Edge Cloud Offloading Algorithms: Issues, Methods, and Perspectives

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

Mobile devices supporting the "Internet of Things" (IoT), often have limited capabilities in computation, battery energy, and storage space, especially to support resource-intensive applications involving virtual reality (VR), augmented reality (AR), multimedia delivery and artificial intelligence (AI), which could require broad bandwidth, low response latency and large computational power. Edge cloud or edge computing is an emerging topic and technology that can tackle the deficiency of the currently centralized-only cloud computing model and move the computation and storage resource 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 the mobile devices and the edge cloud to work together smoothly. In this survey paper, 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 edge cloud have demonstrated profound potentials for future technology and application development.

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

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

1 INTRODUCTION

We are embracing the future world of 5G communication and Internet of Things (IoT). Smart mobile devices are becoming more popular and playing increasingly important roles in every aspect of our daily life [1]. Various applications running on these mobile devices require not only bounded latency, wide bandwidth and high computation performance, but also long battery life [2]. In these cases, mobile devices alone are insufficient as they suffer from limited local capabilities of computation and energy to deliver performant resources-intensive applications. Therefore, such resource gap is fulfilled by the remote and centralized data centers or clouds services such as Amazon Web Services [3], Microsoft Azure [4] and Google Cloud [5]. These centralized clouds (CCs) can offer virtually unlimited computation, networking and storage resources. For many years,

First manuscript: April 30th, 2017.
Author's addresses: J. Wang and J. Pan , the Department of Mathematics and Computer Science, University of Missouri-St. Louis, MO 63121 USA; email: jwgxc@umsl.edu, pan@umsl.edu; F. Esposito, the Department of Computer Science, Saint Louis University, MO 63103 USA.; P. Calyam, the Department of Computer Science, University of Missouri-Columbia, MO 65211 USA; Z. Yang and P. Mohapatra, the Department of Computer Science, University of California, Davis, CA 95616 USA; .
2018. XXXX-XXXX/2018/6-ART1 $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

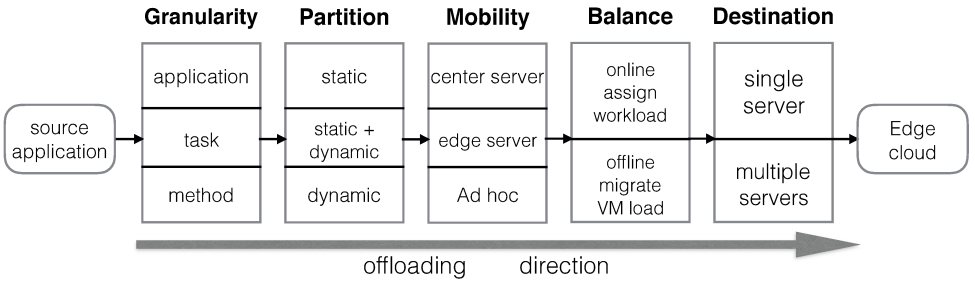

Fig. 1. The whole process of offloading from mobile devices to an edge cloud.

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" [6] or "Mobile Edge Computing" [7] in some literatures) 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 [8] [9], 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 response in emergency to save lives. Finally, with the advent of IoT, ECs can play a key role to build a fundamental tier of IoT systems for many more distributed applications in smart home, smart health, smart vehicles and even smart cities [1]. Recent research such as HomeCloud framework [10] and Incident-Supporting Visual Cloud [11] 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. Our paper 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 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 Fig. 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 paper stands for a typical and creative research direction.

Several other survey papers related to edge cloud [12][13][14] discuss edge cloud computing in different domains (such as edge computing architecture, communication, computation offloading and use case studies). However, our paper is different with them in collecting offloading algorithms and analyze their mathematical models from a holistic comprehensive perspective.

