Journal of Cloud Computing:
Advances, Systems and Applications

REVIEW

Open Access

Task partitioning and offloading in IoT cloud-edge collaborative computing framework: a survey

Haiming Chen*, Wei Qin and Lei Wang

Abstract

Internet of Things (IoT) is made up with growing number of facilities, which are digitalized to have sensing, networking and computing capabilities. Traditionally, the large volume of data generated by the IoT devices are processed in a centralized cloud computing model. However, it is no longer able to meet the computational demands of large-scale and geographically distributed IoT devices for executing tasks of high performance, low latency, and low energy consumption. Therefore, edge computing has emerged as a complement of cloud computing. To improve system performance, it is necessary to partition and offload some tasks generated by local devices to the remote cloud or edge nodes. However, most of the current research work focuses on designing efficient offloading strategies and service orchestration. Little attention has been paid to the problem of jointly optimizing task partitioning and offloading for different application types. In this paper, we make a comprehensive overview on the existing task partitioning and offloading frameworks, focusing on the input and core of decision engine of the framework for task partitioning and offloading. We also propose comprehensive taxonomy metrics for comparing task partitioning and offloading approaches in the IoT cloud-edge collaborative computing framework. Finally, we discuss the problems and challenges that may be encountered in the future.

Keywords: IoT, Cloud-edge collaborative computing, Task partitioning, Offloading

Introduction

In the past few decades, the Internet has expanded to the physical world and grown to the Internet of Things (IoT). Ericsson predicts that more than 50 billion devices will be connected to the Internet by 2025. Most of these devices are located at the end of the Internet, affecting all aspects of our daily life and traditional industry. The Internet of Everything (IoE) has become a major trend in the development of the Internet and wireless communication networks (Patel et al. [1]). According to the reports, global monthly mobile data traffic will reach 49 EB by 2021. Due to the limited computing resources of end devices (e.g., disk capacity, memory size, and operating speed), it is impractical to perform tasks on the end device locally, especially some resource-intensive applications. Under the circumstance, cloud computing grows fast, because it can provide flexible services and data-intensive processing capabilities to end-users through wide area networks (WANs). Its representatives are Amazon Web services, Microsoft Azure and Google Cloud, from which users can obtain seemingly unlimited resources without building new computing infrastructure. However, some of the IoE applications, such as face recognition, ultra-high-definition video, augmented reality (AR), virtual reality (VR), and semantic analysis of speech, which require low latency and high computing performance, challenge the scalable and flexible computing models of the traditional clouds.

To solve this problem, mobile edge computing (MEC) has been proposed (Hu et al. [2]). The core idea of MEC

*Correspondence: chenhaiming@nbu.edu.cn

Faculty of Electrical Engineering and Computer Science, Ningbo University, Ningbo, China

is to place servers at the edge of the network so that computation, bandwidth, and storage resources are provided in places close to the IoT end devices to reduce the service latency and energy consumption of end devices (Abbas et al. [3]; Shi et al. [4]), and also to improve the quality of user experience (QoE) and quality of service (QoS) (Lai et al. [5]; Sodhro et al. [6]). Edge servers can be either temporary devices, such as laptops, micro-servers, etc., or nearby infrastructure. Compared with large-scale server clusters such as cloud data centers, the computing and storage capacity of edge servers is relatively lower, but it has the advantages of short access distance and flexible geographical distribution. However, edge computing also brings complex resource management problems, since the end devices on the access side are featured by fast-changing service demands, high mobility and large data volume, etc. Therefore, it is necessary to design an efficient cloud-edge collaborative computing mechanism through task scheduling and coordinating management of computing resources, so as to provide better services for users. The architecture of cloud-edge collaborative computing system is shown in Fig. 1.

It is a crucial problem to efficiently offload computing tasks for realizing cloud-edge collaborative computing, in terms of response time, energy consumption of end devices, and quality of experience for users. Depending on the type of application tasks, the computing offloading can be generally divided into two categories, namely full offloading (Ma et al. [7]) and partial offloading (Kuang et al. [8]). Full offloading is mainly focused on indivisible applications, which has been studied extensively, while the partial offloading is targeted at divisible applications. For the partial offloading, it handles a complete application by dividing them into locally executed subtasks and offloadable subtasks. The locally executed subtasks are referred to as those that must be executed on the local device, such as tasks calling peripheral device interfaces or requiring user interactions, etc. The offloadable subtasks are referred to as those that do not require interaction with the local device, but tend to do computationally intensive data processing and can be offloaded to run on edge servers or remote clouds.

The problem of computing offloading has been extensively studied so far, mainly considering the challenges posed by the geographically distributed edge servers and the heterogeneity of various resources. Recently, Avgeris et al. [9] proposed an optimal resource allocation framework by introducing a Markov Random Field based mechanism for distributing the excess workload in scenarios with heterogeneous edge devices and dynamic wireless conditions modeled by the dynamic behavior and mobility of the users. There are also some reviews providing analytical overview of these research works from different perspectives. Heidari et al. [10] presented a study on task offloading for IoT devices in various computing scenarios from the following three aspects: computing environment, offloading strategy, and decision process. Zhang et al. [11] conducted a comprehensive study on four critical issues of task offloading in MEC, including architecture, computation migration,

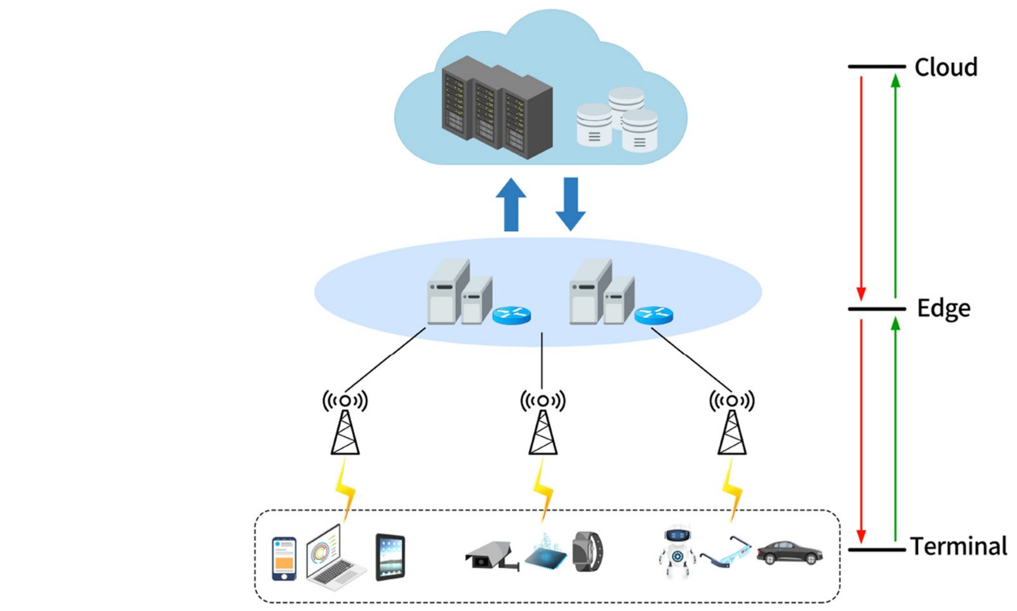

Fig. 1 Architecture of cloud-edge collaborative computing system

edge caching, and service orchestration. Saeik et al. [12] gave a survey of how the Edge and/or Cloud can be combined together to facilitate the task offloading problem, while emphasizing on the mathematical, artificial intelligence and control theory optimization approaches that can be used to satisfy the various objectives, constraints and dynamic conditions for different applications. Feng et al. [13] presented approaches to achieve offloading objectives mainly including mathematical solver, heuristic algorithms, Lyapunov optimization, game theory, and Markov Decision Process (MDP) and Reinforcement Learning (RL), and they compared the approaches by characterizing their pros and cons as well as targeting applications. We summarize the existing survey papers of tasking offloading in Table 1.

Briefly speaking, we found that most of these reviews focus on various offloading strategies and supporting techniques to alleviate the burden on smart devices, whereas few of them consider task partitioning together with computing offloading. Therefore, in this paper we make a survey on task offloading in IoT cloud-edge collaborative computing from a new perspective. The contributions of the paper are embodied in the following aspects.

1) We pay attention to the problem of jointly optimizing task partitioning and offloading for different IoT cloud-edge collaborative computing scenarios;
2) We analyze some existing task partitioning and offloading frameworks and extract a general workflow of task partitioning and offloading, which mainly includes an application model for profiling users' requirements of task partitioning and offloading and a decision engine for solving application models;
3) We propose comprehensive taxonomy metrics for comparing task partitioning and offloading approaches in the IoT cloud-edge collaborative computing framework.

The rest of the paper is organized as follows. Task partitioning and offloading framework section introduces several task partitioning and offloading frameworks and provides a comparative analysis of them, to extract a general workflow for task partitioning and offloading. Input of decision engine: application model section introduces the application modeling approach and focuses on interdependent subtask modeling and cost modeling. Core of decision engine: model solving section analyzes the currently proposed approaches for solving models of task partitioning and offloading decisions. Taxonomy of task partitioning and offloading approaches section proposes a comprehensive taxonomy for task partitioning and offloading approaches. Conclusion and future work section points out some future challenges and perspectives.

Table 1 Some existing surveys on computing offloading in edge computing

Survey paper	Summary
Heidari et al. [10]	The authors outlined task offloading for different computing environments, with offloading strategies and decision processes.
Zhang et al. [11]	The authors focused on architecture, computation migration, edge caching, and service orchestration in task offloading.
Saeik et al. [12]	The authors emphasized on solving the task offloading problem with the mathematical, artificial intelligence and control theory optimization approaches.
Feng et al. [13]	The authors compared the approaches including mathematical solver, heuristic algorithms, Lyapunov optimization, game theory, and MDP/RL for solving the task offloading problem.
Mach et al. [14]	The authors focused mainly on computation offloading decisions, by considering allocation of computing resources and mobility management in MEC.
Jiang et al. [15]	The authors classified the existing work on task offloading into two categories, namely gaming and cooperation between edge and the cloud, and heuristic algorithms.
Wang et al. [16]	The authors classified the existing work on task offloading into five categories, based on offloading destination, load balance of edge servers, device mobility, application partitioning, and partition granularity, respectively.
Lin et al. [17]	The authors classified the computation offloading from offloading flow and offloading scenario. They also classified the computation offloading schemes into five categories, which are (non)convex optimization, MDP, game theory, Lyapunov optimization, and machine learning.
Shakarami et al. [18]	The authors only reviewed the computation offloading approaches based on game-theoretic for edge computing.
Wang et al. [19]	The authors reviewed the task offloading schemes considering response time, device energy, service provider cost, load balance between edge and cloud, and device mobility.
This paper	We consider task partitioning together with computing offloading, and propose a general framework including an application model and a decision engine, and more comprehensive taxonomy metrics for classifying task partitioning and offloading approaches.

