Edge Cloud Offloading Algorithms: Issues, Methods, and Perspectives

JIANYU WANG and JIANLI PAN, University of Missouri-St. Louis
FLAVIO ESPOSITO, Saint Louis University
PRASAD CALYAM, University of Missouri-Columbia
ZHICHENG YANG and PRASANT MOHAPATRA, University of California, Davis

Mobile devices supporting the "Internet of Things" often have limited capabilities in computation, battery energy, and storage space, especially to support resource-intensive applications involving virtual reality, augmented reality, multimedia delivery, and artificial intelligence, which could require broad bandwidth, low response latency, and large computational power. Edge cloud or edge computing is an emerging topic and a technology that can tackle the deficiencies of the currently centralized-only cloud computing model and move the computation and storage resources closer to the devices in support of the above-mentioned applications. To make this happen, efficient coordination mechanisms and "offloading" algorithms are needed to allow mobile devices and the edge cloud to work together smoothly. In this survey article, we investigate the key issues, methods, and various state-of-the-art efforts related to the offloading problem. We adopt a new characterizing model to study the whole process of offloading from mobile devices to the edge cloud. Through comprehensive discussions, we aim to draw an overall "big picture" on the existing efforts and research directions. Our study also indicates that the offloading algorithms in the edge cloud have demonstrated profound potentials for future technology and application development.

CCS Concepts: • **General and reference** → **Surveys and overviews**; • **Networks** → **Cloud computing**; **Network resources allocation**; *Mobile networks*; • **Theory of computation** → **Mathematical optimization**;

Additional Key Words and Phrases: Internet of Things, edge computing, mobile edge computing, offloading algorithms, mathematical models

ACM Reference format:
Jianyu Wang, Jianli Pan, Flavio Esposito, Prasad Calyam, Zhicheng Yang, and Prasant Mohapatra. 2019. Edge Cloud Offloading Algorithms: Issues, Methods, and Perspectives. *ACM Comput. Surv.* 52, 1, Article 2 (February 2019), 23 pages.
https://doi.org/10.1145/3284387

This work has been partially supported by a University of Missouri Research Board (UMRB) award and the National Science Foundation award CNS-1647084.

Authors' addresses: J. Wang and J. Pan, 326 Express Scripts Hall, One University Blvd., St.Louis, MO 63121; emails: {jwgxc, pan}@umsl.edu; F. Esposito, Ritter Hall 217, 220 North Grand Blvd., St. Louis, MO 63103; email: espositof@slu.edu; P. Calyam, 221 Naka Hall, Columbia, MO 65211; email: calyamp@missouri.edu; Z. Yang and P. Mohapatra, 2063 Kemper Hall, Davis, CA 95616; emails: {zcyang, pmohapatra}@ucdavis.edu.

0360-0300/2019/02-ART2 $15.00
https://doi.org/10.1145/3284387

1 INTRODUCTION

We are embracing the future world of 5G communication and the Internet of Things (IoT). Smart mobile devices are becoming more popular and playing increasingly important roles in every aspect of our daily life [30]. Various applications running on these mobile devices require not only bounded latency, wide bandwidth, and high computation performance but also long battery life [10]. In these cases, mobile devices alone are insufficient, as they suffer from limited local capabilities of computation and energy to deliver performant resource-intensive applications. Therefore, such a resource gap is filled by remote and centralized data centers or clouds services, such as Amazon Web Services [1], Microsoft Azure [3], and Google Cloud [2]. These centralized clouds (CCs) can offer virtually unlimited computation, networking, and storage resources. For many years, the cloud elasticity model has been widely successful and an added value for both enterprises and cloud providers. However, recent advances in resource-intensive IoT applications, such as face recognition, ultra-high-definition video, augmented reality (AR), virtual reality (VR), and voice semantic analysis, are challenging the scalability and resiliency models of the traditional cloud computing with more rigorous demands in response latency and data storage. Moreover, the current limited bandwidth of the backbone network cannot afford the back-and-forth transmission of the exponentially increasing amount of data generated by the future IoT devices for the ever-increasing mobile applications.

To tackle the above challenges, Edge Clouds (ECs) (also called "fog computing" [9] or "Mobile Edge Computing" [33] in some articles) have been proposed. The core idea is to add resources at the network edge; in particular, computation, bandwidth, and storage resource are moved closer to the IoT devices to reduce the backbone data traffic and the response latency and to facilitate the resource-intensive IoT applications. EC has relatively smaller computation capacity compared to CC but takes advantage of short access distance, flexible geographical distribution, and relatively richer computational resource than mobile devices.

Other than the common mobile applications, ECs are also more suitable for some special circumstances. For example, in disaster or battlefield environments, ECs can be very useful in providing uninterrupted communication and handling intensive parallel tasks with high accuracy and low communication latency even if the backbone network access is not available. The second example is in health monitoring or remote access to medical devices [14, 34], where patients are equipped with numbers of wearable sensors to monitor vital signs in real time. ECs could process the data collected from these sensors to extend their battery life and generate quick responses in emergencies to save lives. Finally, with the advent of IoT, ECs can play a key role to build a fundamental tier of IoT systems for much more distributed applications in smart home, smart health, smart vehicles, and even smart cities [30]. Recent research such as HomeCloud framework [31] and Incident-Supporting Visual Cloud [16] concentrate on the combinative applications between EC and IoT.

In such an edge cloud vision, the EC offloading problem, i.e., the problem of transmitting a workload from a mobile device to the ECs, is one of the principal challenges. Offloading algorithms are of central importance for an efficient coordination between the ECs and the mobile devices. There are the common and unique aspects between CC and EC in the offloading study. For the common aspect, both of them consider the factors including device energy, bandwidth utilization cost, network connectivity, cloud workload, and application latency. For the unique aspects, the offloading of EC refers to the requirements of the real-time application, service availability, high bandwidth for the huge volume of offloading data, context awareness of information in the device level, and network level close to users and high mobility [8]. The uniqueness of EC comes from the new heterogeneity problems of the variability of mobile or IoT applications, which requires the cloud vendors to provide different services, infrastructure, platforms, communication media, and enabling technologies for application offloading.

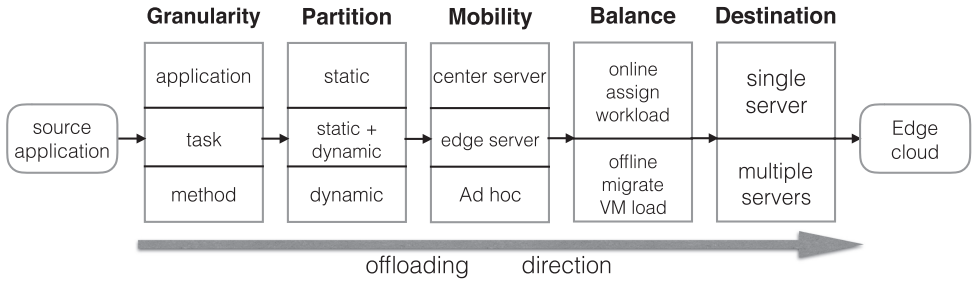

Fig. 1. The whole process of offloading from mobile devices to an edge cloud.

Our article surveys the recent representative offloading algorithms. In particular, we adopt a novel characterizing model that serves as taxonomy to study process of offloading from mobile devices to the ECs. The process responsibility is divided among three main agents: mobile devices, communication links, and ECs. Specifically, mobile devices are responsible for determining how an application is partitioned, which parts should be executed locally or remotely, and the offloading scheme. The communication link is influenced by fluctuation of bandwidth, connectivity, and device mobility. EC servers handle the balance of the server load to achieve maximum service rates and system throughput. Given this offloading model, we classify existing solutions using five dimensions: *offloading destination*, *EC load balancing*, *user devices mobility*, *application partitioning*, and *partition granularity*, as illustrated in Figure 1 from right to left. Such classification covers the whole offloading process in a sequential order from mobile devices to ECs. By analyzing the problem definition, mathematical models, and optimization solutions, each algorithm discussed in this article stands for a typical and creative research direction. Several other survey papers related to edge cloud [4, 26, 44] discuss edge cloud computing in different domains (such as edge computing architecture, communication, computation offloading, and use case studies). However, our article differs from them by collecting offloading algorithms and analyzing their mathematical models from a holistic comprehensive perspective.

The rest of this article is organized as follows. Section 2 introduces scenarios of the single and multiple servers as offloading destinations. Section 3 shows the online and offloading methods to dynamically balance server load. Section 4 analyzes scenarios in which mobile devices lose connectivity due to their continuous movement and the corresponding solutions. Section 5 presents the schemes to offload partitioned components according to their internal execution order and cost. Section 6 reviews the granularity of application partitioning and explains their advantages and disadvantages. Section 7 discusses the related mathematical models, future challenges, as well as technology trends. Finally, the conclusions follow in Section 8.