The rest of this paper is organized as follow. Section 2 introduces scenarios of the single and multiple servers as offloading destination. 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 its 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 in order 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 mobile devices at the run time could be offloaded to only one server for sequential execution or to multiple servers for parallel execution, leading to a lower response latency. Fig. 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 cannot is insufficient to sustain the workload in peak periods, other EC servers or CC servers are made available to assist the application. Current studies on offloading focus on a wide range of applications, where the requirements of network and computing resources vary in different execution environments. 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. On the other hand, 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 [16] and CloneCloud [17] 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 [16]. The remote server could be a CC server or a nearby EC server at WiFi access point. As a pioneer of all offloading systems, the MAUI offloading strategy takes

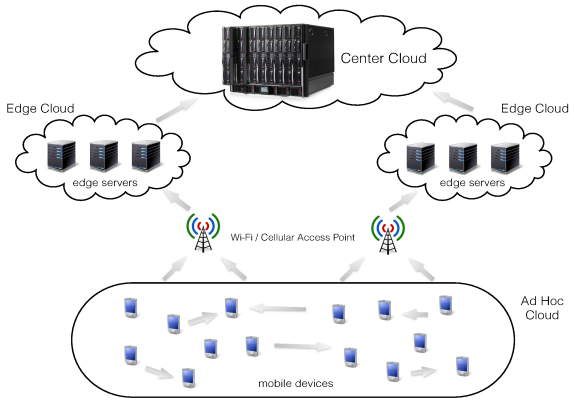

Fig. 2. Mobile cloud computing framework.

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 Fig. 3. The proxy is used to transmit data and control instructions. The profiler retrieves the data about program requirement, execution energy cost and 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 which 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 follow:

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

constraints:

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

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, 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, then MAUI automatically decides which methods should be offloaded with the aid of programming reflecting and type-safety to manage program behavior. Second, at run time, the profiler start 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 start 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.

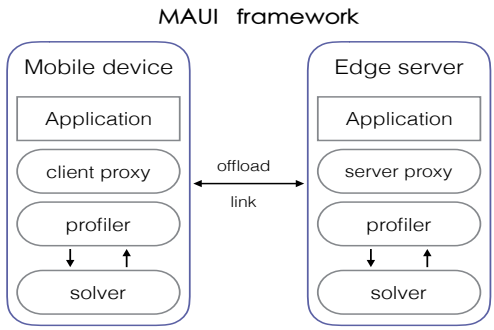

Fig. 3. MAUI system model.

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 experimented: resource-intensive face recognition, latency-sensitive video game, and voice language translation. With their implementation of MAUI, authors demonstrate a 27% energy of the smartphone is saved for video game, 45% for 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. Similar 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 [17]. 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, 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 which 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, it should not be in the callee. Such nested migration is prohibited to avoid triggerings multiple times 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 run time 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, application state is repackaged and shipped back to the original mobile device, 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 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 camera, GPS and sensors. Second, when the offloading strategies try to permit perfunctory concurrency between unoffloaded and offloaded workload, 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, 3D modeling of disaster site and unmanned driving. In these cases, the tolerance of execution latency is relatively rigorous to meet customer requirements. In these cases, and in others where 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 featuring parallel offloading on multiple servers have been proposed to distribute partitioned tasks to a cluster of servers which have different capacities of computation and communication resources. In the next subsections we discuss a few representative solutions that adopt this model.

2.2.1 ThinkAir. After the publication of MAUI and CloneCloud, ThinkAir [18]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 which runs 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 the mobile applications such as face recognization. The system can divide a calculation problem into sub-problems to execute on the multiple VMs to reduce the waiting interval of the tasks between cloud and mobile devices. Seconds, 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.

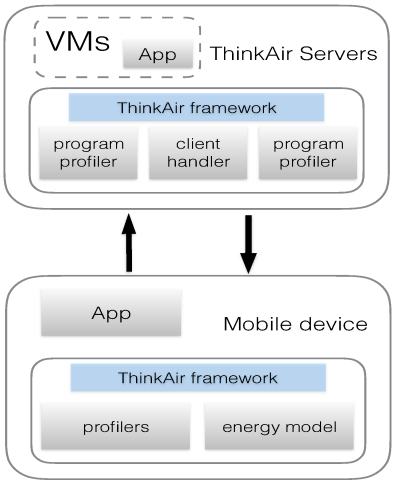

Fig. 4. Functional components in ThinkAir framework.

Similar to MAUI, edge cloud application developers using ThinkAir would still need to modify the code for running application smoothly. However, such modification workload requirement 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 data 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.