Task partitioning and offloading framework

MAUI

MAUI is an offloading framework proposed by Cuervo et al. [20] to support fine-grained energy-aware application code offloading to the edge servers with hosted code running environments. The system architecture of MAUI is shown in Fig. 2. The MAUI runs on the smartphone, and its runtime environment consists of three components: 1) profiler, which detects programs and collects the energy and data transfer requirements of the programs; 2) client proxy, which transfers the control and data messages to finish code offloading; and 3) solver, which coordinates with the MAUI server to simplify the offloading decision to save energy. On the server side, MAUI has four components, which are profiler, server agent, solver, and MAUI controller. The MAUI controller is responsible for identifying and allocating resources of the incoming requests to instantiate the partitioned application. MAUI provides method-level code offloading based on the .NET framework.

Lots of early attempts have been taken to make the code migration method light-weighted. Most of the proposed methods are relied on the programmer to manually modify the program to handle task partitioning, code migration, and make it adapted to network condition changing. These methods are fine-grained and able to save energy consumption of mobile devices, but at the same time place a considerable burden on programmers. MAUI reduces the burden on programmers by combining code portability, reflection, serialization, and type safety. MAUI uses the Microsoft .NET Common Language Runtime (CLR) [21] to handle computing tasks on both the smartphone and the server sides. Hosted code enables MAUI to ignore the differences in instruction set architecture for both mobile devices (typically using ARM-based CPUs) and servers (typically using x86 CPUs). MAUI takes costs of network (i.e., latency, bandwidth, and wait time) and CPU into consideration to provide an efficient solution for determining how to partition applications at running time. Therefore, MAUI has merit in reducing the energy consumption of devices, but it does not consider how to overcome issues such as memory resource limitations of devices.

CloneCloud

CloneCloud is a flexible partition analyzer and application runtime proposed by Chun et al. [22]. It combines static and dynamic analysis to automatically partition applications at a fine-grained level, so as to minimize execution time and energy consumption for the targeted computing and communication environments. The system architecture of CloneCloud is shown in Fig. 3.

The partitioning mechanism in CloneCloud is offline, aiming to determine which parts of the application run on the mobile device and which parts of codes are to be migrated to the cloud. Unlike the previous method (including the MAUI system mentioned above), programmers do not need to write applications in a prescribed manner or annotate source code manually. CloneCloud uses static analysis to determine the parts of migration and the reintegration points in the code, based on a set of constraints. Dynamic analysis performs the

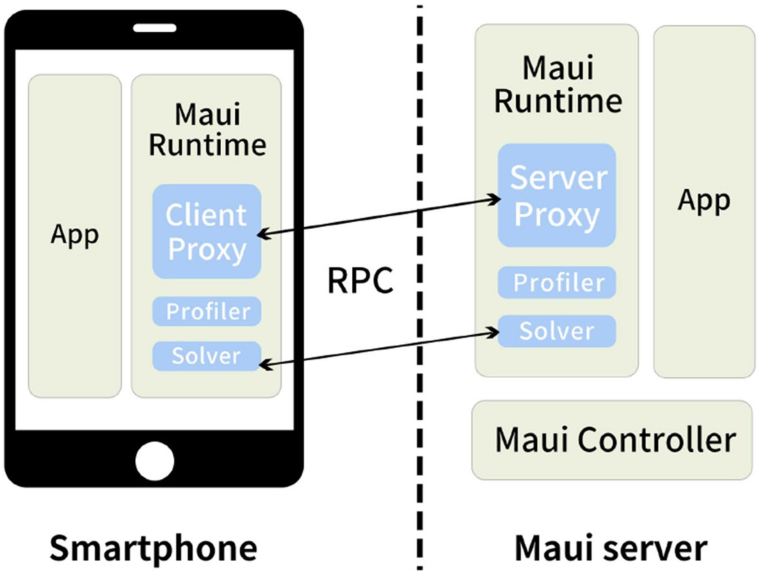

Fig. 2 Architecture of MAUI. (Cuervo et al. [20])

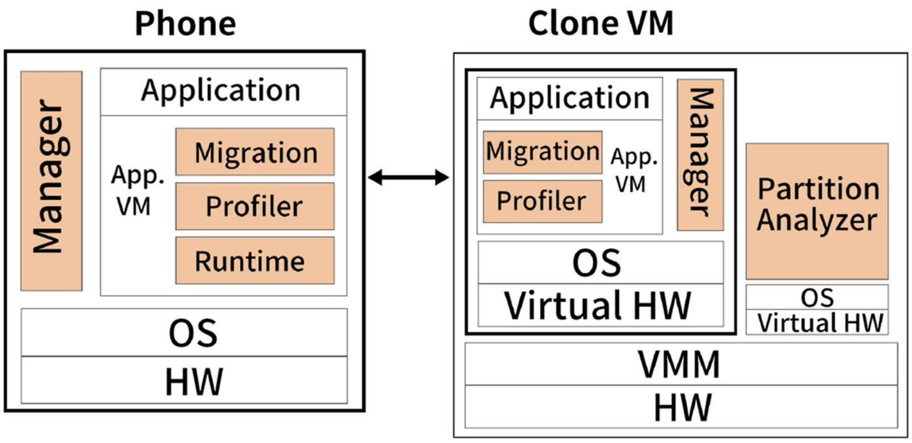

Fig. 3 Architecture of CloneCloud. (Chun et al. [22])

cost model to build the adaptation environment for applications of different partitioning strategies. Finally, the optimization solver uses the cost model derived from the dynamic analysis to make a proper partitioning decision.

CloneCloud does not rely on programmers' annotations in source codes. Moreover, CloneCloud allows native operating systems to execute in a virtual machine in the cloud. Differing from the application-layer virtual machine (VM) migrators (Aridor et al. [23]; Zhu et al. [24]), CloneCloud can utilize both raw CPU functionality and system functionality to implement the underlying libraries and operating systems. Unlike the traditional interruption recovery mechanism used for application migration, CloneCloud migrator operates on a thread-level granularity.

ThinkAir

ThinkAir is a mobile cloud computing (MCC) offloading framework proposed by Kosta et al. [25]. Based on the technology of smartphone virtualization in cloud computing, ThinkAir provides a method-level approach of task offloading through source annotation. It not only considers on-demand resource allocation during task offloading, but also provides a VM manager and an automated parallel processing module in the cloud to manage VMs for end devices. Particularly, ThinkAir enhances the computing power of mobile devices by dynamically creating, restoring, and destroying VM images of the entire smartphone system in the cloud. This enables parallelized applications to call multiple VMs to execute in the cloud in a seamless and on-demand manner, thereby greatly reducing execution time and energy consumption. As shown in Fig. 4, the ThinkAir system architecture consists of three main components, which are execution environment, application server, and profiler.

The execution environment includes programmer API, compiler, and execution controller. ThinkAir provides a simple library, together with the support of the compiler. The programmer only needs to annotate which method to be considered for offloading with '@ Remote'. This essentially relieves the burden on programmers. The compiler is composed of the remote code generator and the customized native development kit (NDK). The remote code generator is a tool that converts annotated code. The NDK provides native code support on the cloud. The execution controller drives offloading decisions and records past-calling data.

The application server manages the code offloaded to the cloud, consisting of three main components: the client processor, the cloud infrastructure, and the automatic parallelization component. The client processor is responsible for building a connection between the client (mobile device) and the cloud server, receiving and executing the offloaded code, and returning the results. To make the cloud infrastructure easy to maintain and to keep the execution environment consistent, ThinkAir uses a virtualized environment that allows the system to be deployed where needed.

The profiler is the most crucial part of the ThinkAir framework, which consists of three sub-profilers (i.e. hardware profiler, software profiler, and network profiler). They collect different data and input it into the energy model. To solve the problem that specific components (e.g., GPS and audio, etc.) must run locally and cannot be migrated to the cloud, ThinkAir designs and implements an energy estimation model based on the PowerTutor model (Zhang et al. [26]).

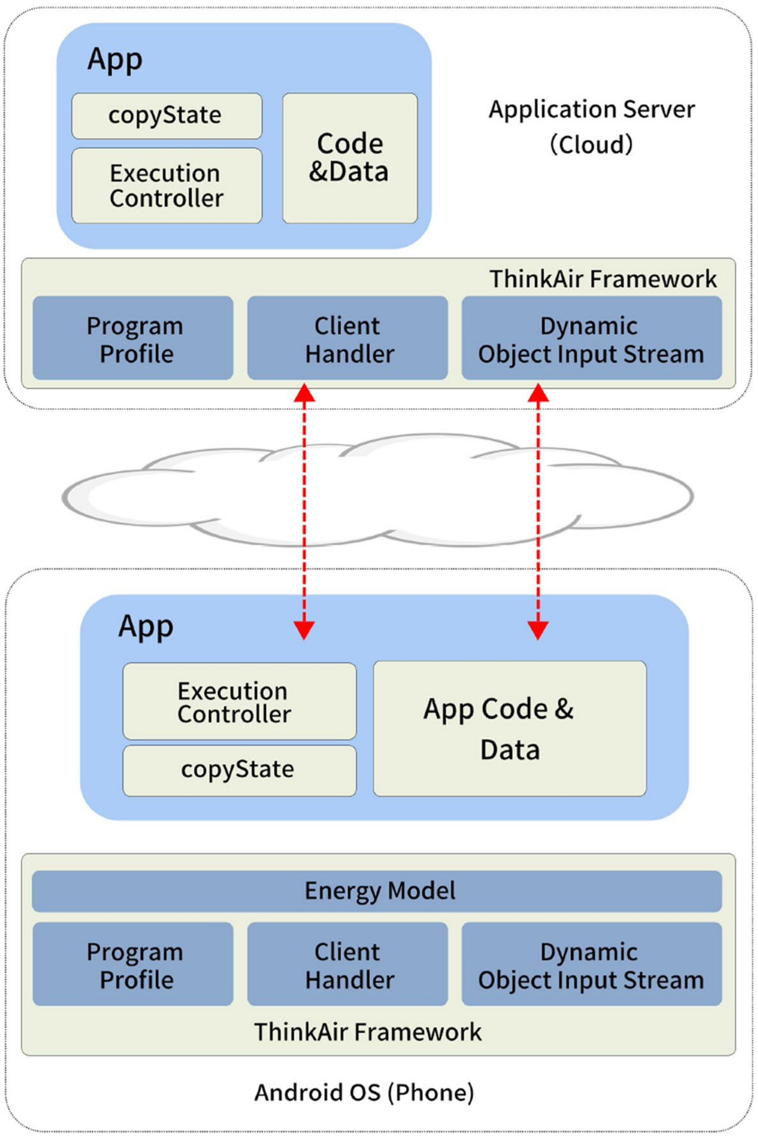

Fig. 4 Architecture of ThinkAir. (Kosta et al. [25])

Phone2Cloud

Phone2Cloud is an energy-saving mobile cloud computing system with semi-automatic offloading proposed by Feng et al. [27]. To improve the energy efficiency of smartphones, Phone2Cloud offloads the computing tasks of applications running on smartphones to the cloud while improving the performance of the applications by reducing the execution time, which ultimately improve the quality of user experience. The system architecture of Phone2Cloud is shown in Fig. 5.