2 OFFLOADING DESTINATION—SINGLE SERVER VS. MULTIPLE SERVERS

Edge offloading is a strategy to transfer computations from the resource-limited mobile device to resource-rich cloud nodes to improve the execution performance of mobile applications. The selection of cloud servers is worth careful consideration at the beginning of the design phase of an offloading algorithm. The workload of an application at the runtime could be offloaded to only one server for sequential execution or to multiple servers for parallel execution to guarantee the user experience including lower response latency and energy consumption. Figure 2 shows the framework of mobile cloud computing that illustrates communication relationship among all functional components. User devices are distributedly located at the edge of the network. They could offload computation to EC servers via WiFi or cellular networks. If a single EC server is insufficient to

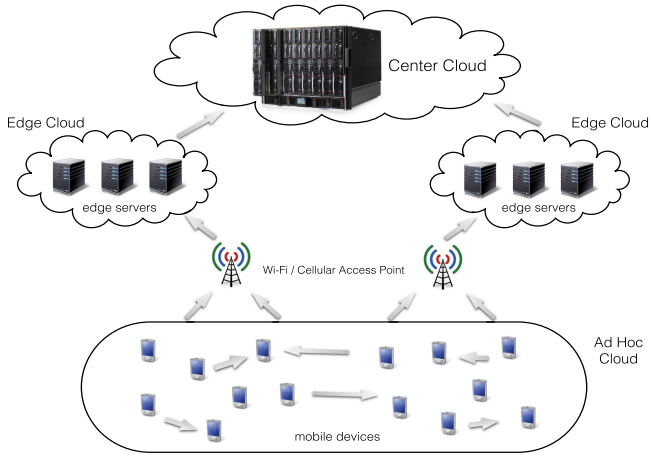

Fig. 2. Mobile cloud computing framework.

meet the latency requirement, then other EC servers or CC servers are made available to assist the application by sharing workload. Current studies on offloading focus on the multiple applications environment, where the requirements of network and computing resources based on the type of applications. For example, processes whose execution follow a sequential order are suitable to be offloaded to one single server, since the communication among serialized parts is frequent. However, applications with repetitive computation are more suitable for parallelized servers. In this section, we discuss some representative algorithms based on such offloading destination taxonomy dimension.

2.1 Single Server

In this section, we discuss MAUI [15] and CloneCloud [12] two algorithms that utilize a single-sever offloading strategy.

2.1.1 MAUI. MAUI is a fine-grained approach that offloads parts of programs remotely to solve the mobile energy problem [15]. The remote server could be a CC server or a nearby EC server at a WiFi access point. As a pioneer of all offloading systems, the MAUI offloading strategy takes advantage of program partitioning and full process migration, which also reduces developers' programming burden.

The architecture of MAUI follows a client–server model. Both the server and mobile devices have three functional components: proxy, profiler and solver, as illustrated in Figure 3. The proxy is used to transmit data and control instructions. The profiler retrieves the data about program requirement, execution energy cost, and the network environment, while the solver decides the program partitioning strategy. Under such an architecture, MAUI models communication cost and computation cost with a 0-1 integer linear optimization problem that is derived from the method called graph. The graph is a flow diagram that presents the computation cycles, energy cost, and data size at each stage of the application execution. When a method is called and a remote server is available, the optimization framework dynamically determines whether the method should be offloaded to maximize the total energy saving or not. The problem is defined as follows:

$$\sum_{v \in V} I_v \times E_v^l - \sum_{(u,v) \in E} |I_u - I_v| \times C_{u,v},$$

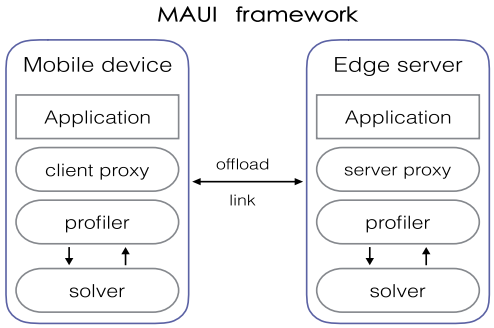

Fig. 3. MAUI system model.

with the following constraints:

$$\sum_{v \in V} \left((1 - I_v) \times T_v^l + \left(I_v \times T_v^r \right) \right) + \sum_{(u,v) \in E} (|I_u - I_v| \times B_{u,v}) \leq L,$$

where E_v^l is energy cost by executing method locally; $C_{u,v}$ is energy cost of tranferring data between nodes; $B_{u,v}$ is time of transferring data; L is time latency limit; l is locally; r is remotely; v, u are vertices on the call graph; and I_v is 0-1 choice.

Specifically, MAUI first helps program developers simply mark the methods as remotable that could be considered offloading to a server, and then MAUI automatically decides which methods should be offloaded with the aid of programming reflecting and type-safety to manage program behavior. Second, at runtime, the profiler starts to periodically collect system information from three factors: device, program, and network. Third, based on the factors information, a linear program solver uses data input by profilers to find a global optimization way for offloading. After obtaining offloading strategy, the proxies on the mobile devices and the server starts to implement the solution, while the mobile application performs simultaneously. Proxies handle the exchange of priority to execute code locally or remotely. When the program on the server calls a method, which is assigned to a local device, the server transfers the execution control to the mobile device, and vice versa. Therefore, the synchronization between local and remote must work in serialization even if there is corresponding data transmission overhead.

Besides the system framework and the offloading algorithm, the authors in MAUI also investigated several approaches to evaluate the system's macro and micro performance. The macro-benchmarks contain energy consumption, performance of mobile applications and ability of supporting resource-intensive applications. The micro-benchmarks evaluate the overhead of each system components, the adjustment of each algorithm parameters, the change of network environment, and the CPU costs. To test the effectiveness of MAUI, three kinds of popular applications on mobile device are being researched: resource-intensive face recognition, latency-sensitive video games, and voice language translation. With their implementation of MAUI, authors demonstrate that 27% of the energy of the smartphone is saved for video games, 45% for a chess game, and even more than 85% for face recognition. Meanwhile, the authors also show how the latency improves by a factor of 4.8 times using nearby edge cloud servers.

2.1.2 CloneCloud. Similarly to MAUI, CloneCloud is also a fine-grained approach that automatically identifies the communication and computation costs for migrating the workload from local to edge cloud [12]. However, CloneCloud does not require any developer efforts in marking whether a part of the program can be offloaded or not. The application is unmodified, and

the suitable portions of its execution are offloaded to the edge end. CloneCloud owns a flexible application partitioner and executes the workload from mobile devices on the task-level virtual machines (VM) such as Java virtual machine and .NET.

To establish a CloneCloud framework, a static analyzer, as the first step of partitioning mechanism, is used to discover constraints of the application to be executed on edge servers. The analyzer then determines the legal executable parts that qualify this set of constraints. If a code segment performs expensive processing and satisfies the constraints, then it runs on the cloud server. There are three kinds of constraints for the analyzer. First, the chosen methods should not use the specific features that are pinned to the mobile machines such as GPS and various sensors. The methods of these features are natively embedded with hardware. Second, the methods that shared native state should be allocated to the same VM when they serve for the same application process. Third, the caller-caller relation among methods is monitored. If a partition point is located in the caller, then it should not be in the callee. Such nested migration is prohibited to avoid multiple triggerings of the same migration at the same partition points.

The dynamic profilers collect the necessary data to build cost models based on the outputs of the analyzer when the various applications adapt different execution configuration. The cost model of an execution trace of an application is presented as a profile tree data structure, where nodes represent methods and edges represent cost values, such as execution time and energy consumption. Next, a mathematical optimization solver determines the migration points where the workload is offloaded at runtime to minimize the overall cost of the mobile devices.

Finally, the chosen program methods are offloaded to an available server with a clone VM installed. When the execution process on the mobile device reaches to the migration points, its process is suspended, and its current state is packaged and transmitted. The clone VM will initiate a new thread with the packaged state in the stack and heap objects. Then the application process resumes on the cloud server. When the assigned tasks are completed, the application state is repackaged and shipped back to the original mobile device, and then the process on mobile device resumes.

With the help of the above offloading framework, CloneCloud achieves application partitioning and seamless cooperation between the local and the remote virtual processing instance. Experiments on tasks such as virus scanning and image search show that the algorithm helps applications achieve up to 20 times speedup and a 20-fold decrease in energy consumption.

Aside from the advantages brought by offloading strategies, a set of new challenges appears. One is the inability to easily offload the workload caused by native functional modules such as a camera, GPS, and sensors. Second, when the offloading strategies try to permit perfunctory concurrency between unoffloaded and offloaded workloads, the synchronization of application data should be prudently considered to keep data updated.

2.2 Multiple Servers

With the prosperity of computation-intensive applications, we are facing more serious challenges on the energy and latency-sensitive for applications such as multimedia, three-dimensional (3D) modeling of a disaster site and unmanned driving. In these cases, the tolerance of execution latency is relatively rigorous to meet customer requirements. Especially when the execution latency constraints are severe, a single server or VM may be unable to provide sufficient communication bandwidth and computing capability. To this aim, solutions handling parallel offloading on multiple servers have been proposed to distribute partitioned tasks to a cluster of servers that have different capacities of computation and communication resources. In the next subsections, we discuss a few representative solutions that adopt this model.

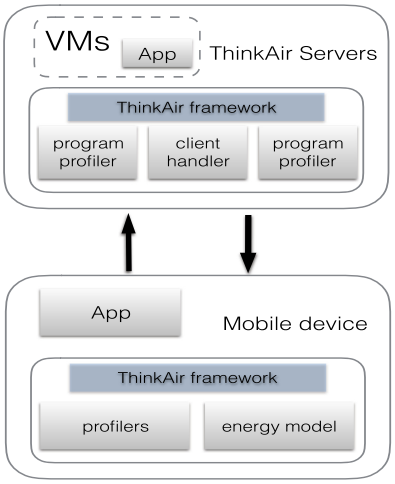

Fig. 4. Functional components in ThinkAir framework.

