. IEEE Internet Things J. 3, 637–646. https://doi.org/10.1109/JIOT.2016.2579198.

Shi, Y., Chen, S., Xu, X., 2018. Maga: a mobility-aware computation offloading decision for distributed mobile cloud computing. IEEE Internet Things J. 5, 164–174. https://doi.org/10.1109/JIOT.2017.2776252.

Standard, E.G.M., 2016. Mobile Edge Computing (MEC); Framework and Reference Architecture v1.1.1. ETSI GS MEC Standard.

Suganuma, T., Oide, T., Kitagami, S., Sugawara, K., Shiratori, N., 2018. Multiagent-based flexible edge computing architecture for iot. IEEE Netw. 32, 16–23. https://doi.org/10.1109/MNET.2018.1700201.

Suh, J., Hougen, D.F., 2017. The context-aware learning model: reward-based and experience-based logistic regression backpropagation. In: 2017 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 1–8. https://doi.org/10.1109/SSCI.2017.8285401.

Sun, X., Ansari, N., 2016. Edgeiot: mobile edge computing for the internet of things. IEEE Commun. Mag. 54, 22–29. https://doi.org/10.1109/MCOM.2016.1600492CM.

Sun, H., Zhou, F., Hu, R.Q., 2019. Joint offloading and computation energy efficiency maximization in a mobile edge computing system. IEEE Trans. Veh. Technol. 68, 3052–3056. https://doi.org/10.1109/TVT.2019.2893094.

Sung, J., Han, S., Kim, J., 2019. Virtual machine pre-provisioning for computation offloading service in edge cloud. In: 2019 IEEE 12th International Conference on Cloud Computing (CLOUD), pp. 490–492. https://doi.org/10.1109/CLOUD.2019.00087.

Sutton, R.S., Barto, A.G., et al., 1998. Introduction to Reinforcement Learning, ume 135. MIT press, Cambridge.

Taleb, T., Samdanis, K., Mada, B., Flinck, H., Dutta, S., Sabella, D., 2017. On multi-access edge computing: a survey of the emerging 5g network edge cloud architecture and orchestration. IEEE Commun. Surv. Tutor. 19, 1657–1681. https://doi.org/10.1109/COMST.2017.2705720.