Phone2Cloud consists of seven components, including bandwidth monitor, resource monitor, execution time predictor, offloading decision engine, local execution manager, remote execution manager, and offloading proxy. Among these components, the offloading decision engine is its core and can be easily added or removed from the framework. Essentially, it was implemented based on the work of Liu et al. [28] and Xian et al. [29], which takes the user's delay tolerance threshold as a constraint (Balasubramanian et al. [30]). The bandwidth monitor and resource monitor are used to monitor the current bandwidth of the network and the CPU load on the smartphone, respectively, as well as to predict the average bandwidth and average CPU load when making offloading decisions. The execution time predictor is used to predict the average execution time of an entire

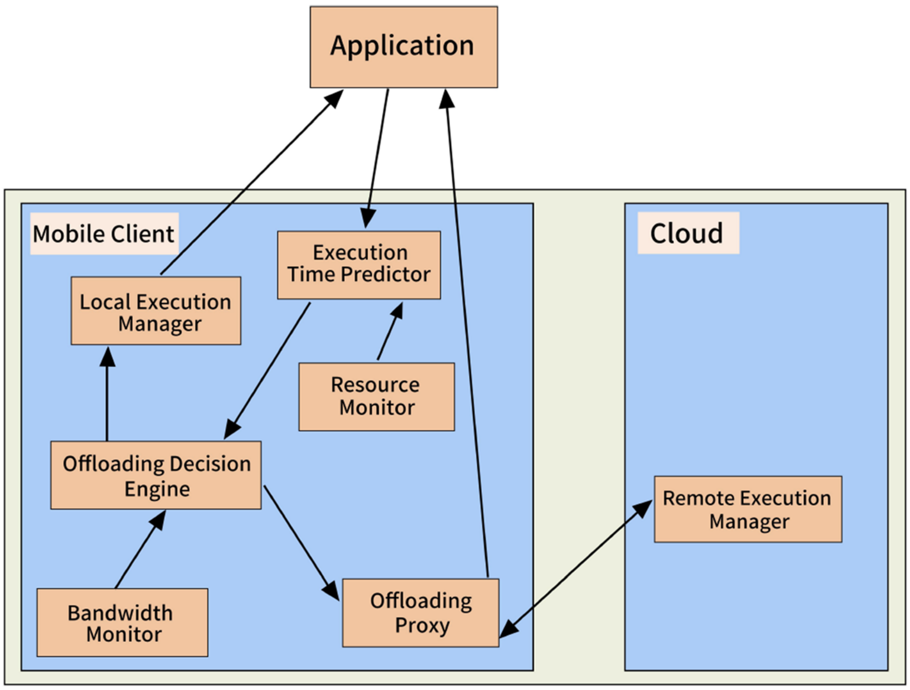

Fig. 5 Architecture of Phone2Cloud. (Feng et al. [27])

application on a smartphone. Local and remote execution managers are mainly used to manage the execution of applications in Phone2Cloud. The offloading proxy connects the offloading decision engine and the remote execution manager together. Thus, Phone2Cloud mostly solves the offloading problem at the task-level granularity.

ParGen
ParGen is a parallel approach for module-level partitioning and offloading of dataflow applications proposed by Wen et al. [31]. ParGen aims to optimize the throughput of dataflow applications, considering that the edge cloud always serves multiple users in the vicinity and has limited resources. A novel model was defined for resource allocation and application partitioning in multi-user scenarios. A divide-and-conquer parallel approach for partitioning dataflow applications in a multi-user scenario with competing resources was proposed to jointly optimize application partitioning and resource allocation among users, so as to maximize the average throughput of all users. The system architecture of ParGen is shown in Fig. 6.

ParGen consists of three components: mobile devices, edge cloud, and cloud. The edge cloud is deployed closer to users than cloud servers, which can effectively reduce latency. There are a limited number of servers on the edge cloud capable of running modules of mobile applications simultaneously. Many mobile devices connect to the edge cloud through a base station (BS) or a wireless access point (AP). On the mobile device side, there is a client middleware between the application layer and the operating system (OS). Its internal monitoring agent program can send execution and transmission costs to the partitioning program in edge and cloud to request computation partitioning and bandwidth allocation. After receiving the request, the partitioning program determines the best partitioning policy for each user and allocates bandwidth. The remote cloud provides APP store services and programming support for mobile applications.

Summary
From the frameworks for task partitioning and offloading discussed above, we can see that partition applications at different granularities can alleviate the resource constraints in mobile devices. In general, the necessary components to implement task partitioning and offloading are: (1) a profiler that detects the hardware conditions of the application, the mobile device itself, and the state of the network connection, and collects the energy and data transfer requirements for task execution; (2) a decision engine that makes decisions about whether and how to offload based on the results of the partitioning and offloading model; and (3) an offloading agent that sends the required input data to a remote server, receives the

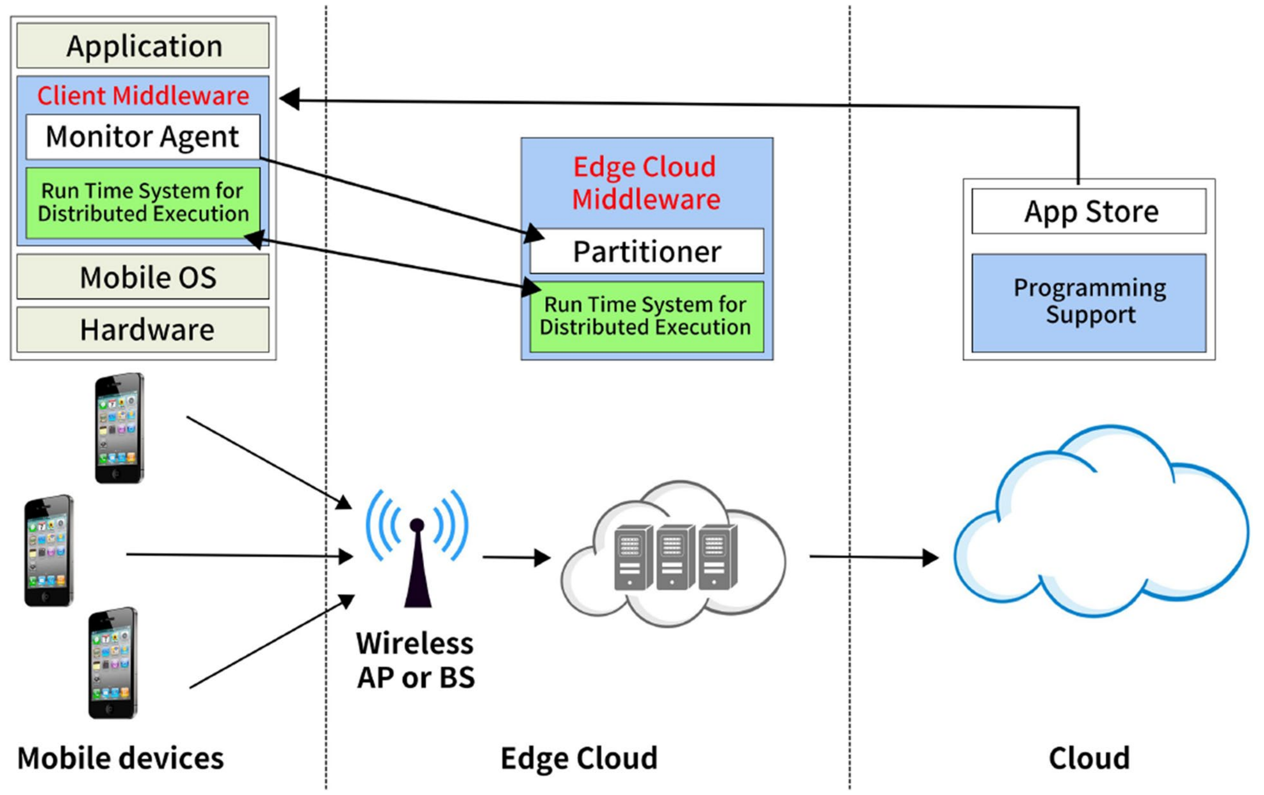

Fig. 6 Architecture of ParGen. (Wen et al. [31])

results from the remote server, and passes the results to the application.

We abstract the workflow of the decision engine as shown in Fig. 7. Briefly speaking, firstly the dependence relation among the modules of a given application task is modelled as an application model, and its partitioning and offloading optimization goal is modelled as a cost model based on actual requirements. Secondly, based on the application model, an appropriate algorithm should be proposed to get the appropriate task partitioning and offloading result. Finally, the result is taken as input of the offloading agent to offload some modules executed on remote servers, while the rest executed locally on the mobile device. In this paper, we focus on making a survey on the input and core of decision engine of the framework for task partitioning and offloading.