2.2.1 ThinkAir. After the publication of MAUI and CloneCloud, ThinkAir [21] was proposed to cope with the disadvantages of these two systems; in particular, ThinkAir extends in new ways for resource allocation and parallel task execution.

ThinkAir attains scalability by providing VMs that run the same smartphone execution environment on the edge nodes synchronously. Moreover, the system improves the performance compared to CloneCloud by dynamically allocating cloud resources instead of statically analyzing partitions of applications. ThinkAir achieves such scalability and flexibility through two perspectives. First, parallel execution on several edge servers can satisfy the high computation requirements of mobile applications such as face recognition. The system can divide a calculation problem into subproblems to execute on the multiple VMs to reduce the waiting interval of the tasks between cloud and mobile devices. Second, the tolerance of energy consumption and latency fluctuates due to the resouce types of applications, the hardware performance of mobile devices, the limited battery capacity, and the specific user configurations. Besides the above two advantages, ThinkAir also deals with the unstable connectivity of cloud service to guarantee the precise execution of applications.

Similarly to MAUI, edge cloud application developers using ThinkAir would still need to modify the code for running application smoothly. However, such a modification workload seems less than the one required in MAUI's, because ThinkAir provides programmers with a customized API and a compiler. Moreover, ThinkAir provides an execution controller that determines the necessity to offload a program method. When the method is executed for the first time, the decision is only based on the environmental parameters. The subsequent execution decision is determined by the combination of factors including latency and energy cost in the past invocation and environmental data. There are execution controllers both on the user devices and cloud servers to determine the offloading node according to the requirement of execution time and energy.

Figure 4 shows an overall ThinkAir system framework and components. In the architecture of ThinkAir, the client handler and the profilers with rich resource are informed of the possibility of performing code migration. The client handler is deployed on the cloud servers to execute the tasks that require multiple VM clones in parallel. Moreover, the handler manages communication protocol, network connection, receiving and executing offloading code, and return results that are sent to the profiler for future offloading decision. The profilers consist of three modules: hardware, software, and network. Specifically, the hardware profiler monitors the data of CPU, WiFi, and 3G,

which are also integrated into an energy estimation model. The software profiler records the data reported during application execution such as running time, CPU efficiency, and memory usage, wherever in local or on cloud. The network profilers combine both intent and instrumentation profiling including bandwidth, response time, and data of network interfaces. By utilizing the dynamic information collected with the monitoring processes within profilers, the system uses an energy estimation model so that the client handler can make efficient offloading decisions.

2.2.2 Cloudlet. In contrast to ThinkAir, the Cloudlet is located at the closest access point that is the next hop to be connected to the Internet. The concept of cloudlet was initially proposed by *Satyanarayanan* [36]. He introduced the cloudlets as resource rich servers at the edge of the network located nearby WiFi access points. The mobile user could rapidly initiate VMs on the edge servers to offload its resource-intensive computation. The main design goal of a cloudlet was to reduce application latencies compared to a case in which mobile devices are left on their own.

Despite the significant technology advancement provided by such a cloudlet system, Verbelen et al. [41] pointed out two drawbacks of this VM-based cloudlet strategy. First, the cloudlets are provided by Internet Service Providers in their local area network (LAN) where cloudlets may have reduced range of operation and their configurations may fail to meet the execution requirements of some applications. Second, VM-based cloudlets run the whole application offloaded in a single VM. However, the resources on the cloudlet are limited. When multiple applications are required to run simultaneously on the cloudlet, the cloudlet service will be impacted, e.g., some user requests could be declined. To overcome the above two limitations, the authors of Reference [41] further proposed an elastic architecture of cloudlets. Their architecture not only provides original cloudlet servers at the edge but also organizes ad-hoc clouds that involve other devices with available resource in the same LAN.

To deploy these new cloudlets, mobile applications are divided into different components that can be transmitted to other devices or basic cloudlet servers managed by a Cloudlet Agent (CA) running on a powerful server within the ad-hoc network. Such a component-level framework allows users to join and leave the cloudlet at runtime without severely impacting application performance. Each available device is regarded as single node that carries on a Node Agent (NA) and multiple nodes located in the physical proximity form a cloudlet that is monitored and controlled by the CA. When the offloading request appears, the NAs will estimate the execution environment and share the information with CAs. Based on the global view of resources on the nodes, CA can form a globally optimal solution. When nodes enter or quit the service coverage of the current cloudlet, the CA would perform calculation again and decide whether to migrate again some service components.

An augmented reality application is used to evaluate the cloudlet performance by splitting the application into five components: video source, renderer, tracker, mapper, relocalizer, and object recognizer. The experiment shows how such flexible cloudlets indeed boost application performance. However, we must note that this kind of cloudlet needs to tackle execution scheduling of components carefully. The data synchronization among many of mobile devices and cloudlet servers can affect the global performance. In these cases, one cloudlet handles computation and data transmission for multiple applications. Meanwhile, an application may also split workload to multiple cloudlets. Given the complexity of offloading and high accuracy of synchronization, application developers and algorithm designers may face additional challenges.

3 OFFLOADING BALANCE—ONLINE OR OFFLINE

In this section, we focus on offloading balance on solutions for distributed edge clouds despite their location. Before or during the offloading, the amount and frequency of offloading requests from

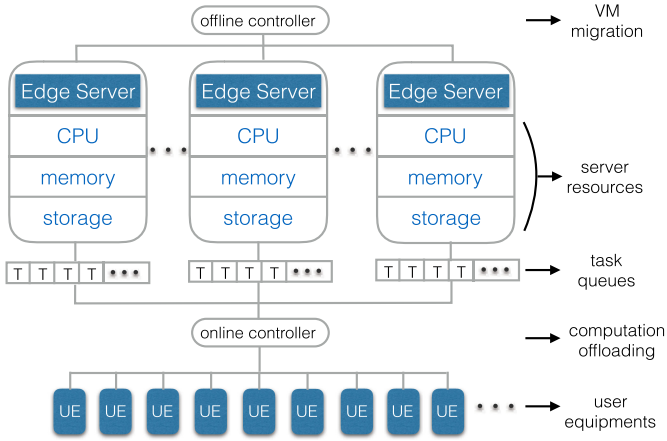

Fig. 5. The framework of balancing control system.

multiple end-users are dynamically floating in real time. Due to the resource limitation, the edge cloud may not have enough available resources to meet the Service Level Agreements (SLA) of the application where not all user or application requests would be accepted. The system needs the online scheduling algorithms to determine the order of requests before the offloading process is finally performed to improve users' Quality of Experience (QoE). After the requests offloading, the edge servers are partially or fully occupied by the computing workload. In these cases, the weakness of the previous online offloading strategies may lead to unbalanced loads among the edge servers, where some parts of them may work with very low loads while others are almost fully utilized. Therefore, some researches focus on the offline balancing among the servers through workload migrations to avoid servers overloaded and improve resource utilization. Under these scenarios, we classify the literature into two types of algorithms, online and offline, whose difference lies in the time to perform the balancing algorithm, before or after the requests offloading. Figure 5 presents the framework of the balanced offloading system and the relationship among its functional components. An online controller distributes the tasks from users to edge servers and an offline controller migrates the workload among edge servers according to the resource usage. Next, we discuss a few relevant works based on such a system.

3.1 Online Balancing

Online balancing is a pre-processing or runtime method that ensures that the workloads are distributed to the appropriate servers when the user requests arrive at the cloud in real time. Online algorithms consider both user configuration requirements and current servers' available capacity without any knowledge of future resource requests.

3.1.1 Online-OBO and Online-Batch. Considering limited resources and processing abilities on the edge cloud, Xia et al. [48] proposed an online request admission algorithm to maximize the system throughput. The offloading environment is a simple system where mobile devices connect to the cloudlet through an access point. There are different types of resources on the cloudlet such as network bandwidth, computing power, data storage space, and service time slots, where any type of resource is associated with an estimated cost value.

The authors first propose an abstract admission cost model to determine different contributions from different resources in a cloudlet with K types of resources. Then they propose and implement a set of algorithms that handle the online requests admissions on the cloudlet without knowing

the future request arrival rate according to the situation of resources occupancies on the cloudlet. When a request arrives at the system, it will be rejected immediately if there is insufficient capacity for every requested resource. Otherwise, the system verifies whether the cost is under a given feasible threshold and, if yes, it admits the request. After completing each request, the allocation mapping is updated, which contains all the functional components of the system.

Given the above admission control mechanism, the throughput maximization problem is modeled using a reduction from the online K-dimensional bin packing problem, a reduction typically utilized to solve space management problems. Two use cases are discussed in the Reference [48]: one request arrival per time slot and multiple requests arrivals per time slot. The former use case is implemented with the idea discussed above, referred to as Online-OBO. The latter use case adopts a greedy strategy to try to handle a group of requests based on the system cost model, referred to as Online Batch. Once a request is rejected, it will be removed for further consideration in the current time slot. It is worth mentioning that these two algorithms dynamically adjust some parameters in their mathematical models to offer convenient configuration of server workload for service providers.