Fig. 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 which 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 which is the next hop to be connected to the Internet. The concept of cloudlet was initially proposed by *Satyanarayanan* [19]. 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.* [20] pointed out two drawbacks of this VM based cloudlet strategy. First, the cloudlets are provided by Internet Service Providers in their LAN (local area network) 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, authors in [20] further proposed an elastic architecture of cloudlets. Their architecture not only provides original cloudlet servers at the edge but also organize 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 which 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. In some cases, the edge cloud may not have enough available resources making it hard to meet the Service Level Agreements (SLA) of the application; this means that not all user or application requests may be accepted. In these cases, the uncertainty on user request satisfaction and the weakness of offloading strategies may lead to unbalanced loads among cloud servers, where some parts of them may work with low loads while others are almost fully utilized. Under these scenarios, we classify the literature into two types of algorithms: online and offline. Fig. 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 work based on such system.

3.1 Online Balancing

Online balancing is a pre-processing or run-time method that ensures that the workloads are distributed to the appropriate servers when the user requests arrive at cloud in real time. Online

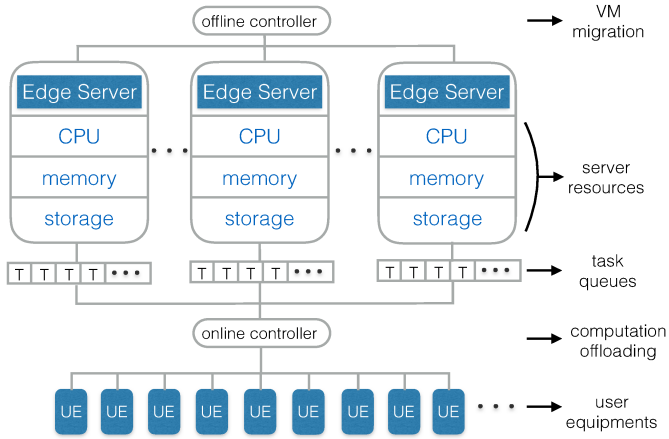

Fig. 5. The framework of balancing control system.

algorithms consider both user configuration requirements and current server's 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.* [21] 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 if 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 paper [21]: 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 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 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. The details of the Online-OBO algorithm are shown in the Algorithm 1.

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.* [22] proposed an algorithm for allocating VMs in a distributed cloud which consists of geographically diversified small data centers

Algorithm 1 Online-OBO balance [21]

Input: a request i at time t $q_i(t)$, available amount of resources $H(t)$, threshold of unit addmission cost of type n resource T_n (their average value T_{avg}) , unit admission cost $C_i(t)$, where $q_i(t) := \{q_{i,1}(t), ..., q_{i,N}(t)\}$, $\forall 1 \leq n \leq N$.
Output: accept or deny request $q_i(t)$

for each $q_{i,n}(t)$ in $q_i(t)$ **do**
  if $q_{i,n}(t) > H_n(t)$ **then**
    deny request $q_i(t)$;
    EXIT;
  else if $C_{i,n}(t) > T_n$ **then**
    deny request $q_i(t)$;
    EXIT;
  else
    update $H_n(t)$;
  end if
end for
if $C_i(t) \leq T_{avg}$ **then**
  update $H(t)$;
  return accept $q_i(t)$;
  deny request $q_i(t)$;
  EXIT;
end if

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 a NP-hard problem which 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 [23].

The authors studied two popular algorithms, pointed out their disadvantages, and improved the MaxWeight method [24]. 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 [24] 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 paper 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, as introduced in Algorithm 2. 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 compare it with the one allocated in the last time slot. All the above algorithms provide us 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.

3.2.1 Resource Intensity Aware Load (RIAL). 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 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 [25].

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. On the other hand, the VMs which 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, seek and migrate 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

Algorithm 2 Myopic MaxWeight Scheduling based on Stochastic Models [24]

Input: real-time requests; task queues
Output: stabilized queues; usage percentage of resources in servers

Join Shortest Queue:
A job with resource type m arrives in the system at time slot t, which is inserted in the shortest queue. The chosen server i is with type m jobs queue $Q_{mi}(t)$.