Input of decision engine: application model

Generally, a task for IoT applications is composed of several inter-dependent subtasks. The dependencies are determined by the context awareness and hardware requirements of the components to support proper

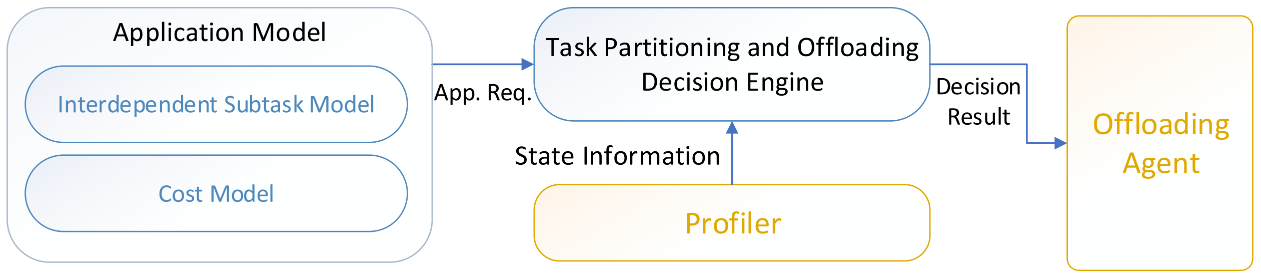

Fig. 7 The general framework and workflow of task partitioning and offloading

operation of the applications. To make offloading decisions on decision engine, the inter-dependent subtask model should be partitioned into multiple sub-graphs of disjoint modules, which are then offloaded to appropriate servers based on cost model. In this section, we will have an overview on these two models comprising application models, which are used as input of decision engine.

Inter-dependent subtask model

The first and essential step in modelling the application is to model inter-dependent subtasks, called inter-dependence subtask models. At present, Directed Acyclic Graph (DAG), which uses the vertices to represent sub-tasks and edges to represent relation, is commonly used to model dependence relation of subtasks.

For a task to be run in IoT systems, the dependence relation among subtasks such as execution order, data flow, and control flow at different granularity levels can be obtained statically or dynamically based on task parameters or context-awareness requirements. For statically modelling, the dependence relation is depicted before tasks are actually executed. For dynamically modelling, the dependence relation is determined during task running under control of the decision engine of the task partitioning and offloading framework. No matter which type of modelling, the dependence relation is mostly modelled as DAG. Depending on the computational scale of tasks, dependence relation models can have a large number of vertices with interdependencies of high complexity compared to simple class diagrams.

Niu et al. [32] constructed weighted object relationship graphs (WORGs) for mobile applications. The object and object-to-object dependence were obtained by analyzing the bytecode of tasks, and the object relationship graph (ORG) was constructed by traversing the application based on method calls, as shown in Fig. 8(a), where nodes represent objects and edges represent dependence between objects. Then, the execution time of each object and the transmission data of each method call were calculated as the weights of nodes and edges, respectively. The resulting weighted object relationship graphs (WORGs) were constructed as shown in Fig. 8(b).

Cost model

Different types of applications, such as data-oriented applications, code-oriented applications, and continuous execution applications, etc. (Munoz et al. [33]), have different requirements for offloading computing tasks to remote servers. The application is modelled by taking into consideration the requirements of users, network environment, and service providers, finding the corresponding optimization goals and constraints to establish a cost model. Several cost factors are usually considered in cost modelling, including latency, energy consumption, quality of service, quality of experience and economic cost.

Latency

Latency includes latencies in computing, transmission, and queuing. It is one of the essential metrics for evaluating the performance of mobile devices. This is especially true for latency-sensitive applications, like emergency event processing in industrial internet of things. The servers in traditional cloud computing centers have high computing power to handle complex workloads (e.g., image processing (Kumar et al. [34]), speech recognition

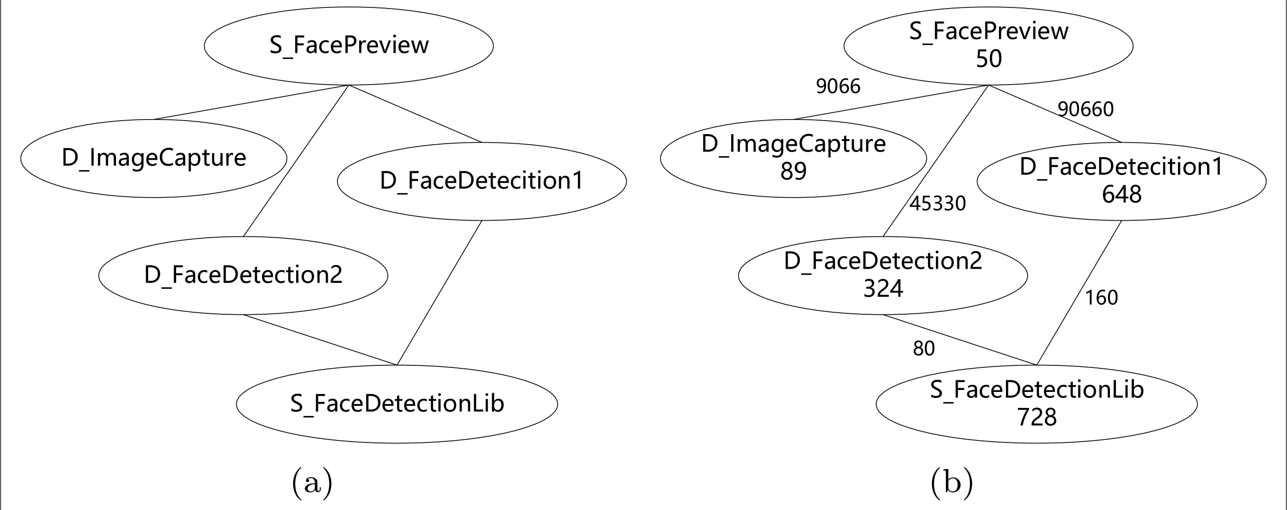

Fig. 8 **a** Object Relationship Graph (ORG). **b** Weighted Object Relationship Graphs (WORGs). (Niu et al. [32])

(Muhammad et al. [35]), etc.). However, sending all subtasks to clouds can make both computing resource and network bandwidth congested, leading to higher queuing latency and then total latency. To reduce the total latency, the best approach is to offload workloads that do not need to be executed locally on mobile devices to the edge servers with sufficient computing power, which are closer to the local device than the remote cloud.

Theoretically, if all the subtasks can be processed at edge servers, the latency can be significantly reduced compared to sending them to cloud servers for execution (Shi et al. [36]), because the transmission time between the edge and the cloud is saved. However, a task arriving at a resource-constrained edge node is not likely to be executed immediately, and there is often a waiting time for execution. Therefore, the geographically closest infrastructure is not always a good choice. By taking a queuing model based on the priority of tasks in calculating the delay, it is a practical problem to dispatch subtasks to the proper layers (i.e. local, edge and cloud), allocate bandwidth for mobile end devices, and find a suitable transmission rate. To solve the problem, the real-time network state (e.g., transmission bandwidth, etc.), computing power and bandwidth usage information of edge and cloud layers must be considered to avoid unnecessary contention and delays.

Energy

Lifetime of mobile devices is considered as one of the indispensable factors affecting the working periods of IoT applications. Although some progress has been made in battery-related technology research, there is still a gap between the rapid growth of power consumption and the battery capacity of current mobile devices. Therefore, another critical optimization goal of partition and offload decision-making is how to extend the battery life of mobile devices by reducing energy consumption. There have been many works that discussed the energy consumption optimization problem from different perspectives. For example, Zhao et al. [37] proposed an approximate algorithm of computation offloading, which was shown saving nearly 82.7% of energy compared to executing it entirely on a mobile device.

Many related works have developed power/energy models for various components on mobile terminal devices, including central processing units (CPUs) as well as cellular and WiFi communications. Recently most of the researchers used specific hardware (external devices or components that attached with the mobile device) to obtain working voltage and current of devices, and then the power models can be obtained. However, this measurement based approach can only obtain power model of the whole device, but not that of each hardware component of the device (e.g., CPU, GPU, network interface, etc.). This limitation causes challenges in analyzing the energy consumption of different types of applications on mobile devices.

Quality of Service (QoS)

Due to the increasing popularity of video streaming services (e.g., YouTube and Netflix, etc.) and the exponential growth of active users, more research works use QoS as a modelling and optimization goal of many partitioning and offloading approaches (Rausch et al. [38]; Song et al. [39]). However, it is challenging to achieve expected QoS due to many factors, such as different client devices/request patterns, changing media content, and network environments. Generally speaking, the following factors should be taken into consideration to optimize QoS of network systems. First, the variability of network resources, the unstable nature of wireless channels, and the characteristics of fixed/mobile networks in heterogeneous environments. Second, the emergence of new services (e.g., video games and virtual/augmented reality (VR/AR)), the diversity of usage environments, users' expectations, and the optimization of operational costs for mobile and service providers. Third, the resource constraints of edge servers, various measurement and evaluation methods for QoS management when allocating resources among users with other users' quality of experience (QoE) preferences.

QoS is sometimes directly related to the response time of an application (amount to the latency metric), which can be taken as a constraint to achieve a specific QoS. Particularly, QoS improvement just requires keeping the response time within a threshold, rather than minimizing that time. Aazam et al. [40] proposed a resource estimation approach based on the QoE history of the cloud service customer (CSC) while enhancing the QoS. Mahmud et al. [41] proposed a delay-aware application module management policy that considers various aspects of distributed applications in a decentralized and coordinated environment inducing latency, so as to ensure QoS while meeting deadlines for all types of applications and maximizing utility of fog computing resources.

Quality of Experience (QoE)

With the proliferation of IoT and the ensuing computing-intensive tasks, the surging need for computing offloading, and how to optimize offloading for the optimal user quality of experience (QoE) is a fundamental question. Despite that it is important to consider the characteristics of the task and make optimal offloading decisions based on the QoE requirements of each end-user. Particularly, different types of end-users pursue different QoE performance. For example, cloud gaming inclines to have lower execution time during computational offloading, while the unmanned aerial vehicles (UAVs) prefer computationally offloading with low energy consumption to prolong battery life. On the basis of comprehensively considering the different QoE performance requirements of IoT users for task execution time, task processing energy consumption and computing cost, Luo et al. [42] proposed a QoE-driven adaptive computing offloading (QEACO) strategy based on theoretical performance analysis. Since then, each IoT user can optimize and adapt to their best QoE to make offloading decisions.

In the business scenario of IoT, in addition to providing low latency and low energy consumption, the cloud-edge collaborative computing module also needs to provide suitable caching, quick communication and vast amounts of computing power (Huang et al. [43]). QoE can be regarded as the most direct experience in service interaction, especially in Internet of Vehicles (IoV). He et al. [44] studied the problem of QoE-based edge task offloading in IoV, and proposed an improved deep reinforcement learning (DRL) algorithm named PS-DDPG, in which a QoE model was designed by taking into account the limited vehicle cache and the unpredictable communication path caused by the diversification of transmission information.