3.1.2 Primal-dual Approach. Compared to the above online allocation approach where the users offload their workload to a single cloudlet server, Hao et al. [18] proposed an algorithm for allocating VMs in a distributed cloud that consists of geographically diversified small data centers close to users. The algorithm takes users' specified constraints into consideration, including server geographic locations, restrict on resource cost, and communication latency for achieving:

- **a.** Balance between optimal revenue and cloud performance. From the aspect of revenue, the system attempt to accept user requests as much as possible in the constant service time. Meanwhile, the system must maintain good performance to satisfy the SLA.
- **b.** Generate the optimal solution without information of future arriving requests. The algorithms focus on obtaining the best allocations for the current requests based on the existing cloud distribution.
- **c.** Be flexible to handle different resource constraints such as VM location, VM service duration, Inter-VM distance, and cost policy of the service provider.

Due to the complexity of resource constraints and performance goals, the offloading is converted into an NP-hard problem that could be solved by an approximate approach. Once a request is receieved, the system tries to solve the primal solution and its dual solution, where the former decides VM allocation and the later present the upper limit for the optimal allocation. The primal and dual problem could be represented and solved by linear programming equations. Overall, this algorithm is a generalized solution through a comprehensive NP-hard approximation, considering the limits of server resources in both central and edge cloud architecture.

3.1.3 Stochastic Models. Besides the above method to transfer the offloading problem to a bin packing problem, a stochastic model was proposed by Maguluri et al. with the aim of finding the maximum system throughput under various theoretical or practical constraints. The model is stochastic, since the authors assume that the user requests arrive by way of a stochastic process [27]. The authors studied two popular algorithms, pointed out their disadvantages, and improved the MaxWeight method [46]. Offloading is in the form of VMs employed in multiple servers where each VM contains various types of resources.

The authors analyze several algorithms using a centralized allocation strategy in which all user requests are received and handled by a central scheduler. First, the Best-Fit method is proved not optimal with a case study. Second, they show how the current MaxWeight approach [46] is not optimal either, since it is only suitable for some ideal condition, where the offloaded

workload can be migrated between cloud clusters without a high cost. However, the MaxWeight algorithm is shown to be costly in practice, since the task execution may be disrupted when the tasks are all allocated at the beginning of each time slot. Third, the author also present a non-preemptive MaxWeight algorithm is designed to allocate workload based on the current server capacity.

The article also discusses an online algorithm in which every server has an individual queue for job requests to route workloads as soon as they arrive. When the global balancing is considered, the join-the-shortest-queue routing is utilized to cooperate with MaxWeight algorithms. The scheduler allocates requests arrived according to the updated queue information and VM configuration. When the arrival rates of requests and the number of servers increase, the information of queue length leads to considerable communication overhead. In this case, Power-of-two-choices method with myopic MaxWeight helps reduce the costly overhead when all servers are identical. Specifically, two servers are randomly sampled, and the user request will be allocated to the server whose queue has less delayed jobs. In contrast, the pick-and-compare scheduling randomly chooses a server and compares it with the one allocated in the last time slot. All the above algorithms provide us with optimal ways for throughput under specific system requirements to keep online load balance.

3.2 Offline Balancing

Offline algorithms aim at balancing over-utilized resources on the edge servers. Since different servers contain a variety of services, they may have different occupancy percentages for computation and communication resources. When a server is overloaded, load migration may be performed to avoid service disruptions and wasting of resources, i.e., a user request may be rejected because one type of the required resource cannot be satisfied. The traditional cloud VM migration mainly aims at maintaining higher system utilization and lowering the energy consumption for the physical server that is underloaded. Besides, the VM migration of edge cloud to guarantee the flexible resource provisioning and quality of cloud service since the type and number of resource requests from user devices are fluctuating.

3.2.1 Resource Intensity Aware Load. VMs are deployed on the physical machines (PMs) that have limited hardware resources. When one type of resources on the PM is close to be completely occupied, the new requests for VM initiation will be rejected even if the usage of other resources is at a low level. Due to this dilemma, such unbalanced allocation leads to waste of computation and energy resources of PMs and the system cannot reach the optimal throughout performance. Therefore, the Resource Intensity Aware Load (RIAL) balancing algorithm is proposed to efficiently migrate VMs among cloud severs while ensuring low migration cost at the same time [11].

Considering resource intensity, we mean that the amount of resource type is demanded for the service. A program may ask for several VMs simultaneously to support different functions, leading to intensive communication between these VMs. RIAL dynamically allocates the offloaded workload to the edge servers based on their current usage of computing resources. However, the VMs that exchange data commonly will be deployed in the same server to avoid migration cost. Meanwhile, the migration among PMs also maintains minimum performance degradation.

For reducing the possibility of overloading, RIAL periodically checks the resource usage on each PM, seeking and migrating the VMs among PMs. Both the migrated VMs and the PMs as a destination are derived by the multi-criteria decision-making method (MCDM), which establishes decision matrix within all types of resources. The ideal VM to migrate owns the highest occupancy of one type of resource and lowest utilization rate of another, as well as the least data transmissionamong other VMs.

MCDM calculates the Euclidean distance between each VM and PM based on the corresponding migration cost. The VM with the shortest distance is selected. The detailed equation of Euclidean distance is as follows:

$$D = \sqrt{\sum_{k=1}^{K} [w_{ik}(x_{ijk} - r_{ik})]^2 + (w_t T_{ij})^2},$$

where K is types of resources, w_{ik} is the weight of resource k in PM i where weight represents the priority of migration, x_{ijk} is the usage of resource k of VM j in PM i, r_{ik} is the largest percentage of occupancy of resource k in PM i, w_t is the weight of data exchanging rate, and T_{ij} is the communication rate of VM j with other VMs in PM i.

3.2.2 Bandwidth Guaranteed Method. From what we have discussed above, the algorithms considered scenarios where users continuously appear and require some (network or application) service at a specific time interval for a specific task. However, offloading can also be performed during extended time periods, for example, during an entire day. In this case, multiple edge servers could work cooperatively to serve customers in daytime, but the quantity of user requirements will dramatically decrease as the midnight approaches. When the load is low, the resources using the VMs are utilized inefficiently. Hence, the distributed active VMs can be migrated to one server while their original physical machines could be turned off to reduce total energy consumption. However, migration can also lead to new challenges. The VMs migration technology requires sufficient bandwidth to copy the current memory state to the destination server to initiate new VMs that resume the original service. At the destination server, existing VMs should keep the minimum bandwidth for the current users. Therefore, both the migration and the maintenance of current services share the same physical link. The bandwidth guaranteed method described in Reference [29] aims at solving such bandwidth competition to migrate VMs in the shortest time while maintaining the minimum bandwidth for user traffic. The migration time is defined as

$$T = \frac{M}{B_m - W},$$

where M is the size of memory used by VMs, B_m is total network bandwidth, and W is the current occupied network traffic. When $B_m \leq W$, migration is impossible. When $B_m > W$, the migration time depends on $B_m - W$.

In the proposed method, the order of the VM migrations is also considered. When the available bandwidth for VM migrations is abundant, the VM that has a large amount of state changes is migrated. Similarly, when the amount of available bandwidth is limited, the VM a small amount of state changes is migrated. The bandwidth guaranteed method has a very practical contribution in achieving a reduction of electric power consumption of the service provider, which may be widely deployed as an elastic scheme to perform cloud control automatically.

4 MOBILITY OF DEVICES

Mobility poses new challenges to the offloading algorithm designer. As a mobile user moves across different service areas, the device may leave the service coverage area of its original edge server. Such mobility will lead to two problems: First, we need to decide whether the edge service should be migrated out of the original server to a new server to keep the communication efficient. The migration deciding factor needs to resolve a tradeoff between the cost of long-distance communication and migration cost. Second, the network signal, e.g., in WiFi and 3G/4G/5G, may be affected by large objects data transfers, heterogeneous network environments, and the connection policies of smart devices, especially in the overlapped service areas. Persistent connectivity is not

guaranteed and the intermittent connection is possible. In this section, we will discuss four representative approaches to handle the mobility challenges from four aspects: (1) path selection for offloading traffic, (2) multi-tier edge computing architecture, (3) edge service migration, and (4) ad hoc cloudlets.

4.1 Path Selection for Offloading Traffic

The aim of path selection is to minimize the transmission delay and guarantee service continuity between users and edge nodes. The works [6, 32] proposed a handover mechanism based on the Manhattan distance considering the quality of communication links. The path selection is formulated as Markov Decision Process considering the transmission delay and energy caused by the transmission of offloading data. However, the path selection algorithms are not sufficient enough if the user device is too far away from the computing location since the transmission delay may reduce the QoS. In this case, the handover mechanisms to seek the new service server is necessary and more efficient. The authors of Reference [52] study how to balance the handover frequency and effectiveness to mobility management by performing handover decisions, which are formatted as a non-cooperative game model with the help of edge computing. The work [7] investigated an SDN-based method to decouple the mobility management and data forwarding functions. The SDN-based architecture was proposed, and a novel handover scheme is designed to improve the efficiency of the frequent handover between edge nodes and users caused by users' mobility. The SDN controller can pre-compute the optimal path by estimating the transmission delay of each possible option.