Myopic MaxWeight Scheduling:
Step 1: Generate a super slot with a set of n continuous time slots, represented by T, where $T = nt$

Step 2: If the request arrives at the start of a super slot T. The optimal configuration is

$$C_i(t) \in arg\ max \sum_m Q_{mi}(t)N_{mi}, N \in \mathbf{N}_i$$

Otherwise, the configure is chosen

$$C_i(t) \in arg\ max \sum_m Q_{mi}(t)N_{mi}, N \in \mathbf{N}_i + N_i(t^-)$$

where N_{mi} is number of VMs with resource type$m$, $\mathbf{N}_i$ is the set of configuration in server i, $N_i(t^-)$ is VMs hosted at the beginning of super slot T.
Step 3: Update the queue length information.

$$Q_{mi}(t+1) = Q_{mi}(t) + W_{mi}(t) - N_{mi}(t)$$

where $W_{mi}(t)$ is the requests at time t.

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 least data transmission with other VMs.

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

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

where K is types of resources, w_{ik} is the weight of resource k in PM i where weight represent 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, 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 be performed also 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 in order 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 [26] aims at solving such bandwidth competition to migrate VMs in a 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. 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 with small amount of state changes is migrated. The bandwidth guaranteed method has very practical contribution in achieving reduction of electric power consumption of 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 if 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 three representative approaches that handle the mobility problems in different environments under the edge cloud infrastructure.

4.1 Offloading in Two-Tiered Mobile Cloud

To improve the performance while satisfying the SLA of mobile applications, *Xia et al.* [27] 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 edge cloud may run into over-utilized problem when too many users offload their workloads to the same edge cloud which could suffer from longer delay 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 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. The details of Alg-MBM algorithm are shown in Algorithm 3.

Algorithm 3 Location-aware Alg-MBM [27]

Input: user requests $r(t)$ that arrives at time slot t, a local cloudlet C_e, a remote cloud center C_c
Output: maximize the battery life of each mobile devices to extend its work time

1: Step 1: Collect the requirements of the requests $r(t)$ including resource consumption $R(t)$ and device location $L(t)$.
2: Step 2: Generate a bipartite graph $G(t) = (R(t), L(t))$ for requests arrived at time slot t, in which the weight of edges represents the energy cost.
3: Step 3: Find the optimal path with least edge weight in total to complete all the request at time t.
4: Step 4: Execute workload in mobile devices, or offload to C_e or C_c according to the path in the previous step.

4.2 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 [28]. 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. On the other hand, if the destination servers use different hypervisor with the original server or the bandwidth is not qualified, 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 [29]. The authors defined a Continuous Time MDP (CTMDP) that contains the state of user devices, 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 one-dimension model (1-D). 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 leads 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 limitations of MDP by *Wang et al.* [30]. The authors considered a two-dimensional (2-D) mobility model that considers a MDP with arbitrarily larger state space. 2-D mobility means that the user moves in a 2-D space. Since both network topology and user mobility continuously changes, practically the cost function and transition probabilities of MDP may fluctuate frequently. Therefore, the MDP should be solved in a dynamic manner. The 2-D 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.

4.3 Ad Hoc Cloud Assisted Offloading

Besides the two-tiered mobile cloud and FMC (Follow Me Cloud), mobile ad hoc networks are another important applications that motivates researchers to pursue higher resources utilization

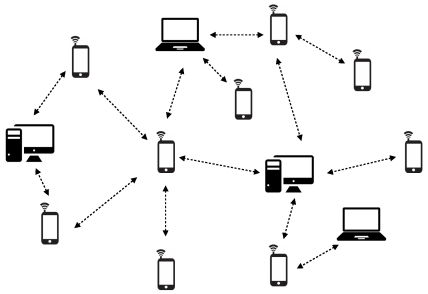

Fig. 6. Self-organized ad-hoc mobile cloud.

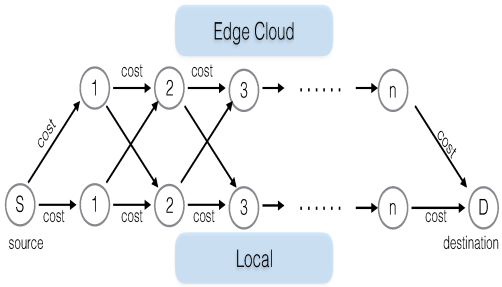

Fig. 7. Offloading choice control flow graph.

and deal with user mobility. When the intermittent connectivity happens, a type of ad hoc self-organized mobile cloudlets help 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 Fig. 6. As discussed in the Section 2, Cloudlet provides such an ad hoc architecture which integrates ad hoc mobile device cloud with infrastructure-based edge servers [20] [31].