Economic cost

The economic cost is an essential factor when selecting computing resources for device users and when service providers offer solutions. The reasonable pricing of services and the optimization of economic cost are essential issues to be addressed in task partition and offloading. From the perspective of service providers such as YouTube and Amazon, cloud-side collaborative computing provides lower latency and energy consumption, potentially increasing throughput and improving user experience (Shi et al. [36]). As a result, they can earn more benefits by handling the same volume of computing or storage. The service provider's investment is mainly spent in building and maintaining each tier of resources. To fully utilize the resources of each layer, the provider can charge the users based on the data location and expected resource utility. Therefore, how to build a reasonable economic cost model based on the characteristics of cloud-edge collaborative computing to ensure the profitability of service providers and the acceptability of users is the focus of current research work.

Integrated model

Due to diverse needs of users and service providers in fact, the joint optimization of multiple costs needs to be considered in task partitioning and offloading solutions. However, multiple costs are usually in conflict with each other, and it is difficult to make them optimal simultaneously. For example, to reduce energy consumption of the whole system (including mobile devices, edge servers and cloud servers), it may induce increasing response time of applications. A compromise between these costs is needed to obtain optimal partitioning and offloading decisions. In general, there are several ways to build a cost model for joint optimization of multiple costs: 1) Converting multiple costs into one cost. For example, a square-weighted addition approach assigns each cost its weight to approximate the optimal solution as closely as possible; 2) Selecting the most important cost as the optimization objective and the rest as constraints; 3) Obtaining the Pareto-optimal solutions (Lin et al. [45]) of each cost for the mobile device users or service providers according to actual situations.

Let the first approach be used as an example for cost modelling. Assume that the application had n modules, for a given module i, the size of data transmission was t_i, the memory cost was m_i, and the code size was c_i. Also, for each module, a variable $x_i \in \{0,1\}$ was introduced to indicate whether module i was executed locally ($x_i = 0$) or remotely ($x_i = 1$). The cost model was thus expressed as follows:

$$Min\ (w_t \cdot C_t + w_m \cdot C_m + w_c \cdot C_c) \tag{1}$$

$$\text{s.t. } C_t = \sum_{i=1}^{n} c_i \cdot x_i + \sum_{i=1}^{n}\sum_{j=1}^{n} t_j \cdot (x_j \oplus x_i) \tag{2}$$

$$C_m = \sum_{i=1}^{n} m_i \cdot (1 - x_i) \tag{3}$$

$$C_c = \sum_{i=1}^{n} \alpha \cdot c_i \cdot (1 - x_i) \tag{4}$$

In Eq. (4), α is the conversion factor that maps the code size to CPU instructions. As shown in Eq. (1), the cost model combines three costs, namely transmission cost C_t, local device memory cost C_m, and local device CPU occupation cost C_c, into one, where w_t, w_m, w_c are the weights of each component that determine priorities of each optimization goal.

Let the second approach be used as an example for cost modeling. Assuming that the goal is to minimize latency and energy consumption of local devices, let E^{tot} denote the total energy consumption of mobile devices, while E^{loc} and E^{com} denote the energy consumption of local execution and the energy consumption of migrating data between local devices and edge servers respectively. Thus, the total consumed energy E^{tot} can be expressed as:

$$E^{tot} = E^{loc} + E^{com} \tag{5}$$

Then assume that T^{tot} denotes the execution time of application from start to end, T^{loc}, T^{edge} and T^{com} denote the time needed for a part of the task to be executed locally, the time needed for a part of the task to be executed at the edge node and the communication time needed for data migration between the local device and the edge server respectively. So T^{tot} can be expressed as:

$$T^{tot} = T^{loc} + T^{edge} + T^{com} \tag{6}$$

Next, we choose energy consumption minimization as the final optimization objective while limiting the total execution time:

$$\begin{aligned} &Min\ E^{tot} \\ \text{s.t. }\ &T^{tot} < D, \end{aligned} \tag{7}$$

where D denotes the deadline to complete execution of the application. The values of E^{loc} and T^{loc} depend on the workload of the application running locally and the computing capability of the local device. The values of E^{com} and T^{com} are mainly affected by the variation of network bandwidth. T^{edge} is determined by the workload of the application offloaded to the edge and the computing capability of the edge servers.

Core of decision engine: model solving

Taking the application mode as input, the decision engine is responsible for decomposing the inter-dependent subtask models to determine at the specified granularity when and where to place which parts of tasks on the remote servers (i.e. edge and cloud servers). In particular, the decision engine first checks whether the application requires annotations. If annotations are needed, it will look for pre-annotated components for partitioning the application. If annotations are not necessary, it proceeds to check if the event profiler functionality is implemented in the application. If the event profiler is available, it collects information to optimize the application partitioning and task offloading decision. The decision process is usually modelled as a mathematically constrained optimization problem.

The core of decision engine is the solver of the optimization problem based on the graph model of inter-dependent subtask model and cost model. The optimization algorithms can exhibit different characteristics when targeting different scenarios and optimization metrics. For example, some algorithms produce low overhead in the analysis technique, while others generate unnecessary overhead. Some algorithms need to assign decisions online, while others need offline pre-processing. The performance of the resulting optimization is improved by extracting the algorithmic advantages of both types of models. At present, the methods of modelling and solving the optimization problem can be divided into three categories, which are graph optimization, traditional programming, and Markov Decision Process (MDP) and Reinforcement Learning (RL).

Graph optimization

Giurgiu et al. [46] took profile information such as characterization of modular architecture, CPU load, and network state in the cloud and locally to generate a compact specification of the application and environment in the form of resource consumption graph. This graph was used together with the modular and highly cohesive subtask model by the optimization solver (i.e. the core of decision engine) to determine the optimal allocation of modules and deploy them accordingly through traversing the graph in an adaptive topological order.

Niu et al. [32] used weighted object relationship graphs (WORGs) as application model, and modelled the task partitioning and offloading problem as a graphical problem to minimize execution time and energy cost. The problem was solved by the Branch-Bounded Application Partitioning (BBAP) algorithm and the Minimal Cut Set Greedy Application Partitioning (MCGAP). In the proposed approaches, bandwidth was considered as an essential operational variable.

Lakhan et al. [47] proposed an adaptive content-aware task scheduling (CATSA) framework for mobile workflow applications. To satisfy all mobile workflow application constraints while reducing their execution time, CATSA first represented inter-dependent tasks with a directed acyclic graph, and then sorted them in topological order based on the earliest due first (EDF), shortest processing first (SPF), and shortest spare time first (SSTF) approach. Thereafter, the optimal task offloading decision was made by selecting an optimal solution from the solution set using a random search approach.

Wu et al. [48] represented general tasks in arbitrary topological consumption graphs, and proposed a Min-Cost Offloading Partitioning (MCOP) algorithm to determine which portions of the application must run on the mobile device and which portions on cloud/edge servers under different cost models and mobile environments.

Traditional programming

Linear programming

Traditionally, the problem of deciding when and where to place which parts of tasks on remote servers can be modelled as an integer linear programming (ILP) model. For an ILP model, it has linear optimization objective function, which is bounded by linear equations or inequations, and all of its variables are restricted to integers. The annotations and analysis results given by the programmer about whether the application is partitionable or not are essential for the integer programming (IP) optimization problems. Linear programming (LP) is a traditional mathematical planning method valid for solving the problems. The advantage of ILP model is that it always produces the best results for a particular objective function, but solving linear programming problems usually requires a lot of computing time.

MAUI (Cuervo et al. [20]) provided a fine-grained task partitioning and code offloading decision engine. It took inter-dependent subtask modal presented as call graphs, and then used 0-1 ILP for task partitioning to maximize energy savings during computing offloading.

CloneCloud (Chun et al. [22]) models the application as a control flow graph. It partitions applications based on an offline static analysis of the various running conditions of the target smartphone and the process binaries on the cloud. Thus, CloneCloud needs to run the analysis again for each newly created application. Based on the application partitioning, it performs ILP to get the optimized offloading decisions.

In the service-based middleware MACS proposed by Kovachev et al. [49] for mobile cloud computing, it had a decision engine that can seamlessly offload computationally intensive parts of the application to nearby infrastructure or a remote cloud. For developers, the application model was the same as the one on the Android platform, which encapsulated computationally intensive parts as Android services and each service had a specific functionality. The services are partitioned based on set conditions or parameters into two groups, one running locally and the other running in the cloud. State information (e.g., CPU load, available memory, remaining battery power on local devices, bandwidth between the cloud and local devices) was monitored by MACS as parameter of decision engine for both the cloud and the local device. Finally, the partitioning and offloading problems were modelled as an ILP and an open-source class library was called to perform decision solving on the Android platform.

Yang et al. [50] studied the application partitioning problem by evaluating the constraints on cloud resources in order to ensure the benefits of service providers. Initially, a single-user computation partitioning problem (SCPP) was considered, and then the resource-constrained multi-user computation partitioning problem (RCMCPP) was derived from the SCPP. The RCMCPP was formulated as a mixed-integer linear programming (MILP) problem to minimize the total execution time. Application providers leveraged the performance-resource-load (PRL) model to design the resource provisioning mechanism to make a tradeoff between application performance and cost of cloud resources and improve utilization of cloud resources.

ParGen et al. [31] used a data flow graph to model relations of subtasks and formulated the QoS optimized task partitioning and offloading problem as a 0-1 ILP problem. A parallel genetic algorithm was then used to maximize the throughput of the multi-user application. The approach allows dynamic partitioning of multiple users and supports sharing of computational requirements among numerous users.

Alameddine et al. [51] aimed to provide delay-sensitive 5G IoT services in different domains, by solving the dynamic joint resource allocation and fine-grained task scheduling problem. The problem was defined and formulated as a mixed-integer linear programming (MILP) model. Given the heterogeneous requirements of offloading task predicted by application models and the limited MEC capabilities, they combined three NP-hard problems of task offloading, application resource allocation, and non-preemptive task scheduling. Finally, they designed the dynamic task offloading and scheduling approach based on LBBD (Logical Benders Decomposition technique) (Hooker et al. [52]). LBBD solved a relaxed master problem with fewer constraints and a subproblem whose solution allows generation of cut sets that iteratively guided the master problem to narrow the search space. Eventually, both the master problem and the subproblem converged to the optimal solution. The running time of this approach was reduced by several orders of magnitude, but did not take other constraints such as energy consumption and cost into consideration.