4.2 Offloading in Two-Tiered Mobile Cloud

To improve the performance while satisfying the SLA of mobile applications, Xia et al. [49] proposed a two-tiered mobile cloud architecture that contains both the edge clouds and center clouds. Even if the edge cloud has the advantages of low latency and high scalability, the capacity of an edge cloud may run into an over-utilization problem when too many users offload their workloads to the same edge cloud, which then could suffer from longer delays and heavier energy consumption (such peak-load situation may happen when the assemblies are held by big groups of people in public place). The proposed algorithm aims at offloading location-aware tasks of mobile applications to local or remote servers to ensure fairness of energy consumption that the battery life of each mobile device is prolonged equally. In this case, each device should consume the same portion of energy regarding its total energy capacity.

The two-tiered architecture supports an opportunistically flexible approach called Alg-MBM to help each mobile device choose the appropriate cloud server. The Alg-MBM constructs a weighted bipartite graph to find a weighted maximum matching offloading destinations including remote data centers, local edge servers, and even mobile device itself. It is worth noting that executing locally on the mobile device instead of offloading may be the best choice when the outside computation resources are heavily costly due to poor network conditions.

4.3 Follow Me Cloud

Follow Me Could (FMC) is a framework in which the mobile devices move across edge severs while the cloud service smoothly supports the user applications [38]. Because of the unpredictability of user mobility, VMs migration as the key technology for continuous cloud services breaks the limitation of geography. However, there are unresolved technical issues, since migrating VMs may have two restrictions: the latency of converting a VM to be ready for migration and the latency to transmit VM state over the network among edge servers. However, if the destination servers use

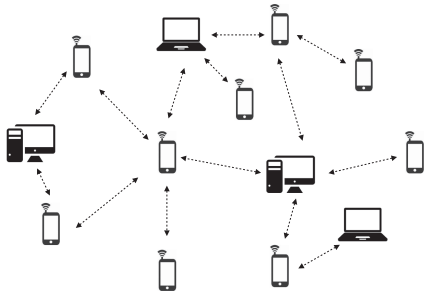

Fig. 6. Self-organized ad hoc mobile cloud.

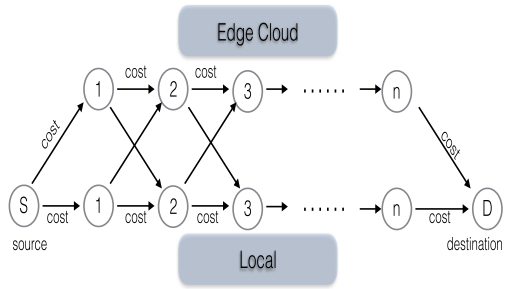

Fig. 7. Offloading choice control flow graph.

different hypervisor with the original server or the bandwidth is not qualified, then the service migration becomes more costly and may even be rejected.

An algorithm based on Markov Decision Process (MDP) is proposed to determine whether such migrations should be performed when the user device is at a given distance from the original server [22]. The authors defined a Continuous Time MDP (CTMDP) that contains the state of user devices and their transition probabilities and cost information. Moreover, they propose a Decision Time MDP (DTMDP) based on CTMDP with limited state spaces, which is regarded as a 1D model. Such a MDP is a static way to derive the optimal offloading since the migration cost function is pre-defined. Moreover, the finite state space will lead to high response time in solving the MDP.

To overcome the limitation of a static time cost calculation, a new dynamic service migration method is proposed to solve the restrictions of MDP by Wang et al. [43]. The authors considered a 2D mobility model that considers a MDP with arbitrarily larger state space. Two-dimensional mobility means that the user moves in a 2D space. Since both network topology and user mobility continuously change, the cost function and transition probabilities of MDP may fluctuate frequently. Therefore, the MDP should be solved in a dynamic manner. The 2D mobility algorithm can obtain an approximate solution by applying the distance-based MDP. Meanwhile, it decreases one order of magnitude of the overall complexity in each MDP iteration to improve time efficiency.

4.4 Ad Hoc Cloud-Assisted Offloading

Besides the two-tiered mobile cloud and FMC, mobile ad hoc networks are important applications that motivate researchers to pursue higher resource utilization and deal with user mobility. When the intermittent connectivity happens, a type of ad hoc self-organized mobile cloudlet helps mobile devices obtain close computation resources from other idle terminal devices, including smartphones, laptops, and desktop computers, to form a self-organized mobile cloud, as illustrated in Figure 6. As discussed in Section 2, Cloudlet provides such an ad hoc architecture that integrates ad hoc mobile device cloud with infrastructure-based edge servers [41, 42].

Based on such a mobile cloud at network edge, an up-to-date centralized task scheduling (CTS) algorithm was proposed by Wu et al. to guarantee SLA and achieve energy consumption balance [47]. As discussed in the algorithm, the execution of mobile application could be represented by a control flow graph that contains the computation components working in flow. A collaborative task execution scheme is implemented to determine where the components execute, in the local device or edge cloud, as illustrated in Figure 7. Aside from computation cost, the data transmission cost between edge and local is considered.

Under such a collaborative scheme, offloading is more flexible. When the infrastructure-based cloudlet is unavailable to execute assigned work tasks, the centralized task scheduler starts to seek the available mobile devices resource in the certain range by qualification judgement based

on the current usage of resource on the mobile cloud. The judgement process is formulated as a 0-1 integer linear programming problem and approximately solved by the greedy algorithm to get a solution and keep complexity to a minimum.

5 OFFLOADING PARTITION

Under the premise that the offloading achieves low latency, researchers make their best endeavors to prolong the battery life. Since the increases of battery capacity cannot catch up with the fast advances of program and application technologies like virtual reality and augmented reality, it is impossible to run all the parts of such applications only on the mobile device. The division and organization of partitioned components are the foundations to design an offloading algorithm. For some applications following the pipeline workflow such as VR streaming, the dependent relations between different functional components should be considered. For the applications with many human interactions such as voice control, the response latency is the most significant factor to guarantee the user experiments. Therefore, the algorithms of computation partitioning are further studied to determine which parts of the user application are offloaded and how they executed in order. The current partition strategies can be divided into three aspects: static, dynamic, and a combination of static and dynamic.

5.1 Static Partition

In the early years of research on cloud offloading, research studies were proposed on offloading computation of mobile devices to a close powerful server in the same LAN through a wireless connection. Li et al. [23] proposed a static partition approach based on the cost graph generated by the data of computing time and data sharing at the level of procedure calls. The cost graph statically divides the application program into server tasks and mobile device tasks to minimize energy consumption of handheld devices. Moreover, the program execution follows the order of sequential control flow. The data shared between two tasks is sent by the push-and-pull method, which guarantees that the server and client continuously update the most recent data modifications.

Based on such a scheme, the cost graph contains computation and energy information during the whole execution of sequential tasks. Then a Branch-and-Bound algorithm defines the offloading problem by linear expression to calculate optimal solution. However, the worst-case complexity of Branch-and-Bound is unacceptably costly according to the cost graphs of some applications. In this case, a pruning heuristic method is proposed to reduce the calculation time by only focusing on the components with heavy workload. The scheme is static, because all the profiled information is based on the intrinsic characteristics of the program, which leads to only one optimal solution. Figure 8 presents a flow graph used to the partitioning algorithms discussed above.

Besides the study on offloading partition from the standpoint of single application, Yang et al. [50] investigated the multi-user computation partitioning problem that jointly considered the task scheduling on the cloud servers side and the partition on the applications side. The applications are divided into task modules with the attributes including computation time of modules on devices or cloud servers and data transmission time while the computation resource on the cloud servers is constrained. The authors focused on latency-sensitive applications to minimize their average delay with a global view of all devices. The partitioning problem is solved by a static heuristic algorithm, SearchAdjust, by improving the performance of the classical list scheduling algorithms in the heterogeneous computingby 10% [40].

5.2 Dynamic Partition

While the static method considers all the parameters of the system and generates a globally optimal solution, the dynamic methods are more flexible, as they also evaluate network and server states.

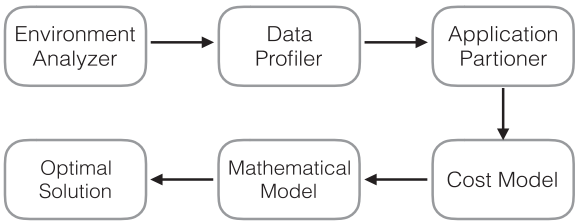

Fig. 8. Flow graph of offloading partition.

The Dynamical Offloading Algorithm (DOA) proposed by Huang et al. [19] focuses on achieving energy saving given the change of communication environment. Meanwhile, the interdependency of the partitioning application components should be considered because of the different execution latency constraints and data sharing cost with each other. In Reference [19], Huang et al. proposed a DOA to offload partitioned components with the change of wireless connection. They also created a cost graph where the vertices present application modules and the directed edges are data sizes from the source vertex to the destination vertex. In this case, the data transmission rates based on a wireless environment take dynamic effect on the decision of offloading, locally or remotely. The energy consumption and the total application execution time are formulated as a Lyapunov optimization problem that introduces a control parameter to take a tradeoff between energy and latency.