Tan, L.T., Hu, R.Q., 2018. Mobility-aware edge caching and computing in vehicle networks: a deep reinforcement learning. IEEE Trans. Veh. Technol. 67, 10190–10203. https://doi.org/10.1109/TVT.2018.2867191.
Tanaka, H., Yoshida, M., Mori, K., Takahashi, N., 2018. Multi-access edge computing: a survey. J. Inf. Process. 26, 87–97.
Tanzil, S.M.S., Gharehshiran, O.N., Krishnamurthy, V., 2015. Femto-cloud formation: a coalitional game-theoretic approach. In: 2015 IEEE Global Communications Conference (GLOBECOM), pp. 1–6. https://doi.org/10.1109/GLOCOM.2015.7417264.
Tong, L., Li, Y., Gao, W., 2016. A hierarchical edge cloud architecture for mobile computing. In: IEEE INFOCOM 2016 - the 35th Annual IEEE International Conference on Computer Communications, pp. 1–9. https://doi.org/10.1109/INFOCOM.2016.7524340.
Tran, T.X., Pompili, D., 2018. Joint task offloading and resource allocation for multi-server mobile-edge computing networks. IEEE Trans. Veh. Technol. 68, 856–868.
Tziritas, N., Koziri, M., Bachtsevani, A., Loukopoulos, T., Stamoulis, G., Khan, S.U., Xu, C., 2017. Data replication and virtual machine migrations to mitigate network overhead in edge computing systems. IEEE Trans. Sustain. Comput. 2, 320–332. https://doi.org/10.1109/TSUSC.2017.2715662.
Varsha, H.S., Shashikala, K.P., 2017. The tactile internet. In: 2017 International Conference on Innovative Mechanisms for Industry Applications (ICIMIA), pp. 419–422. https://doi.org/10.1109/ICIMIA.2017.7975649.
Vita, F.D., Bruneo, D., Puliafito, A., Nardini, G., Virdis, A., Stea, G., 2018. A deep reinforcement learning approach for data migration in multi-access edge computing. In: 2018 ITU Kaleidoscope: Machine Learning for a 5G Future (ITU K), pp. 1–8. https://doi.org/10.23919/ITU-WT.2018.8597889.
Wang, F., Zhang, X., 2018. Dynamic interface-selection and resource allocation over heterogeneous mobile edge-computing wireless networks with energy harvesting. In: IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pp. 190–195. https://doi.org/10.1109/INFCOMW.2018.8406978.
Wang, Z., Tajer, A., Wang, X., 2012. Communication of energy harvesting tags. IEEE Trans. Commun. 60, 1159–1166. https://doi.org/10.1109/TCOMM.2012.022912.110298.
Wang, S., Tu, G., Ganti, R., He, T., Leung, K., Tripp, H., Warr, K., Zafer, M., 2013. Mobile Micro-cloud: Application Classification, Mapping, and Deployment.
Wang, Y., Sheng, M., Wang, X., Wang, L., Li, J., 2016. Mobile-edge computing: partial computation offloading using dynamic voltage scaling. IEEE Trans. Commun. 64, 4268–4282. https://doi.org/10.1109/TCOMM.2016.2599530.
Wang, F., Xu, J., Wang, X., Cui, S., 2018a. Joint offloading and computing optimization in wireless powered mobile-edge computing systems. IEEE Trans. Wireless Commun. 17, 1784–1797. https://doi.org/10.1109/TWC.2017.2785305.
Wang, S., Xu, J., Zhang, N., Liu, Y., 2018b. A survey on service migration in mobile edge computing. IEEE Access 6, 23511–23528. https://doi.org/10.1109/ACCESS.2018.2828102.
Wang, Y., Wang, K., Huang, H., Miyazaki, T., Guo, S., 2019. Traffic and computation co-offloading with reinforcement learning in fog computing for industrial applications. IEEE Trans. Ind. Inf. 15, 976–986. https://doi.org/10.1109/TII.2018.2883991.
Waqas, M., Niu, Y., Ahmed, M., Li, Y., Jin, D., Han, Z., 2019. Mobility-aware fog computing in dynamic environments: understandings and implementation. IEEE Access 7, 38867–38879. https://doi.org/10.1109/ACCESS.2018.2883662.
Wei, Z., Zhao, B., Su, J., Lu, X., 2019. Dynamic edge computation offloading for internet of things with energy harvesting: a learning method. IEEE Internet Things J. 1 https://doi.org/10.1109/JIOT.2018.2882783.
Wu, H., 2018. Multi-objective decision-making for mobile cloud offloading: a survey. IEEE Access 6, 3962–3976. https://doi.org/10.1109/ACCESS.2018.2791504.
Wu, H., Chen, L., Shen, C., Wen, W., Xu, J., 2018. Online geographical load balancing for energy-harvesting mobile edge computing. In: 2018 IEEE International Conference on Communications (ICC), pp. 1–6. https://doi.org/10.1109/ICC.2018.8422299.
Xiao, Y., Krunz, M., 2017. Qoe and power efficiency tradeoff for fog computing networks with fog node cooperation. In: IEEE INFOCOM 2017 - IEEE Conference on Computer Communications, pp. 1–9. https://doi.org/10.1109/INFOCOM.2017.8057196.
Xiao, Y., Jia, Y., Liu, C., Cheng, X., Yu, J., Lv, W., 2019. Edge computing security: state of the art and challenges. Proc. IEEE 107, 1608–1631. https://doi.org/10.1109/JPROC.2019.2918437.
Xu, J., Chen, L., Ren, S., 2017. Online learning for offloading and autoscaling in energy harvesting mobile edge computing. IEEE Trans. Cognit. Commun. Netw. 3, 361–373. https://doi.org/10.1109/TCCN.2017.2725277.
Xu, X., Li, D., Dai, Z., Li, S., Chen, X., 2019. A heuristic offloading method for deep learning edge services in 5g networks. IEEE Access 7, 67734–67744. https://doi.org/10.1109/ACCESS.2019.2918585.
Yaqoob, I., Ahmed, E., Gani, A., Mokhtar, S., Imran, M., Guizani, S., 2016. Mobile ad hoc cloud: a survey. Wireless Commun. Mobile Comput. 16, 2572–2589.
Yiqing, L., Xigang, Y., Yongjian, L., 2007. An improved pso algorithm for solving non-convex nlp/minlp problems with equality constraints. Comput. Chem. Eng. 31, 153–162.
You, C., Huang, K., Chae, H., Kim, B., 2017. Energy-efficient resource allocation for mobile-edge computation offloading. IEEE Trans. Wireless Commun. 16, 1397–1411. https://doi.org/10.1109/TWC.2016.2633522.