Based on such a mobile cloud at network edge, an up-to-dated centralized task scheduling (CTS) algorithm was proposed by *Wu et al.* to guarantee SLA and achieve energy consumption balance [32]. As discussed in the algorithm, the execution of mobile application could be represented by a control flow graph which contains the computation components working in flow. A collaborative task execution scheme is implemented to determine where the components execute, local device or edge cloud, as illustrated in Fig. 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 low complexity.

5 OFFLOADING PARTITION

Under the premise that the offloading achieves low latency, researchers make their best endeavors to prolong the battery life. Since the increment of battery capacity cannot catch up the fast advance of program and application technologies like virtual reality and augment 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. 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 to 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 wireless connection. *Li et al.* [33] 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

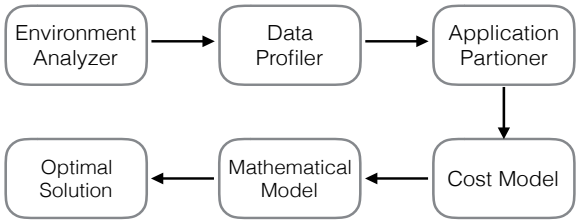

Fig. 8. Flow graph of offloading partition.

statically divides the application program into server tasks and mobile device tasks to minimize energy consumption. 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 guaranteed 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.

This scheme is static because all the profiled information is based on the intrinsic characteristics of the program which leads to only one optimal solution. Fig. 8 presents a flow graph used to the partitioning algorithms discussed above.

5.2 Dynamic Partition

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

The Dynamical Offloading Algorithm (DOA) proposed by *Huang et al.* [34] 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 [34], *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 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 which 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 [35]. Such factors include device operating system, 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.

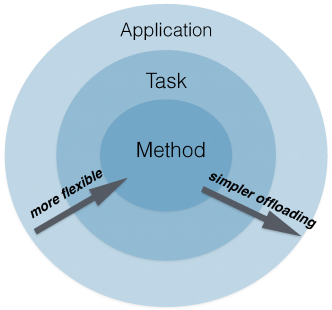

Fig. 9. Inclusion relationship of granularities.

Additionally, another algorithm is proposed [36] to systematically discusses the factors affecting the performance of real-time video based applications in dynamic wireless network condition. In [37], an algorithm based on a dynamic programming with hamming distance terminations is proposed, which mainly focuses on available network bandwidth.

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.* [38] propose a cost flow graph which is established based on the functional units and interdependency degree in terms of resources consumption such as data sharing, code size and memory cost, similar to Fig. 8.

Two partitioning algorithms in [38], 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 depth-first and breadth-first method till 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, 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. Fig. 9 offers an overview of their relationship. The more fine-grained 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

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

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 time slot. 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 high energy cost. The Cloudlets [20] [19] we introduced in the previous parts 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 the face recognizing program which sequentially executes face detection, face verification and face identification. As discussed in [34] and [39], 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 Java virtual machine (Java Run Environment), .Net framework and 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 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, 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 lead to the lower possibility of container reuse. On the other hand, 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 form of functions as a code fragment. MAUI [16] and ThinkAir [18] 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 optimal solution may take longer time 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 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 need a robust identification mechanism to distinguish their source applications.

7 DISCUSSION AND PERSPECTIVES

After presenting the above research work 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 edge cloud.

Mathematical Models: We further summarize and compare the algorithms discussed in this paper into Table 2. Besides the proposed five categories, we classify the related work also based on the mathematical models and optimization methods utilized. Examples of such methods include 0-1 integer linear programming problem, K-dimensional bin parking, 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. For example, a hierarchical edge cloud architecture was proposed to solve the offloading problem during the peak hours [40]. Dynamic voltage scaling technique was implemented to vary the energy supplement of mobile devices based on the computation loads [41]. To address the potential requirements of future applications, the existing algorithms should be adjusted and new approaches need to be explored.

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 shares more frequent and fast than ever before. [1] According to the forecast of IHS and Mckinsey [42] [43], the devices connected to 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, 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 cover 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 [44]. By offloading heavy workloads to edge servers, both computation and communication latency are guaranteed to extract valuable information from data. Facilitated by edge cloud, 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 unexploited field.