Besides the uncertainty of network connectivity, the dynamic partitioning for saving energy should take more heterogeneous factors into account [13]. Such factors include device operating systems, network types, cloud availability, and user workload. For example, an image matching program contains three stages: image feature generation, similarity calculation against a database, and classification. The content complexity of images are various, and the size of the matching database is changing so that the image processing may take a longer or shorter time during different stages. Given such an application, its device platform can be smart phones, tablets, or laptops with a wide variety of CPU, memory, and storage resources. The network is assumed to be a 3G/4G cellular network or a WiFi with different bandwidth. The cloud providers are assumed to have different prices and performance for their services, while the workload is dynamic at different stages. Additionally, another algorithm is proposed [53] that systematically discusses the factors affecting the performance of real-time video-based applications in dynamic wireless network conditions. The partitioning strategies should be dynamically adjusted during the execution of applications due to the changing of device and network status [51]. The initial partitioning at the beginning of execution may not hold for the whole process. In Reference [37], an algorithm based on dynamic programming with hamming distance terminations is proposed, which mainly focuses on available network bandwidth. Considering the workload of edge servers, the authors of Reference [50] utilized the proposed SearchAdjust static methods to deploy a dynamic solution in the practical system by performing the partitioning algorithms in every timeslot instead of one by one. The timeslot is small enough compared to the execution time of the offloaded tasks. This dynamic algorithm tries to maintain the balance between provisioning enough resource for current tasks and preparing for the future requests.

5.3 Combination of Static and Dynamic Partition

We could also combine the static and dynamic approaches to build a model for optimal offloading partition. Giurgiu et al. [17] propose a cost flow graph that is established based on the functional units and interdependency degree in terms of resources consumption such as data sharing, code size, and memory cost, similarly as in Figure 8.

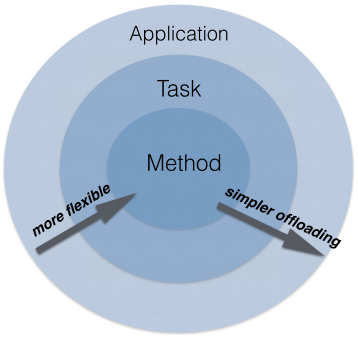

Fig. 9. Inclusion relationship of granularities.

Table 1. Offloading Granularity Comparison

Granularity	Advantages	Disadvantages
Application	1. thin workload on mobile devices 2. easy VM configuration on servers	relatively long time of VM initialization
Task	1. offer flexibility to developers 2. less synchronization work	low reuse posibility of execution environment
Method	wider authority to developers	1. high complexity to achieve optimal offloading 2. harder data synchronization 3. more fragmented user requests

Two partitioning algorithms in Reference [17] are proposed to derive static and dynamic optimization separately: ALL and K-step. ALL determines the best partitioning by evaluating all offline information of application and network. In addition, K-step performs partitioning of applications in real time when the mobile devices request services from cloud servers with their specific requirements. The algorithm starts estimating one node at a time by combining a depth-first and breadth-first method to the last node. Compared to ALL, K-step is faster, because it considers only a reduced set of configuration and less accurate. If there is a new configuration on nodes that offers a better solution, then a new local optimum will be updated.

6 PARTITIONING GRANULARITY

Before offloading computation to the edge servers, we must also consider the rational size of components that could run remotely. Given that different applications consist of the customized functional components designed by their developers, the partition granularity is a significant factor to improve the global execution performance. The granularity of partitioning is defined as the different sizes of offloading components. In this section, we classify the granularity of partitioning into three levels (from the biggest scale to smallest): Application, Task Module, and Method. Figure 9 offers an overview of their relationship. The more fine-grained the granularity, the more flexible and complex the offloading system is. Meanwhile, the comparison of advantages and disadvantages are summarized in Table 1.

6.1 Application Level

The computation components at the application-level contain the whole functionalities of software; this is the case, for example, for face recognition and voice translation applications. The partitioning algorithms at the application level impose a very thin workload to the mobile devices.

The corresponding software has already existed on the server that only needs to configure initialization data. Virtual machines (VMs), whose images are preinstalled on the servers, are most commonly utilized to meet the requirements at this granularity level. One of the advantages of application-level partitioning is an easier system configuration on the server side where the initiated VMs can be simply removed and new clean VMs can be ready for the next service timeslot. In this case, the mobile devices do not need to upload any functional parts, because the entire computation is executed on servers. Meanwhile, the less-intensive tasks, such as user interface and system data management, run on mobiles devices without a high energy cost. The Cloudlets [36, 41] we introduced in the previous sections adopt this application-level partitioning for offloading.

6.2 Task Module Level

The offloaded parts in the task module level are the application elements whose responsibilities are separated in a sequential or parallel order. A typical example is a face recognition program, which sequentially executes face detection, face verification, and face identification. As discussed in References [19] and [35], the cost model for task control flow, which is derived from the specific system features, displays the tasks partitioning at this granularity level. Such task modules generally execute in the containers of code running environment such as a Java virtual machine (Java Run Environment), a .Net framework, and a self-built running platform. Task module-level offloading is relatively more flexible than application-level partitioning, since the developer is allowed to make decisions to maximize performance using the edge cloud. Moreover, every task module is relatively enclosed, which means it receives input data from the previous stage and return its state to the next stage where there is not much synchronization work between the mobile devices and edge cloud. However, a task module as a medium granularity form puts forward more rigorous demand to the cloud servers, where the running environment faces more diverse requests from users. The environment configuration from user requests may ask for different classes of libraries, which leads to the lower possibility of container reuse. However, the programming languages available to developers are also relatively limited by running platforms.

6.3 Method Level

Method is in a lower level than task module for partitioning, which can also be presented in the form of functions as a code fragment. MAUI [15] and ThinkAir [21] require the developers to manually or semi-automatically annotate the methods as offloading permitted. The advantage of method level partitioning is that the developers have wider authority to improve their applications. However, such low-level granularity brings several challenges to achieve offloading optimization. First, the high complexity of obtaining an optimal solution may take longer because of a large number of methods. Second, the data synchronization between local devices and remote servers is harder to guarantee while they should share the same execution results. A synchronizing scheme runs periodically or in real time to collect and update the method data. Third, the service deployed on the server will handle more fragmented requests, which requires a robust identification mechanism to distinguish their source applications.

7 DISCUSSION AND PERSPECTIVES

After presenting the above research from five perspectives, in this section, we present some analysis and discussions on the mathematical models of existing algorithms, potential challenges, and technological trends for future offloading on an edge cloud.

Mathematical Models: We further summarize and compare the algorithms discussed in this article into Table 2. Besides the proposed five categories, we classify the related work also based on

Table 2. Characteristics of Offloading Algorithms

Algorithms	Destination	Balance	Mobility of Device	Partition	Granularity	Mathematic Model
MAUI[14]	Single server	Online	simply restart latest disconnected offloading	Dynamic	Method	Linear programming
CloneCloud[11]	Single server	Online	NS (not specified)	Dynamic	Method	Linear programming
ThinkAir[20]	Multiple servers	Online	NS	Dynamic	Method	Linear programming
Cloudlet[35]	Multiple servers	Online	NS	Dynamic	Application	Linear programming
Online-OBO and Online-Batch[47]	Single server	Online	NS	NS	Application	K-dimension bin packing
Primal-dual approach[17]	Multiple servers	Online	NS	NS	Application	Linear programming
Myopic Maxweight[45]	Multiple servers	Online	NS	NS	Application	Myopic MaxWeight algorithms with various routing policies
RIAL[10]	Multiple servers	Offline	NS	NS	Application	Multi-criteria decision making method
Bandwidth Guaranteed[28]	Single server	Offline	NS	NS	Application	Self-designed model
Alg-MBM[48]	Single server	Online	Center cloud and edge cloud work cooperatively in the two-tiered architecture	Dynamic	Task	Weighted maximum matching problem in the bipartite graph
1D MDP[21]	Multiple servers	Online	Cooperation among edge servers	Static	NS	Markov Decision Process
2D MDP[42]	Multiple servers	Online	Cooperation among edge servers	Static	NS	Markov Decision Process
Ad Hoc assisted CTS[46]	Multiple servers	Online	Cloudlets and ad hoc mobile devices	Dynamic	Task	0-1linear programming
Static partition scheme[22]	Single server	Online	NS	Static	Task	Branch and bound + pruning heuristic
ALL and K-Step[16]	Single server	Online	NS	Static + dynamic	Task	Self-designed model
DOA[18]	Single server	NS	Center cloud and edge cloud work cooperatively in the two-tiered architecture	Dynamic	Task	Lyapunov optimization
Joint optimization algorithm[34]	Multiple servers	NS	NS	Dynamic	Task	Convex optimization

the mathematical models and optimization methods utilized. Examples of such methods include 0-1 integer linear programming problem, K-dimensional bin parking, a Markov decision process, and Lyapunov optimization. Since most of these optimization models attempt to solve an NP-hard problem, the approximation solutions with higher performance and lower complexity are designed and evaluated in many cases. When researchers try to include more factors or constraints in their offloading algorithms to meet specific performance requirements, the algorithms need to be improved to be more flexible. Moreover, the offloading strategies should be dynamically adjusted with the change of the edge server's workload during the offloading process. The server could increase the resource provisioning to the device to speed up the application execution. For example, a hierarchical edge cloud architecture was proposed to solve the offloading problem during peak hours [39]. A dynamic voltage scaling technique was implemented to vary the energy supplement of mobile devices based on the computation loads [45]. To address the potential requirements of future applications, the existing algorithms should be adjusted, and new approaches need to be explored.