Ding et al. [53] proposed a decentralized computing offload strategy (DCOS) algorithm to develop a task partitioning and offloading strategy for multi-user multi-mobile-edge scenarios to minimize the application execution overhead (i.e., the weighted sum of energy consumption and computation time). It took MAUI (Cuervo et al. [20]), CloneCloud (Chun et al. [22]), ThinkAir (Kosta et al. [25]), mCloud (Zhou et al. [54]) or ULOOF (Neto et al. [55]) to partition the application into a set of tasks with correlations, which were represented as a program call graph. Then the model solving was transformed into a convex optimization problem, and the optimal offloading decision was derived.

Wu et al. [56] formulated the offloading decision problem in blockchain-enabled heterogeneous IoT-edge-cloud computing environments as a mixed-integer linear programming (MILP) problem and further derived an online and polynomial-time-complexity algorithm by taking advantage of the Lyapunov optimization technique, such that the energy consumption of the IoT device can be minimized when only sacrificing a little delay.

Non-Linear Programming (NLP)

Li et al. [57] used task graphs to construct application models for offloading and parallel execution. The task graph execution time was minimized by choosing the optimal task allocation and scheduling policy by modelling the problem as a mixed-integer nonlinear programming (MINLP) problem. The problem was transformed into a MILP model using the big-M approach. Then, a hybrid approach combining the LBBD principle (Hooker et al. [52]; Emeretlis et al. [58]) and MILP was designed for solving the problem. However, the computing time was high when the number of tasks increased, which cloud not be applied to large-scale applications.

MDP and RL

At present, researchers found that in large-scale offloading scenarios, the traditional programming methods have difficulty in coping with growth of the search space. There have been many works on modelling the task partitioning and offloading problem as the Markov Decision Process (MDP). It then was solved by Reinforcement Learning (RL) or Deep Reinforcement Learning (DRL) (Dash et al. [59]). RL can learn optimal offloading policies by interacting with edge networks, which can achieve model-free control without knowing the system's internal transitions. Traditional reinforcement learning stores tuples containing states, actions, and rewards into Q tables. However, they are not applicable when the state or action space is enormous. DRL can solve the problem of space explosion by integrating neural networks into reinforcement learning to approximate Q values. Therefore, DRL for joint optimization problems has attracted increasing interest from researchers, and complex offloading decision problems can be solved more efficiently using this approach (Wu et al. [60]).

Dab et al. [61] studied the problem of online computing offloading and resource allocation for applications with constraints of limited latency and device resources, and proposed a QL-based QL-JTAR algorithm. QL-JTAR was designed to jointly optimize total energy consumption and total latency. They considered the interdependence of multi-user subtasks, and the system state changes with time slot. The problem was modelled as MDP, and the Q-Learning algorithm based on the value function was used to solve complex large-scale MDPs. Afterwards, the joint computation offloading and resource allocation method was evaluated by simulating real scenarios with real data input using the NS3 simulator. This method considered computational offloading for divisible tasks, but was only suitable for computational offloading for small state spaces.

Meng et al. [62] proposed an online task scheduling algorithm based on DRL, by establishing a system model based on MDP. They designed a new behavior space and reward function expression in order to minimize the task queuing delay and the average energy consumption of the terminal system. The author expressed the behavior space as a Cartesian product model, and used the Cartesian product to describe the scheduling of tasks. However, the model only considered a single user and a single edge server, which could not cope with the complex IoT environment.

Su et al. [63] proposed a spectrum access scheme based on Q-Learning in the mobile network environment to select the optimal spectrum for users. They aimed to improve the QoE of mobile users and maximize the content transmission rate by using QL. The optimal strategy for mobile users to download content from nodes was determined according to the content value function to maximize node utilization, and a non-cooperative Stackelberg game was used to optimize the optimal strategy. The authors combined the QL-based optimal spectrum selection algorithm with the two-stage Stackelberg game content selection algorithm to jointly maximize network throughput and improve node utilization. However, the author did not consider the user's energy consumption and the node's cache consumption when the content was delivered.

Lu et al. [64] proposed a task scheduling algorithm with discrete server selection based on MA-DDPG, in order to reduce system energy consumption, task latency, and task loss rates for mobile devices and edge servers. It used the characteristics of multi-agent continuous learning to solve the problem of environmental instability caused by a single decision maker. It integrated the SAC algorithm with maximum entropy reward function to make Actors in DDPG explore as many actions as possible to obtain more near-optimal path choices. It combined multi-agent DDPG with SAC to solve the problems of instability and small learning in reinforcement learning, and optimized energy consumption, latency, and task discarding rate in offloading.

Tang et al. [65] proposed a model-free online offloading algorithm based on DRL by considering the dynamic load problem of edge servers, in order to minimize long-term costs. In order to improve the cost estimation, the author innovatively integrated LSTM in DRL for predicting future server load. It used LSTM to predict the load information of future servers by adding the load level of the server to the input of neural network. Since the addition of LSTM complicated the original network, it was necessary to compress the network to reduce the number of operations. The algorithm was evaluated based on the demonstration system, which needed to consider the real computing tasks in practical problems.

Cheng et al. [66] proposed a DRL-based joint deep reinforcement learning (FDRL) framework to effectively reduce the training loss and privacy leakage during training phase. They also proposed a joint optimization algorithm for task offloading and resource allocation method based on FDRL. The method was able to effectively protect the data privacy in the drone environment, reduce the amount of original data transmission, and effectively reduce the training loss.

Wang et al. [67] adopted a sequence-to-sequence (S2S) neural network to obtain the task dependencies of applications, and used off-policy reinforcement learning to make task offloading decisions. The training of the S2S neural network was based on the regularly collected task DAG. According to the experimental results, the offloading method based on reinforcement learning achieved better performance than the heuristic algorithm, but it required a lot of training when applied to online computing offloading, which would cause a lot of consumption.

Yan et al. [68] proposed a DRL-based joint optimization task offloading algorithm by taking into account the waste of energy consumption when tasks were discarded. The problem of optimizing long-term latency and system energy consumption in task offloading was solved by combining the DQN-based reinforcement learning method with the cloud-edge collaborative computing offloading. However, its performance was evaluated based on static simulation experiments. So it needs to be migrated to a really dynamic environment to evaluate the performance.

Taxonomy of task partitioning and offloading approaches

Based on the above analysis on the current work of task partitioning and offloading approaches, we see that they were designed for different applications with different users, application models, and model solving approaches. Here we propose a comprehensive taxonomy of these methods by investigating all critical issues involved in the task partitioning and offloading framework. As shown in Fig. 9, the metrics of taxonomy include end-user, inter-dependent subtask model, cost model, profiler, decision engine, decision granularity, decision strategy, and offloading infrastructure.

- **End-user** means the number of mobile end-users considered in the application scenarios, which can be single end-user scenarios or multi-end-user scenarios. Compared with the single end-user scenario, the task partitioning and offloading problem in the multi-end-user scenarios needs to consider more environmental factors. For example, in the situation of limited edge servers, the partitioning and offloading strategy adopted by one end user can be affected by other end-users' strategies, if they offload tasks to the same edge server simultaneously.
- **Inter-dependent subtask model** means the type of graph models to represent inter-dependent subtasks for an application task. It can be program call graph, control graph, data flow graph, object graph, service graph, module graph, and task graph.
- **Cost model** is identified by the metrics of optimization objectives, which are latency, energy consumption, QoS, QoE and economic cost.
- **Profiler** is identified by the type of profile information used by the decision engine. The currently

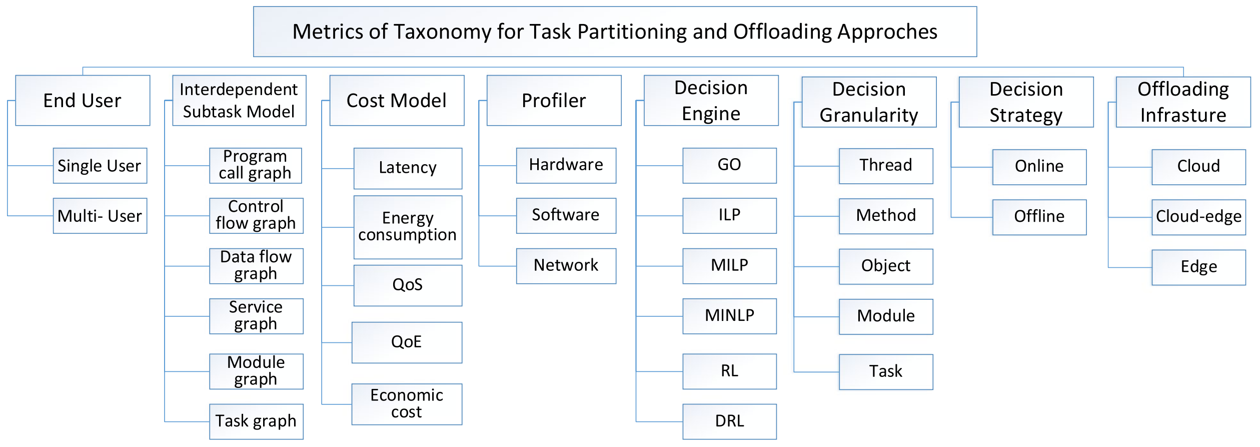

Fig. 9 Comprehensive taxonomy of task partitioning and offloading approaches

used profile information includes: 1) hardware-related, including CPU utilization, RAM and device power, etc. 2) software-related, including data size, call relationships between module structures, throughput, etc. 3) network-related, including network connectivity, bandwidth, transmission rate, etc.

– **Decision engine** is identified by its method used for model solving, such as graph optimization (GO) model, linear programming (LP) model, integer linear programming (ILP) model, mixed integer linear programming (MILP) model, mixed integer nonlinear programming (MINLP) model, reinforcement learning (RL) model, deep reinforcement learning (DRL) model, etc.
– **Decision granularity** refers to the level of granularity used to partition applications for task offloading. Current applications on mobile terminal devices are divided into the following granularity levels:

  (1) Thread-level: partitioning at the thread level of the application.
  (2) Method-level: partitioning and annotating at the method entry and exit of the application.
  (3) Object-level: the objects of an application are partitioned to perform a random search to build an object-relationship graph.
  (4) Module-level: the application is partitioned into independent or interdependent modules and assigned to local or remote servers.
  (5) Task-level: partitioning the application into several subtasks based on task type and function.