Yu, Y., 2016. Mobile edge computing towards 5g: vision, recent progress, and open challenges. China Commun. 13, 89–99. https://doi.org/10.1109/CC.2016.7833463.
Yu, W., Liang, F., He, X., Hatcher, W.G., Lu, C., Lin, J., Yang, X., 2018. A survey on the edge computing for the internet of things. IEEE Access 6, 6900–6919. https://doi.org/10.1109/ACCESS.2017.2778504.
Zhang, X., Wang, J., 2018. Joint heterogeneous statistical-qos/qoe provisionings for edge-computing based wifi offloading over 5g mobile wireless networks. In: 2018 52nd Annual Conference on Information Sciences and Systems. CISS), pp. 1–6. https://doi.org/10.1109/CISS.2018.8362265.
Zhang, Y., Niyato, D., Wang, P., 2015. Offloading in mobile cloudlet systems with intermittent connectivity. IEEE Trans. Mobile Comput. 14, 2516–2529. https://doi.org/10.1109/TMC.2015.2405539.
Zhang, K., Mao, Y., Leng, S., Zhao, Q., Li, L., Peng, X., Pan, L., Maharjan, S., Zhang, Y., 2016a. Energy-efficient offloading for mobile edge computing in 5g heterogeneous networks. IEEE Access 4, 5896–5907. https://doi.org/10.1109/ACCESS.2016.2597169.
Zhang, S., Zhang, N., Zhou, S., Gong, J., Niu, Z., Shen, X., 2016b. Energy-aware traffic offloading for green heterogeneous networks. IEEE J. Sel. Area. Commun. 34, 1116–1129. https://doi.org/10.1109/JSAC.2016.2520244.
Zhang, K., Mao, Y., Leng, S., Maharjan, S., Zhang, Y., 2017. Optimal delay constrained offloading for vehicular edge computing networks. In: 2017 IEEE International Conference on Communications (ICC), pp. 1–6. https://doi.org/10.1109/ICC.2017.7997360.
Zhang, G., Chen, Y., Shen, Z., Wang, L., 2018a. Energy management for multi-user mobile-edge computing systems with energy harvesting devices and qos constraints. In: 2018 27th International Conference on Computer Communication and Networks (ICCCN), pp. 1–6. https://doi.org/10.1109/ICCCN.2018.8487435.
Zhang, G., Zhang, W., Cao, Y., Li, D., Wang, L., 2018b. Energy-delay tradeoff for dynamic offloading in mobile-edge computing system with energy harvesting devices. IEEE Trans. Ind. Inf. 14, 4642–4655. https://doi.org/10.1109/TII.2018.2843365.
Zhang, Y., Zhao, Z., Shu, C., Min, G., Wang, Z., 2018c. Embedding virtual network functions with backup for reliable large-scale edge computing. In: 2018 5th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2018 4th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom), pp. 190–195. https://doi.org/10.1109/CSCloud/EdgeCom.2018.00041.
Zhang, Z., Yu, F., Fu, F., Yan, Q., Wang, Z., 2018d. Joint Offloading and Resource Allocation in Mobile Edge Computing Systems: an Actor-Critic Approach, pp. 1–6. https://doi.org/10.1109/GLOCOM.2018.8647593.
Zhang, D., Tan, L., Ren, J., Awad, M.K., Zhang, S., Zhang, Y., Wan, P., 2019. Near-optimal and truthful online auction for computation offloading in green edge-computing systems. IEEE Trans. Mobile Comput. 1 https://doi.org/10.1109/TMC.2019.2901474.
Zhao, H., Du, W., Liu, W., Lei, T., Lei, Q., 2018a. Qoe aware and cell capacity enhanced computation offloading for multi-server mobile edge computing systems with energy harvesting devices. In: 2018 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computing, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), pp. 671–678. https://doi.org/10.1109/SmartWorld.2018.00133.
Zhao, S., Yang, Y., Shao, Z., Yang, X., Qian, H., Wang, C., 2018b. Femos: fog-enabled multitier operations scheduling in dynamic wireless networks. IEEE Internet Things J. 5, 1169–1183. https://doi.org/10.1109/JIOT.2018.2808280.
Zhao, Y., Leung, V.C.M., Gao, H., Chen, Z., Ji, H., 2018c. Uplink resource allocation in mobile edge computing-based heterogeneous networks with multi-band rf energy harvesting. In: 2018 IEEE International Conference on Communications (ICC), pp. 1–6. https://doi.org/10.1109/ICC.2018.8422201.
Zhao, B., Chen, Z., Lin, H., 2019a. Cycle oram: a practical protection for access pattern in untrusted storage. IEEE Access 7, 26684–26695. https://doi.org/10.1109/ACCESS.2019.2900304.
Zhao, Y., Wang, W., Li, Y., Colman Meixner, C., Tornatore, M., Zhang, J., 2019b. Edge computing and networking: a survey on infrastructures and applications. IEEE Access 7, 101213–101230. https://doi.org/10.1109/ACCESS.2019.2927538.
Zheng, J., Cai, Y., Wu, Y., Shen, X.S., 2016. Stochastic computation offloading game for mobile cloud computing. In: 2016 IEEE/CIC International Conference on Communications in China (ICCC), pp. 1–6. https://doi.org/10.1109/ICCChina.2016.7636777.
Zheng, J., Cai, Y., Wu, Y., Shen, X., 2019a. Dynamic computation offloading for mobile cloud computing: a stochastic game-theoretic approach. IEEE Trans. Mobile Comput. 18, 771–786. https://doi.org/10.1109/TMC.2018.2847337.
Zheng, X., Li, M., Tahir, M., Chen, Y., Alam, M., 2019b. Stochastic computation offloading and scheduling based on mobile edge computing. IEEE Access. https://doi.org/10.1109/ACCESS.2019.2919651, 11.
Zhou, F., Wu, Y., Hu, R.Q., Qian, Y., 2018. Computation rate maximization in uav-enabled wireless-powered mobile-edge computing systems. IEEE J. Sel. Area. Commun. 36, 1927–1941. https://doi.org/10.1109/JSAC.2018.2864426.
Zhou, Z., Chen, X., Li, E., Zeng, L., Luo, K., Zhang, J., 2019. Edge intelligence: paving the last mile of artificial intelligence with edge computing. Proc. IEEE 1–25. https://doi.org/10.1109/JPROC.2019.2918951.
Zhu, Z., Zhang, J., Zhao, J., Cao, J., Zhao, D., Jia, G., Meng, Q., 2019. A hardware and software task-scheduling framework based on cpu+fpga heterogeneous architecture in edge computing. IEEE Access 7, 148975–148988.