User Mobility: The mobility influences the offloading decision, since the change of device location and distance from edge servers may cause a decrease of communication quality and an increase of device energy consumption. The existing works focus on the user mobility from the aspects of path selection, multi-tier edge cloud, edge service migration, and ad hoc cloudlets. Moreover, the prediction of future trajectory should be further studied to achieve the pre-migration of the VM to reduce the migration delay and service interruption. However, the current migration studies mostly assume that a single VM provide computing service to one user and how to perform the migration when the computation of one user is offloaded to multiple servers. Therefore, new advanced methods should be developed to meet the latency requirement for the user devices, even for real-time application.

EC Offloading for Future IoT Environment: The Internet of Things is estimated to bring the next major economic and societal revolution by turning billions of independent electric devices into an enormous interconnected community where the data are shared more frequently and faster than ever before [30]. According to the forecast of IHS and Mckinsey [20, 25], devices connected to the IoT network will increase from 15.4 billion in 2015 to 30.7 billion in 2020. We can imagine that the future society will be boosted by seamless intelligent cooperation among smart devices, and the creation of smart-x applications such as smart health, smart home, smart city, smart energy, smart transport, and smart farming and food security.

Under the background of IoT, edge computing can potentially support significant progress to solve problems, including communication latency, energy saving, user mobility, the variety of personalized applications, support of real-time applications, and network heterogeneity. However, the research in this aspect is not mature enough yet to accommodate various IoT standards, and the current work still covers only a limited range of application scenarios. Many smart-x applications have not yet adopted corresponding customized cloud computing models very effectively. Therefore, the future research can focus on implementing a specific class of IoT systems and related algorithms.

EC Offloading for Big Data: With the development of Big Data technologies, media transmission and targeting delivery of customized content can improve the service efficiency and accuracy for mobile users. In conjunction with edge cloud computing, applying Big Data techniques, researchers can improve the performance of data processing such as collecting, capturing, analyzing, searching, exchanging, transmiting, and protecting. e.g, a cloud platform for the medical education to measure patients' personal big data [5]. By offloading heavy workloads to edge servers, both computation and communication latency are guaranteed to extract valuable information from data. Facilitated by edge clouds, the application of Big Data could be conveniently accessed by terminal

users. In this case, how to maximize the advantages of EC by effective offloading approaches to boost Big Data technology is still an unexploited field.

EC Offloading for 5G: In the future, 5G will bring us broader network bandwidth and greater convenience of device connectivity that allows a larger number of mobile users per area unit. The mobile devices are also provided with high availability and speeds of network access. However, the spectrum resource is limited, which could bear a heavier burden in the 5G era. It may increase costs when cloud resources are accessed through the 5G network. Currently, there are some related researches of offloading in such situations. For example, a time-adaptive heuristic algorithm with multiple radio access technology (Multi-RAT) is proposed [28]. A distributed computation offloading algorithm solves the offloading decision-making problem in a wireless network with multi-channel to avoid mutual interference [24]. To improve the performance of offloading in a 5G network, EC can play an important role to enhance its upper bound.

8 CONCLUSIONS

In this article, we collected and investigated the key issues, methods, and various state-of-the-art efforts related to the offloading problem in the edge cloud framework. We adopted a new characterizing model to study the whole process of offloading from mobile devices to the edge cloud, which consists of the basic categorizing criteria of offloading destination, load balance, mobility, partitioning, and granularity. The overall goal of offloading is to achieve low latency and better energy efficiency at each step of computation offloading. An integrated offloading system of an edge cloud should be a well-balanced combination of these five perspectives to properly solve offloading issues. The factors of algorithms such as environment constraints, cost models, user configuration, and mathematical principles were discussed in detail. We endeavored to draw an overall "big picture" for the existing efforts. Embracing future network development, we plan to continuously explore emerging technologies and creative ideas that improve offloading performance.

REFERENCES

[1] 2018. Amazon Web Service. Retrieved from https://aws.amazon.com.

[2] 2018. Google Cloud. Retrieved from https://cloud.google.com/.

[3] 2018. Microsoft Azure. Retrieved from https://azure.microsoft.com.

[4] A. Ahmed and E. Ahmed. 2016. A survey on mobile edge computing. In *Proceedings of the 2016 10th International Conference on Intelligent Systems and Control (ISCO'16)*. 1–8. DOI: https://doi.org/10.1109/ISCO.2016.7727082

[5] Maqbool Ali, Hafiz Syed Muhammad Bilal, Muhammad Asif Razzaq, Jawad Khan, Sungyoung Lee, Muhammad Idris, Mohammad Aazam, Taebong Choi, Soyeon Caren Han, and Byeong Ho Kang. 2017. IoTFLiP: IoT-based flipped learning platform for medical education. *Digital Communications and Networks* 3, 3 (2017), 188–194. DOI: https://doi.org/10.1016/j.dcan.2017.03.002

[6] Z. Becvar, J. Plachy, and P. Mach. 2014. Path selection using handover in mobile networks with cloud-enabled small cells. In *Proceedings of the 2014 IEEE 25th Annual International Symposium on Personal, Indoor, and Mobile Radio Communication (PIMRC'14)*. 1480–1485. DOI: https://doi.org/10.1109/PIMRC.2014.7136402

[7] Y. Bi, G. Han, C. Lin, Q. Deng, L. Guo, and F. Li. 2018. Mobility support for fog computing: An SDN approach. *IEEE Commun. Mag.* 56, 5 (May 2018), 53–59. DOI: https://doi.org/10.1109/MCOM.2018.1700908

[8] L. F. Bittencourt, M. M. Lopes, I. Petri, and O. F. Rana. 2015. Towards virtual machine migration in fog computing. In *Proceedings of the 2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC'15)*. 1–8. DOI: https://doi.org/10.1109/3PGCIC.2015.85

[9] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. 2012. Fog computing and its role in the internet of things. In *Proceedings of the 1st Edition of the MCC Workshop on Mobile Cloud Computing*. ACM, 13–16.

[10] H. Chang, A. Hari, S. Mukherjee, and T. V. Lakshman. 2014. Bringing the cloud to the edge. In *Proceedings of the 2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS'14)*. 346–351. DOI: https://doi.org/10.1109/INFCOMW.2014.6849256

[11] L. Chen, H. Shen, and K. Sapra. 2014. RIAL: Resource intensity aware load balancing in clouds. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM'14)*. 1294–1302. DOI: https://doi.org/10.1109/INFOCOM.2014.6848062

[12] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. 2011. CloneCloud: Elastic execution between mobile device and cloud. In *Proceedings of the 6th Conference on Computer Systems (EuroSys'11)*. ACM, New York, NY, 301–314. DOI: https://doi.org/10.1145/1966445.1966473

[13] Byung-Gon Chun and Petros Maniatis. 2010. Dynamically partitioning applications between weak devices and clouds. In *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond (MCS'10)*. ACM, New York, NY. DOI: https://doi.org/10.1145/1810931.1810938

[14] S. S. Cross, T. Dennis, and R. D. Start. 2002. Telepathology: Current status and future prospects in diagnostic histopathology. *Histopathology* 41, 2 (2002), 91–109. DOI: https://doi.org/10.1046/j.1365-2559.2002.01423.x

[15] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. 2010. MAUI: Making smartphones last longer with code offload. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services (MobiSys'10)*. ACM, New York, NY, 49–62. DOI: https://doi.org/10.1145/1814433.1814441

[16] R. Gargees, B. Morago, R. Pelapur, D. Chemodanov, P. Calyam, Z. Oraibi, Y. Duan, G. Seetharaman, and K. Palaniappan. 2017. Incident-supporting visual cloud computing utilizing software-defined networking. *IEEE Trans. Circ. Syst. Vid. Technol.* 27, 1 (Jan. 2017), 182–197. DOI: https://doi.org/10.1109/TCSVT.2016.2564898

[17] Ioana Giurgiu, Oriana Riva, Dejan Juric, Ivan Krivulev, and Gustavo Alonso. 2009. *Calling the Cloud: Enabling Mobile Phones as Interfaces to Cloud Applications*. Springer, Berlin, 83–102. DOI: https://doi.org/10.1007/978-3-642-10445-9_5

[18] F. Hao, M. Kodialam, T. V. Lakshman, and S. Mukherjee. 2017. Online allocation of virtual machines in a distributed cloud. *IEEE/ACM Transactions on Networking* 25, 1 (Feb 2017), 238–249. DOI: https://doi.org/10.1109/TNET.2016.2575779

[19] D. Huang, P. Wang, and D. Niyato. 2012. A dynamic offloading algorithm for mobile computing. *IEEE Trans. Wireless Commun.* 11, 6 (Jun. 2012), 1991–1995. DOI: https://doi.org/10.1109/TWC.2012.041912.110912

[20] Chris Ip. 2016 June 21. The IoT opportunity—Are you ready to capture a once-in-a lifetime value pool? Retrieved from http://hk-iot-conference.gs1hk.org/2016.