Table 2. Characteristics of offloading algorithms

Algorithms	Destination	Balance	Mobility of Device	Partition	Granularity	Mathematic Model
MAUI[4]	single server	online	simply restart latest disconnected offloading	dynamic	method	linear programming
CloneCloud[5]	single server	online	NS (not specified)	dynamic	method	linear programming
ThinkAir[6]	multiple servers	online	NS	dynamic	method	linear programming
Cloudlet[7]	multiple servers	online	NS	dynamic	application	linear programming
Online-OBO and Online-Batch[9]	single server	online	NS	NS	application	K-dimension bin packing
Primal-dual approach[10]	multiple servers	online	NS	NS	application	NP-hard primal-dual approach
Myopic Maxweight[11]	multiple servers	online	NS	NS	application	myopic MaxWeight algorithms with various routing policies
RIAL[12]	multiple servers	offline	NS	NS	application	multi-criteria decision making method
Bandwidth Guaranteed[13]	single server	offline	NS	NS	application	self-designed model
Alg-MBM[14]	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[16]	multiple servers	online	cooperation among edge servers	static	NS	Markov Decision Process
2D MDP[17]	multiple servers	online	cooperation among edge servers	static	NS	Markov Decision Process
Ad Hoc assisted CTS[19]	multiple servers	online	cloudlets and ad hoc mobile devices	dynamic	task	0-1linear programming
Static partition scheme[20]	single server	online	NS	static	task	branch and bound + pruning heuristic
ALL and K-Step[21]	single server	online	NS	static + dynamic	task	self-designed model
DOA[22]	single server	NS	center cloud and edge cloud work cooperatively in the two-tiered architecture	dynamic	task	Lyapunov optimization
Joint optimization algorithm[23]	multiple servers	NS	NS	dynamic	task	convex optimization

EC Offloading for 5G: In the future, 5G will bring us broader network bandwidth and greater convenience of device connectivity which 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 heavier burden in the 5G era. It may casue the increment of cost when cloud resources are accessed through the 5G network. Currently, there are some related researches of offloading in such situation. For example, a time-adaptive heuristic algorithm with multiple radio access technology (Multi-RAT) is proposed [45]. A distributed computation offloading algorithm solves the offloading decision-making problem in the wireless network with multi-channel in order to avoid mutual interference [46]. To improve the performance of offloading in 5G network, EC can play an important role to enhance its upper bound.

Security Problems with EC Offloading: Security is another serious challenge for the future edge cloud service. First, data offloaded from mobile devices are exposed to the multiple threats: malicious eavesdropping, changing contents of data packets, destroying channel connectivity and uploading destination. The confidentiality, access controllability and integrity can not easily be guaranteed. Second, the orchestration of resources in the edge servers may be disturbed and the reliability and dependability of edge service can be compromised. In this case, the initialization and migration of VMs may not accurately match application requirements. Third, the IoT function visualization needs data security to align horizontal and vertical input information by utilizing SDN (Software-Defined Networking) and NFV (Network Functions Virtualization) technologies [47]. Moreover, the data synchronization of an application should not be disrupted even if a single communication access bears multiple applications.

8 CONCLUSIONS

In this paper, 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 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 in drawing an overall "big picture" for the existing efforts. Embracing the future network development, we plan to continuously explore emerging technologies and creative ideas that improve the offloading performance.

ACKNOWLEDGMENTS

The work is supported in part by National Security Agency (NSA) under grants No.: H98230-17-1-0393 and H98230-17-1-0352, National Aeronautics and Space Administration (NASA) EPSCoR Missouri RID research grant under No.: NNX15AK38A, National Science Foundation (NSF) award CNS-1647084, and a University of Missouri System Research Board (UMRB) award.

REFERENCES

[1] V. Ovidiu and F. Peter, "Building the hyperconnected society," 2015. [Online]. Available: http://www.internet-of-things-research.eu/pdf/Building_the_Hyperconnected_Society_IERC_2015_Cluster_eBook_978-87-93237-98-8_P_Web.pdf

[2] H. Chang, A. Hari, S. Mukherjee, and T. V. Lakshman, "Bringing the cloud to the edge," in *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, April 2014, pp. 346–351.