Hai Lin received the B.S. degree from the Department of Thermal Engineering, Huazhong Sciences and Technologies University, Wuhan, China, in 1999, the M.S. degree in computer science from the University of Pierre and Marie Curie, Paris, France, in 2005, and the Ph.D. degree from the Institute Telecom–Telecom ParisTech, Paris, in 2008. He held Postdoctoral Research with France Telecom. He was a Researcher with ZTE Europe. Since 2012, he has been with Wuhan University, Wuhan, where he is currently an Associate Professor with the School of Cyber Science and Engineering. His research interests include the Internet of Things, edge computing, sensor networks, and future networks.

Sherali Zeadally earned his bachelor's degree in computer science from the University of Cambridge, England. He also received a doctoral degree in computer science from the University of Buckingham, England, followed by postdoctoral research at the University of Southern California, Los Angeles, CA. He is currently an Associate Professor in the College of Communication and Information, University of Kentucky. His research interests include Cybersecurity, privacy, Internet of Things, computer networks, and energy-efficient networking. He is a Fellow of the British Computer Society and the Institution of Engineering Technology, England.

Zhihong Chen received the B.S. degree in information security from Wuhan University, Wuhan, China, in 2007, and the M.S. degree in information security from the University of Electronic Science and Technology of China, in 2010. She was a Software Engineer with ZTE from 2010 to 2012. She is currently pursuing the Ph.D. degree in information security with the School of Cyber Science and Engineering with Wuhan University. Her research interests include system security, trusted computing, and cloud computing security.

Houda Labiod is a Professor at Department INFRES (Computer Science and Network department) at Telecom Paris (previously named ENST) in Paris (France). She was the head of the research group CCN "Cybersecurity for Communication and Networking" from 2015 to 2018. In 2005, she obtained her HDR (Habilitation à diriger les recherches). Prior to this, she held a research position at Eurecom Institute, Sophia-antipolis, France. Her current research interests include security in cooperative ITS, cooperation in wireless and autonomous networks (WLANs/MANETs/WSN/Mesh/vehicular/cellular), QoS, performance evaluation and link adaptation mechanisms. She published six books and many research papers in these areas. She published six books, 10 patents, 4 drafts IETF and more than 250 research papers in these areas. She is a Founder of IFIP NTMS Conference, on New Technologies, Mobility and Security (NTMS2007). She served as an Associate Editor and a member on the Editorial Board for several journals. She is currently involved in major national and European projects focusing on security for connected and autonomous vehicles (SCOOP@F, InterCor and C-roads). She is a co-leader of the Chaire C3S on Cybersecurity for connected and autonomous vehicles with French partners Renault, Valéo, Thalès, Nokia and Wavestone.

Lusheng Wang received the B.Sc. degree in communications engineering from the Beijing University of Posts and Telecommunications (BUPT), China, in 2004, and the Ph.D. degree in computer science and networks from Telecom ParisTech (ENST), France, in 2010. He worked as a Postdoctoral with INSA-Lyon, during 2010, and a Postdoctoral with Eurecom, France, from 2011 to 2012. He is currently a Research Professor and the Vice-Dean of Communications Engineering Department, Hefei University of Technology (HFUT), China. His research interests include resource and interference management in hyper-dense and heterogeneous networks. He has published about 50 refereed international journals and conference papers.