[21] S. Kosta, A. Aucinas, Pan Hui, R. Mortier, and Xinwen Zhang. 2012. ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *Proceedings of the 2012 IEEE International Conference on Computer Communications (INFOCOM'12)*. 945–953. DOI: https://doi.org/10.1109/INFCOM.2012.6195845

[22] A. Ksentini, T. Taleb, and M. Chen. 2014. A markov decision process-based service migration procedure for follow me cloud. In *Proceedings of the 2014 IEEE International Conference on Communications (ICC'14)*. 1350–1354. DOI: https://doi.org/10.1109/ICC.2014.6883509

[23] Zhiyuan Li, Cheng Wang, and Rong Xu. 2001. Computation offloading to save energy on handheld devices: A partition scheme. In *Proceedings of the 2001 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES'01)*. ACM, New York, NY, 238–246. DOI: https://doi.org/10.1145/502217.502257

[24] Yujiong Liu, Shangguang Wang, and Fangchun Yang. 2016. A multi-user computation offloading algorithm based on game theory in mobile cloud computing. In *Proceedings of the IEEE/ACM Symposium on Edge Computing (SEC'16)*. IEEE, 93–94.

[25] Sam Lucero. 2016 March. IoT platforms: Enabling the Internet of Things. Retrieved from https://cdn.ihs.com/www/pdf/enabling-IOT.pdf.

[26] P. Mach and Z. Becvar. 2017. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Commun. Surv. Tutor.* 19, 3 (2017), 1628–1656. DOI: https://doi.org/10.1109/COMST.2017.2682318

[27] S. T. Maguluri, R. Srikant, and L. Ying. 2012. Stochastic models of load balancing and scheduling in cloud computing clusters. In *2012 Proceedings IEEE International Conference on Computer Communications (INFOCOM'12)*. 702–710. DOI: https://doi.org/10.1109/INFCOM.2012.6195815

[28] S. E. Mahmoodi and K. P. S. Subbalakshmi. 2016. A time-adaptive heuristic for cognitive cloud offloading in multi-RAT enabled wireless devices. *IEEE Trans. Cogn. Commun. Netw.* 2, 2 (Jun. 2016), 194–207. DOI: https://doi.org/10.1109/TCCN.2016.2588508

[29] K. Mochizuki, H. Yamazaki, and A. Misawa. 2013. Bandwidth guaranteed method to relocate virtual machines for edge cloud architecture. In *Proceedings of the 2013 15th Asia-Pacific Network Operations and Management Symposium (APNOMS'13)*. 1–6.

[30] Vermesan Ovidiu and Friess Peter. 2015. Building the Hyperconnected Society. Retrieved from http://www.internet-of-things-research.eu/pdf/Building_the_Hyperconnected_Society_IERC_2015_Cluster_eBook_978-87-93237-98-8_P_Web.pdf.

[31] J. Pan, L. Ma, R. Ravindran, and P. TalebiFard. 2016. HomeCloud: An edge cloud framework and testbed for new application delivery. In *Proceedings of the 2016 23rd International Conference on Telecommunications (ICT'16)*. 1–6. DOI: https://doi.org/10.1109/ICT.2016.7500391

[32] Jan Plachy, Zdenek Becvar, and Pavel Mach. 2016. Path selection enabling user mobility and efficient distribution of data for computation at the edge of mobile network. *Comput. Netw.* 108 (2016), 357–370. DOI: https://doi.org/10.1016/j.comnet.2016.09.005

[33] European Telecommunications Standards Institute (ETSI). 2016. Mobile-Edge Computing Initiative. Retrieved from http://www.etsi.org/technologies-clusters/technologies/mobile-edge-computing.

[34] Seewon Ryu. 2012. Telemedicine: Opportunities and developments in member states: Report on the second global survey on eHealth 2009 (global observatory for eHealth series, volume 2). *Healthcare Inf. Res.* 18, 2 (2012), 153–155.

[35] S. Sardellitti, G. Scutari, and S. Barbarossa. 2015. Joint optimization of radio and computational resources for multicell mobile-edge computing. *IEEE Trans. Sign. Inf. Process. Netw.* 1, 2 (Jun. 2015), 89–103. DOI: https://doi.org/10.1109/TSIPN.2015.2448520

[36] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. 2009. The case for VM-based cloudlets in mobile computing. *IEEE Perv. Comput.* 8, 4 (Oct. 2009), 14–23. DOI: https://doi.org/10.1109/MPRV.2009.82

[37] H. Shahzad and T. H. Szymanski. 2016. A dynamic programming offloading algorithm for mobile cloud computing. In *Proceedings of the 2016 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE'16)*. 1–5. DOI: https://doi.org/10.1109/CCECE.2016.7726790

[38] T. Taleb and A. Ksentini. 2013. An analytical model for follow me cloud. In *Proceedings of the 2013 IEEE Global Communications Conference (GLOBECOM'13)*. 1291–1296. DOI: https://doi.org/10.1109/GLOCOM.2013.6831252

[39] L. Tong, Y. Li, and W. Gao. 2016. A hierarchical edge cloud architecture for mobile computing. In *Proceedings of the The 35th Annual IEEE International Conference on Computer Communications (IEEE INFOCOM'16)*. 1–9. DOI: https://doi.org/10.1109/INFOCOM.2016.7524340

[40] H. Topcuoglu, S. Hariri, and Min-You Wu. 2002. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parallel Distrib. Syst.* 13, 3 (Mar. 2002), 260–274. DOI: https://doi.org/10.1109/71.993206

[41] Tim Verbelen, Pieter Simoens, Filip De Turck, and Bart Dhoedt. 2012. Cloudlets: Bringing the cloud to the mobile user. In *Proceedings of the 3rd ACM Workshop on Mobile Cloud Computing and Services (MCS'12)*. ACM, New York, NY, 29–36. DOI: https://doi.org/10.1145/2307849.2307858

[42] T. Verbelen, P. Simoens, F. De Turck, and B. Dhoedt. 2013. Leveraging cloudlets for immersive collaborative applications. *IEEE Perv. Comput.* 12, 4 (Oct. 2013), 30–38. DOI: https://doi.org/10.1109/MPRV.2013.66

[43] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung. 2015. Dynamic service migration in mobile edge-clouds. In *Proceedings of the 2015 IFIP Networking Conference (IFIP Networking'15)*. 1–9. DOI: https://doi.org/10.1109/IFIPNetworking.2015.7145316

[44] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang. 2017. A survey on mobile edge networks: Convergence of computing, caching and communications. *IEEE Access* 5 (2017), 6757–6779. DOI: https://doi.org/10.1109/ACCESS.2017.2685434

[45] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li. 2016. Mobile-edge computing: Partial computation offloading using dynamic voltage scaling. *IEEE Trans. Commun.* 64, 10 (Oct. 2016), 4268–4282. DOI: https://doi.org/10.1109/TCOMM.2016.2599530

[46] Douglas B. West. 2000. *Introduction to Graph Theory* (2nd ed.). Prentice Hall.

[47] Z. Wu, L. Gui, J. Chen, H. Zhou, and F. Hou. 2016. Mobile cloudlet assisted computation offloading in heterogeneous mobile cloud. In *Proceedings of the 2016 8th International Conference on Wireless Communications Signal Processing (WCSP'16)*. 1–6. DOI: https://doi.org/10.1109/WCSP.2016.7752655

[48] Q. Xia, W. Liang, and W. Xu. 2013. Throughput maximization for online request admissions in mobile cloudlets. In *Proceedings of the 38th Annual IEEE Conference on Local Computer Networks*. 589–596. DOI: https://doi.org/10.1109/LCN.2013.6761295

[49] Q. Xia, W. Liang, Z. Xu, and B. Zhou. 2014. Online algorithms for location-aware task offloading in two-tiered mobile cloud environments. In *Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*. 109–116. DOI: https://doi.org/10.1109/UCC.2014.19

[50] L. Yang, J. Cao, H. Cheng, and Y. Ji. 2015. Multi-user computation partitioning for latency sensitive mobile cloud applications. *IEEE Trans. Comput.* 64, 8 (Aug. 2015), 2253–2266. DOI: https://doi.org/10.1109/TC.2014.2366735

[51] L. Yang, J. Cao, S. Tang, D. Han, and N. Suri. 2016. Run time application repartitioning in dynamic mobile cloud environments. *IEEE Trans. Cloud Comput.* 4, 3 (Jul. 2016), 336–348. DOI: https://doi.org/10.1109/TCC.2014.2358239

[52] H. Zhang, Y. Qiu, X. Chu, K. Long, and V. C. M. Leung. 2017. Fog radio access networks: Mobility management, interference mitigation, and resource optimization. *IEEE Wireless Commun.* 24, 6 (Dec. 2017), 120–127. DOI: https://doi.org/10.1109/MWC.2017.1700007

[53] L. Zhang, D. Fu, J. Liu, E. C. H. Ngai, and W. Zhu. 2017. On energy-efficient offloading in mobile cloud for real-time video applications. *IEEE Trans. Circ. Syst. Vid. Technol.* 27, 1 (Jan. 2017), 170–181. DOI: https://doi.org/10.1109/TCSVT.2016.2539690

Received May 2017; revised July 2018; accepted October 2018