[3] "Amazon web service." [Online]. Available: https://aws.amazon.com
[4] "Microsoft azure." [Online]. Available: https://azure.microsoft.com
[5] "Google cloud." [Online]. Available: https://cloud.google.com/
[6] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012, pp. 13–16.
[7] European Telecommunications Standards Institute (ETSI), "Mobile-edge computing initiative," 2016, available at: http://www.etsi.org/technologies-clusters/technologies/mobile-edge-computing.
[8] S. S. Cross, T. Dennis, and R. D. Start, "Telepathology: current status and future prospects in diagnostic histopathology," *Histopathology*, vol. 41, no. 2, pp. 91–109, 2002. [Online]. Available: http://dx.doi.org/10.1046/j.1365-2559.2002.01423.x
[9] S. Ryu, "Telemedicine: Opportunities and developments in member states: Report on the second global survey on ehealth 2009 (global observatory for ehealth series, volume 2)," *Healthcare Informatics Research*, vol. 18, no. 2, pp. 153–155, 2012.
[10] J. Pan, L. Ma, R. Ravindran, and P. TalebiFard, "Homecloud: An edge cloud framework and testbed for new application delivery," in *2016 23rd International Conference on Telecommunications (ICT)*, May 2016, pp. 1–6.
[11] R. Gargees, B. Morago, R. Pelapur, D. Chemodanov, P. Calyam, Z. Oraibi, Y. Duan, G. Seetharaman, and K. Palaniappan, "Incident-supporting visual cloud computing utilizing software-defined networking," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, no. 1, pp. 182–197, Jan 2017.
[12] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys Tutorials*, vol. PP, no. 99, pp. 1–1, 2017.
[13] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A survey on mobile edge networks: Convergence of computing, caching and communications," *IEEE Access*, vol. PP, no. 99, pp. 1–1, 2017.
[14] A. Ahmed and E. Ahmed, "A survey on mobile edge computing," in *2016 10th International Conference on Intelligent Systems and Control (ISCO)*, Jan 2016, pp. 1–8.
[15] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, "A survey of computation offloading for mobile systems," *Mobile Networks and Applications*, vol. 18, no. 1, pp. 129–140, 2013.
[16] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: Making smartphones last longer with code offload," in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '10. New York, NY, USA: ACM, 2010, pp. 49–62. [Online]. Available: http://doi.acm.org/10.1145/1814433.1814441
[17] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic execution between mobile device and cloud," in *Proceedings of the Sixth Conference on Computer Systems*, ser. EuroSys '11. New York, NY, USA: ACM, 2011, pp. 301–314. [Online]. Available: http://doi.acm.org/10.1145/1966445.1966473
[18] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *2012 Proceedings IEEE INFOCOM*, March 2012, pp. 945–953.
[19] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, Oct 2009.
[20] T. Verbelen, P. Simoens, F. De Turck, and B. Dhoedt, "Cloudlets: Bringing the cloud to the mobile user," in *Proceedings of the Third ACM Workshop on Mobile Cloud Computing and Services*, ser. MCS '12. New York, NY, USA: ACM, 2012, pp. 29–36. [Online]. Available: http://doi.acm.org/10.1145/2307849.2307858
[21] Q. Xia, W. Liang, and W. Xu, "Throughput maximization for online request admissions in mobile cloudlets," in *38th Annual IEEE Conference on Local Computer Networks*, Oct 2013, pp. 589–596.
[22] F. Hao, M. Kodialam, T. V. Lakshman, and S. Mukherjee, "Online allocation of virtual machines in a distributed cloud," *IEEE/ACM Transactions on Networking*, vol. PP, no. 99, pp. 1–12, 2016.
[23] S. T. Maguluri, R. Srikant, and L. Ying, "Stochastic models of load balancing and scheduling in cloud computing clusters," in *2012 Proceedings IEEE INFOCOM*, March 2012, pp. 702–710.
[24] D. B. West, *Introduction to Graph Theory*, 2nd ed. Prentice Hall, September 2000.
[25] L. Chen, H. Shen, and K. Sapra, "Rial: Resource intensity aware load balancing in clouds," in *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, April 2014, pp. 1294–1302.
[26] K. Mochizuki, H. Yamazaki, and A. Misawa, "Bandwidth guaranteed method to relocate virtual machines for edge cloud architecture," in *2013 15th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, Sept 2013, pp. 1–6.
[27] Q. Xia, W. Liang, Z. Xu, and B. Zhou, "Online algorithms for location-aware task offloading in two-tiered mobile cloud environments," in *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, Dec 2014, pp. 109–116.
[28] T. Taleb and A. Ksentini, "An analytical model for follow me cloud," in *2013 IEEE Global Communications Conference (GLOBECOM)*, Dec 2013, pp. 1291–1296.