– **Decision strategy** is identified by how tasks partitioned (i.e. statically or dynamically) and how offloading decisions are made (i.e. before task execution or during task execution). If tasks are partitioned statically and offloading decisions are made before task execution, we call it online decision strategy. On the contrary, if tasks are partitioned dynamically and decisions are made during task execution, we call it offline decision.
– **Offloading infrastructure** includes traditional cloud and cloud-edge collaborative architecture (Kai et al. [69]). Traditional cloud-based offloading infrastructure means that task modules can run on local devices or remote cloud servers. The cloud-edge collaboration-based offloading infrastructure means that task modules can run on local devices, edge servers, and cloud servers. The edge-based offloading infrastructure means that task can run on local devices or edge servers.

As shown in Table 2, we make a comprehensive comparison on the current work on tasking partitioning and offloading decision optimization approaches based on the proposed taxonomy metrics.

Conclusion and future work

In this paper, we first make a brief review on some representative task partitioning and offloading frameworks for mobile edge computing (MEC) and mobile cloud computing (MCC). Then, we extract a general workflow of task partitioning and offloading, which mainly includes an application model for profiling users' requirements of task partitioning and offloading and a decision engine for solving application models. Most importantly, through making a deep investigation on the current work on the application model and the core of decision engine, we show the challenges and approaches to fulfil users' requirements on delay, energy, and economic cost in IoT systems by making proper task partitioning and offloading decisions with different decision granularity and decision strategy. Based on the survey, we propose comprehensive taxonomy metrics for comparing different task partitioning and offloading approaches in the IoT cloud-edge collaborative computing framework.

From the comparison, we think that the following issues may be further addressed in development of task partitioning and offloading frameworks.

Granularity of task partitioning. When designing the partitioning and offloading algorithms, the granularity of partitioning greatly affects the computational complexity and resource utilization of the service runtime. Partitioning at different levels of granularity can cause many problems, such as subtask dependencies and additional resource overhead. However, because of the diversity of tasks, it is challenging to achieve the optimal level of granularity. It should consider minimal communication overhead for offloading and lightweight mechanisms for distributed platform establishment and management.

Resource constraints and environmental dynamics. In a real cloud-edge collaborative computing offloading environment, it is necessary to consider the limitations of computing and storage capabilities of edge nodes. Besides, since the network environment is unstable, the computational offloading framework needs to adapt to the dynamically changing network bandwidth. In addition, due to the fluctuating network environment and limited resources, failures of task partitioning and offloading may occur. So the framework needs to support the optimization of resource consumption due to task loss caused by unpredictable factors.

Offloading in blockchain-enabled communications systems. Blockchain will play an important role in emerging 6G network infrastructure, since it has the

Table 2 Task partitioning and offloading decision optimization approaches based on the proposed taxonomy metrics

Algorithms	End-user	Inter-dependent Subtask Model	Cost Model	Profiler	Decision Engine	Decision Granularity	Decision Strategy	Offloading Infrastructure
Giurgiu et al. [46]	single user	module graph	latency, energy consumption	hardware, software, network	GO	module	online	cloud
BBAP &MCGAP (Niu et al. [32])	single user	object graph	latency, energy	hardware, network	GO	object	offline	cloud
CATSA (Lakhan et al. [47])	single user	module graph	latency	software, network	GO	module	online	cloud-edge
MAUI (Cuervo et al. [20])	single user	control flow graph	energy consumption	hardware, software, network	ILP	method	online	cloud-edge
CloneCloud (Chun et al. [22])	single user	control flow graph	latency, energy consumption	software, network	ILP	thread	offline	cloud
MACS (Kovachev et al. [49])	single user	service graph	energy consumption, QoS	hardware, network	ILP	module	online	cloud
Yang et al. [50]	single user, multi-user	service graph	QoS, economic cost	hardware, network	MILP	module	offline	cloud
ParGen (Wen et al. [31])	multi-user	data flow graph	latency	hardware, software, network	ILP	module	offline	cloud-edge
DTOS-LBBD (Alameddine et al. [51])	single user	task graph	latency	hardware, software	MILP	task	offline	cloud-edge
DCOS (Ding et al. [53])	multi-user	program call graph	latency	hardware, software	ILP	task	offline	cloud-edge
Li et al. [57]	single user	task graph	latency	software, network	MINLP	task	offline	cloud-edge
QL-JTAR (Dab et al. [61])	multi-user	task graph	energy consumption, latency	hardware, software, network	RL	task	online	cloud-edge
Meng et al. [62]	single-user	data flow	energy consumption, latency	hardware, software	DRL	module	online	edge
Su et al. [63]	multi-user	module graph	latency, QoE	hardware, software	RL	objext	offline	edge
MA-DDPG (Lu et al. [64])	multi-user	service graph	energy consumption, latency	hardware, software, network	DRL	module	online	edge
Tang et al. [65]	multi-user	data flow	latency	hardware, software, network	DRL	task	online	edge
FDRL (Cheng et al. [66])	multi-user	data flow	energy consumption, latency	hardware, software, network	DRL	task	online	cloud-edge
Wang et al. [67]	single-user	task graph	energy consumption, latency, QoS	hardware, software, network	DRL	task	online	cloud-edge
TOLBO (Yan et al. [68])	multi-user	data flow	energy consumption, latency	hardware, software, network	DRL	module	online	cloud-edge

promising possibility to provide fundamentally stable, reliable, and consistent connectivity among clouds, edges and devices (Liu et al. [70]). However, it is of extreme complexity and sophistication to make scalable and reliable offloading decisions, to model offloading costs, and to extract reward due to the dynamic nature of offloading contents, network states, and device states in the future blockchain-empowered networking and computing infrastructure (Alam et al. [71]).

Real experimental evaluation. Most of the existing work evaluates the framework of task partitioning and offloading with simulation experiments. Because the setting of simulation experiments can be ideal, the simulation results cannot reflect performance of the offloading algorithms in practice. Therefore, it is necessary to migrate the method to a real IoT environment to detect the performance of the algorithm in real scenarios.

Acknowledgements
Not applicable.

Authors' contributions
H. Chen wrote the main manuscript text. W. Qin wrote the "MDP and RL" section and completed the Table 2. L. Wang drew Fig. 9. All authors reviewed the manuscript.

Funding
This research was funded by the Natural Science Foundation of Ningbo City under Grant No. 2021J090 and Ningbo Manicipal Commonweal S &T Project under Grant No. 2022S005.

Availability of data and materials
Not applicable.

Declarations

Ethics approval and consent to participate
There is no ethical approval required.

Competing interests
The authors declare that they have no competing interests.

Received: 10 August 2022 Accepted: 12 November 2022
Published online: 03 December 2022

References
1. Patel M, Naughton B, Chan C, Sprecher N, Abeta S, Neal A (2014) Mobile-edge computing introductory technical white paper. White paper, mobile-edge computing (MEC) industry initiative 29:854–864
2. Hu YC, Patel M, Sabella D, Sprecher N, Young V (2015) Mobile edge computing-a key technology towards 5g. ETSI white paper 11(11):1–16
3. Abbas N, Zhang Y, Taherkordi A, Skeie T (2018) Mobile edge computing: A survey. IEEE Internet Things J 5(1):450–465
4. Shi W, Zhang X, Wang Y, Zhang Q (2019) Edge computing: state-of-the-art and future directions. Journal of Computer Research and Development 56(1):73–93
5. Lai P, He Q, Cui G, Xia X, Abdelrazek M, Chen F, Hosking J, Grundy J, Yang Y (2020) QoE-aware user allocation in edge computing systems with dynamic QoS. Futur Gener Comput Syst 112:684–694
6. Sodhro AH, Luo Z, Sangaiah AK, Baik SW (2019) Mobile edge computing based QoS optimization in medical healthcare applications. Int J Inf Manag 45:308–318
7. Ma X, Cui Y, Wang L, Stojmenovic I (2012) Energy optimizations for mobile terminals via computation offloading. In: The 2nd IEEE International Conference on Parallel, Distributed and Grid Computing. IEEE, Piscataway, pp 236–241
8. Kuang Z, Li L, Gao J, Zhao L, Liu A (2019) Partial offloading scheduling and power allocation for mobile edge computing systems. IEEE Internet Things J 6(4):6774–6785
9. Avgeris M, Spatharakis D, Dechouniotis D, Leivadeas A, Karyotis V, Papavassiliou S (2022) ENERDGE: Distributed energy-aware resource allocation at the edge. Sensors 22:660
10. Heidari A, Jabraeil Jamali MA, Jafari Navimipour N, Akbarpour S (2020) Internet of things offloading: Ongoing issues, opportunities, and future challenges. Int J Commun Syst 33(14):e4474
11. Zhang K, Gui X, Ren D, Li J, Wu J, Ren D (2019) Survey on computation offloading and content caching in mobile edge networks. J Softw 30(08):2491–2516
12. Saeik F, Avgeris M, Spatharakis D et al (2021) Task offloading in edge and cloud computing: A survey on mathematical, artificial intelligence and control theory solutions. Comput Netw 195(3):10817
13. Feng C, Han P, Zhang X, Yang B, Liu Y, Guo L (2022) Computation offloading in mobile edge computing networks: A survey. Journal of Network and Computer Application 202(103366). https://doi.org/10.1016/j.jnca.2022.103366
14. Mach P, Becvar Z (2017) Mobile edge computing: A survey on architecture and computation offloading. IEEE Communications Surveys and Tutorials 19(3):1628–1656
15. Jiang C, Cheng X, Gao H, Zhou X, Wan J (2019) Toward computation offloading in edge computing: A survey. IEEE Access 7:131543–131558
16. Wang J, Pan J, Esposito F, Calyam P, Yang Z, Mohapatra P (2019) Edge cloud offloading algorithms: Issues, methods, and perspectives. ACM Comput Surv 52(1):1–23
17. Lin H, Zeadally S, Chen Z, Labiod H, Wang L (2020) A survey on computation offloading modeling for edge computing. Journal of Network and Computer Applications 169:102781
18. Shakarami A, Shahidinejad A, Ghobaei-Arani M (2020) A review on the computation offloading approaches in mobile edge computing: A game-theoretic perspective. Software-Practice and Experience 50(9):1719–1759
19. Wang B, Wang C, Huang W, Song Y, Qin X (2020) A survey and taxonomy on task offloading for edge-cloud computing. IEEE Access 8:186080–186101
20. Cuervo E, Balasubramanian A, Dk Cho, Wolman A, Saroiu S, Chandra R, Bahl P (2010) MAUI: Making smartphones last longer with code offload. The 8th International Conference on Mobile Systems, Applications, and Services (MobiSys'10). Association for Computing Machinery, New York, pp 49–62
21. Richter J, Press M (2010) Applied Microsoft .net Framework Programming, 2nd edn (chap CLR via C#). Microsoft Press, Hoboken
22. Chun BG, Ihm S, Maniatis P, Naik M, Patti A (2011) Clonecloud: Elastic execution between mobile device and cloud. The Sixth Conference on Computer Systems (EuroSys'11). Association for Computing Machinery, New York, pp 301–314
23. Aridor Y, Factor M, Teperman A (1999) cJVM: A single system image of a JVM on a cluster. In: Proceedings of the 1999 International Conference on Parallel Processing (ICPP), IEEE, Piscataway, pp 4–11
24. Zhu W, Wang CL, Lau F (2002) JESSICA2: A distributed Java virtual machine with transparent thread migration support. In: IEEE International Conference on Cluster Computing, IEEE, Piscataway, pp 381–388
25. Kosta S, Aucinas A, Hui P, Mortier R, Zhang X (2012) ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. IEEE INFOCOM 945–953:945–953
26. Zhang L, Tiwana B, Dick RP, Qian Z, Mao ZM, Wang Z, Yang L (2010) Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In: IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), IEEE, Piscataway, pp 105–114
27. Feng X, Ding F, Jie L, Kong X, Ma J (2014) Phone2cloud: Exploiting computation offloading for energy saving on smartphones in mobile cloud computing. Inf Syst Front 16(1):95–111
28. Liu J, Kumar K, Lu YH (2010) Tradeoff between energy savings and privacy protection in computation offloading. The 16th ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED'10). ACM, New York, pp 213–218
29. Xian C, Lu YH, Li Z (2007) Adaptive computation offloading for energy conservation on battery-powered systems. In: The International Conference on Parallel and Distributed Systems (ICPADS), IEEE, Piscataway, pp 1–8
30. Balasubramanian A, Mahajan R, Venkataramani A (2010) Augmenting Mobile 3G Using WiFi. The 8th ACM international conference on mobile systems, applications and services (MobiSys'10). ACM, San Francisco, pp 209–221
31. Wen H, Yang L, Wang Z (2017) Pargen: A parallel method for partitioning data stream applications in mobile edge computing. IEEE Access 6:5037–5048