[29] A. Ksentini, T. Taleb, and M. Chen, "A markov decision process-based service migration procedure for follow me cloud," in *2014 IEEE International Conference on Communications (ICC)*, June 2014, pp. 1350–1354.

[30] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic service migration in mobile edge-clouds," in *2015 IFIP Networking Conference (IFIP Networking)*, May 2015, pp. 1–9.

[31] T. Verbelen, P. Simoens, F. D. Turck, and B. Dhoedt, "Leveraging cloudlets for immersive collaborative applications," *IEEE Pervasive Computing*, vol. 12, no. 4, pp. 30–38, Oct 2013.

[32] Z. Wu, L. Gui, J. Chen, H. Zhou, and F. Hou, "Mobile cloudlet assisted computation offloading in heterogeneous mobile cloud," in *2016 8th International Conference on Wireless Communications Signal Processing (WCSP)*, Oct 2016, pp. 1–6.

[33] Z. Li, C. Wang, and R. Xu, "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*, ser. CASES '01. New York, NY, USA: ACM, 2001, pp. 238–246. [Online]. Available: http://doi.acm.org/10.1145/502217.502257

[34] D. Huang, P. Wang, and D. Niyato, "A dynamic offloading algorithm for mobile computing," *IEEE Transactions on Wireless Communications*, vol. 11, no. 6, pp. 1991–1995, June 2012.

[35] B.-G. Chun and P. Maniatis, "Dynamically partitioning applications between weak devices and clouds," in *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, ser. MCS '10. New York, NY, USA: ACM, 2010, pp. 7:1–7:5. [Online]. Available: http://doi.acm.org/10.1145/1810931.1810938

[36] L. Zhang, D. Fu, J. Liu, E. C. H. Ngai, and W. Zhu, "On energy-efficient offloading in mobile cloud for real-time video applications," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, no. 1, pp. 170–181, Jan 2017.

[37] H. Shahzad and T. H. Szymanski, "A dynamic programming offloading algorithm for mobile cloud computing," in *2016 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, May 2016, pp. 1–5.

[38] I. Giurgiu, O. Riva, D. Juric, I. Krivulev, and G. Alonso, *Calling the Cloud: Enabling Mobile Phones as Interfaces to Cloud Applications*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 83–102. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-10445-9_5

[39] S. Sardellitti, G. Scutari, and S. Barbarossa, "Joint optimization of radio and computational resources for multicell mobile-edge computing," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 1, no. 2, pp. 89–103, June 2015.

[40] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, April 2016, pp. 1–9.

[41] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, "Mobile-edge computing: Partial computation offloading using dynamic voltage scaling," *IEEE Transactions on Communications*, vol. 64, no. 10, pp. 4268–4282, Oct 2016.

[42] S. Lucero, "Iot platforms: enabling the internet of things," 2016 March. [Online]. Available: https://cdn.ihs.com/www/pdf/enabling-IOT.pdf

[43] C. Ip, "The iot opportunity - are you ready to capture a once-in-a lifetime value pool?" 2016 June 21. [Online]. Available: http://hk-iot-conference.gs1hk.org/2016

[44] M. Ali, H. S. M. Bilal, M. A. Razzaq, J. Khan, S. Lee, M. Idris, M. Aazam, T. Choi, S. C. Han, and B. H. Kang, "Iotflip: Iot-based flip learning platform for medical education," *Digital Communications and Networks*, 2017.

[45] S. E. Mahmoodi and K. P. S. Subbalakshmi, "A time-adaptive heuristic for cognitive cloud offloading in multi-rat enabled wireless devices," *IEEE Transactions on Cognitive Communications and Networking*, vol. 2, no. 2, pp. 194–207, June 2016.

[46] Y. Liu, S. Wang, and F. Yang, "A multi-user computation offloading algorithm based on game theory in mobile cloud computing," in *Edge Computing (SEC), IEEE/ACM Symposium on*. IEEE, 2016, pp. 93–94.

[47] F. Reynaud, F. X. Aguessy, O. Bettan, M. Bouet, and V. Conan, "Attacks against network functions virtualization and software-defined networking: State-of-the-art," in *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, June 2016, pp. 471–476.