32. Niu J, Song W, Atiquzzaman M (2014) Bandwidth-adaptive partitioning for distributed execution optimization of mobile applications. Journal of Network & Computer Applications 37(Jan.):334–347
33. Munoz 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(10):4738–4755
34. Kumar K, Lu Y (2010) Cloud computing for mobile users: Can offloading computation save energy? Computer 43(4):51–56
35. Muhammad G (2015) Automatic speech recognition using interlaced derivative pattern for cloud based healthcare system. Clust Comput 18(2):795–802
36. Shi W, Jie C, Quan Z, Li Y, Xu L (2016) Edge computing: Vision and challenges. IEEE Internet Things J 3(5):637–646
37. Zhao T, Zhou S, Song L, Jiang Z, Guo X, Niu Z (2020) Energy-optimal and delay-bounded computation offloading in mobile edge computing with heterogeneous clouds. China Communications 17(5):191–210
38. Rausch T, Nastic S, Dustdar S (2018) EMMA: Distributed QoS-aware MQTT middleware for edge computing applications. In: The IEEE International Conference on Cloud Engineering (IC2E), IEEE, Piscataway, pp 191–197
39. Song Y, Yau SS, Yu R, Zhang X, Xue G (2017) An approach to qos-based task distribution in edge computing networks for iot applications. In: The 2017 IEEE International Conference on Edge Computing (EDGE), IEEE, Piscataway, pp 32–39
40. Aazam M, St-Hilaire M, Lung C, Lambadaris I (2016) MeFoRE: QoE based resource estimation at Fog to enhance QoS in IoT. In: The 23rd International Conference on Telecommunications (ICT), IEEE, Piscataway, pp 1–5
41. Mahmud R, Ramamohanarao K, Buyya R (2018) Latency-aware application module management for fog computing environments. ACM Trans Internet Technol 19(1):1–21
42. Luo Q, Shi W, Fan P, etal (2021) Qoe-driven computation offloading: Performance analysis and adaptive method. In: 2021 13th International Conference on Wireless Communications and Signal Processing (WCSP), Changsha, IEEE, Piscataway, pp 1–5
43. Huang M, Liu W, Wang T, Liu A, Zhang S et al (2020) A cloud-MEC collaborative task offloading scheme with service orchestration. IEEE Internet Things J 7(7):5792–5805
44. He X, Lu H, Du M, Mao Y, Wang K et al (2021) Qoe-based task offloading with deep reinforcement learning in edge-enabled internet of vehicles. IEEE Trans Intell Transp Syst 22(4):2252–2261
45. Lin J (1976) Multiple-objective problems: Pareto-optimal solutions by method of proper equality constraints. IEEE Trans Autom Control 21(5):641–650
46. Giurgiu I, Riva O, Alonso G (2012) Dynamic software deployment from clouds to mobile devices. In: ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing, USENIX, Berkeley, pp 394–414
47. Lakhan A, Li X (2019) Content aware task scheduling framework for mobile workflow applications in heterogeneous mobile-edge-cloud paradigms: CATSA framework. In: The IEEE International Conference on Parallel Distributed Processing with Applications (ISPA'19), IEEE, Piscataway, pp 242–249
48. Wu H, Knottenbelt WJ, Wolter K (2019) An efficient application partitioning algorithm in mobile environments. IEEE Trans Parallel Distrib Syst 30(7):1464–1480
49. Kovachev D, Klamma R (2012) Framework for computation offloading in mobile cloud computing. International Journal of Interactive Multimedia and Artificial Intelligence 1(7):6–15
50. Yang L, Cao J, Cheng H, Ji Y (2015) Multi-User Computation Partitioning for Latency Sensitive Mobile Cloud Applications. in IEEE Transactions on Computers 64(8):2253–2266
51. Alameddine HA, Sharafeddine S, Sebbah S, Ayoubi S, Assi C (2019) Dynamic task offloading and scheduling for low-latency iot services in multi-access edge computing. IEEE J Sel Areas Commun 37(3):668–682
52. Hooker JN, Ottosson G (2003) Logic-based benders decomposition. Math Program 96(1):33–60
53. Ding Y, Liu C, Zhou X, Liu Z, Tang Z (2020) A code-oriented partitioning computation offloading strategy for multiple users and multiple mobile edge computing servers. IEEE Trans Industr Inf 16(7):4800–4810
54. Zhou B, Dastjerdi AV, Calheiros RN, Srirama SN, Buyya R (2017) mCloud: A context-aware offloading framework for heterogeneous mobile cloud. IEEE Trans Serv Comput 10(5):797–810
55. Neto JLD, Yu S, Macedo DF, Nogueira JMS, Langar R, Secci S (2018) ULOOF: A user level online offloading framework for mobile edge computing. IEEE Trans Mob Comput 17(11):2660–2674
56. Wu H, Wolter K, Jiao P, Deng Y, Zhao Y, Xu M (2021) EEDTO: An Energy-Efficient Dynamic Task Offloading algorithm for blockchain-enabled IoT-edge-cloud orchestrated computing. IEEE Internet Things J 8(4):2163–2176
57. Li S, Chen W, Chen Y, Chen C, Zheng Z (2019) Makespan-minimized computation offloading for smart toys in edge-cloud computing. Electron Commer Res Appl 37:100884
58. Emeretlis A, Theodoridis G, Alefragis P, Voros N (2016) A logic-based benders decomposition approach for mapping applications on heterogeneous multicore platforms. ACM Trans Embed Comput Syst 15(1):19
59. Dash SK, Dash S, Mishra J et al (2020) Opportunistic mobile data offloading using machine learning approach. Wirel Pers Commun 110(1):125–139
60. Wu H, Li X, Deng Y (2020) Deep learning-driven wireless communication for edge-cloud computing: Opportunities and challenges. Journal of Cloud Computing 9(21):1–14
61. Dab B, Aitsaadi N, Langar R (2019) Q-learning algorithm for joint computation offloading and resource allocation in edge cloud. In: Proceedings of 2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), IEEE, Piscataway, pp 45–52
62. Meng H, Chao D, Guo Q (2019) Deep reinforcement learning based task offloading algorithm for mobile-edge computing systems. In: Proceedings of the 2019 4th International Conference on Mathematics and Artificial Intelligence, IEEE, Piscataway, pp 90–94
63. Su Z, Dai M, Xu Q et al (2020) Q-learning-based spectrum access for content delivery in mobile networks. IEEE Transactions on Cognitive Communications and Networking 6(1):35–47
64. Lu H, Gu C, Luo F et al (2020) Optimization of task offloading strategy for mobile edge computing based on multi-agent deep reinforcement learning. IEEE Access 8:202573–202584
65. Tang M, Wong V (2022) Deep reinforcement learning for task offloading in mobile edge computing systems. IEEE Trans Mob Comput 21(1):1985–1997
66. Cheng Z, Gao Z, Liwang M, Huang L, Du X, Guizani M et al (2021) Intelligent task offloading and energy allocation in the uav-aided mobile edge-cloud continuum. IEEE Netw 35(5):42–49
67. Wang J, Hu J, Min G, Zhan W, Zomaya A, Georgalas N et al (2022) Dependent task offloading for edge computing based on deep reinforcement learning. IEEE Trans Comput 71(10):2449–461
68. Yan L, Chen H, Tu Y, Zhou X et al (2022) A task offloading algorithm with cloud edge jointly load balance optimization based on deep reinforcement learning for unmanned surface vehicles. IEEE Access 10(1):16566–16576
69. Kai C, Zhou H, Yi Y, Huang W (2021) Collaborative cloud-edge-end task offloading in mobile-edge computing networks with limited communication capability. IEEE Transactions on Cognitive Communications and Networking 7(2):624–634
70. Liu Y, Su Z, Wang Y (2022) Energy-efficient and physical layer secure computation offloading in blockchain-empowered internet of things. IEEE Internet Things J. https://doi.org/10.1109/JIOT.2022.3159248
71. Alam T, Ullah A, Benaida M (2022) Deep reinforcement learning approach for computation offloading in blockchain-enabled communications systems. J Ambient Intell Humaniz Comput. https://doi.org/10.1007/s12652-021-03663-2

Publisher's Note