Joint Computation Partitioning and Resource Allocation for Latency Sensitive Applications in Mobile Edge Clouds

Lei Yang, *Member, IEEE,* Bo Liu, Jiannong Cao, *Fellow, IEEE,* Yuvraj Sahni, and Zhenyu Wang

Abstract—The proliferation of mobile devices and ubiquitous access of the wireless network enable many new mobile applications such as augmented reality, mobile gaming and so on. As the applications are latency sensitive, researchers propose to offload the complex computations of these applications to the nearby edge cloud, in order to reduce the latency. Existing works mostly consider the problem of partitioning the computations between the mobile device and the traditional cloud that has abundant resources. The proposed approaches can not be applied in the context of mobile edge cloud, because both the resources in the mobile edge cloud and the wireless access bandwidth to the edge cloud are constrained. In this paper, we study *joint computation partitioning and resource allocation problem* for latency sensitive applications in mobile edge clouds. The problem is novel in that we combine the computation partitioning and the two-dimensional resource allocations in both the computation resources and the network bandwidth. We develop a new and efficient method, namely Multi-Dimensional Search and Adjust (MDSA), which is an offline algorithm, to solve the problem. We compare MDSA with the classic list scheduling method and the *SearchAdjust* algorithm via comprehensive simulations. The results show that MDSA outperforms the benchmark algorithms in terms of the overall application latency. Moreover, we also design an online method, named by Cooperative Online Scheduling (COS), which can be easily deployed in practical systems. By extensive evaluations, we show that COS outperforms the benchmark methods by 25% on average.

Index Terms—computation partitioning; latency sensitive applications; mobile edge clouds

1 Introduction

The proliferation of sensors on the mobile devices and the increasing wireless bandwidth enable several new mobile applications such as augmented reality, mobile gaming, mobile biometric, touch free human-device interactions an so on. The applications are latency sensitive, since it directly affect the user experiences. Traditional mobile cloud computation offloading approaches rely on the powerful resources in the cloud to accelerate the execution of computation-complex application in order to achieve low latency. Recently as the edge cloud dmodel begins to be deployed in the real world, the mobile applications are offloaded to the edge cloud because of the closer distance and lower delay to the mobile users than the Internet cloud [17].

Computation partitioning is an effective way to optimize the decisions on which parts of an application should be offloaded to the edge cloud and which others are executed on the local devices. Existing works mostly focus on the partitioning of the applications from the viewpoint of the mobile users [3] [4] [6]. The optimization is often done for a single mobile user such that the execution cost in terms of time or energy consumption on the device is minimized [1] [2] [7]. However, in the context of the edge cloud, the traditional methods of optimizing a single user's partitioning may not be applied. The resources of the edge cloud are quite constrained compared with the traditional cloud on Internet. The users compete with each other for both the computation resources in the edge cloud and the access bandwidth to the edge cloud. The partitioning decisions are highly dependent on the resources allocated from the edge cloud. It is more important to partition the user's applications cooperatively from the standpoint of the edge cloud in order to achieve high overall application performances for all the users.

In this paper, we study the joint computation partitioning and resource allocation for latency sensitive application in the mobile edge cloud. Our problem is novel in that we combine the computation partitioning with the two dimensional resource allocation. In recent related work, the authors propose the similar idea, but they consider the resource allocation in the cloud and neglect the bandwidth allocation [18] [19]. We model the new problem as a joint optimization for the three decisions, i.e., the partitioning of each user's application, the allocation of computation resources in the edge cloud, and the allocation of bandwidth to the mobile users. The problem is challenging since the decisions are affected by each other. In particular, the partitioning decisions of the users have various resources by demand, and meanwhile the resource allocation affects the user's partitioning. Furthermore, we need to balance the resource allocation among the users, and balance the two dimensional resources including the computation resources and the bandwidth allocated for a user.

To solve the problem, we develop an efficient heuristic, named Multi Dimensional Search and Adjust (MDSA),

- *L. Yang, B. Liu, and Z. Wang are with the School of Software Engineering, South China University of Technology, China. Email: sely, wangzy@sely.scut.edu.cn; boliu168@gmail.com*
- *J. Cao and Y. Sahni are with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong. E-mail: csjcao, csysahni@comp.polyu.edu.hk*

which is an offline solution. The method divides the network bandwidth into a few virtual channels. It first relaxes the resource constraints and generates an initial partitioning and execution schedule for every user. The method searches the earliest time when there exists a violation of the constraint either on the network bandwidth or on the edge cloud computation resources, and adjusts the user's partitioning to avoid the violation at that time. The searching and adjustment are done alternatively and iteratively until the initial partitioning converges to a feasible solution. We evaluate our off-line method through extensive numeric simulations. The results show that MDSA achieves much lower overall application delay than the benchmark methods including the classic list scheduling [16], and the single dimensional SearchAdjust which is a heuristic proposed in our previous work [18].

On the basis of the off-line solution, we further develop an online solution named by Cooperative Online Scheduling (COS) for the problem. COS makes the partitioning and allocation decision for the users every time a new task is released, and it makes use of the adjustment (or rescheduling) strategy to balance the workloads onto the network and edge cloud. The load balancing avoids extremely long waiting time in the network transmission and edge cloud execution. Through extensive simulations, it is shown that COS has better performance than the benchmark methods especially when the workload is high. As an online method, COS can be easily deployed in the practical systems for mobile edge offloading. More importantly, COS is considered as a general method for online scheduling with multiple types of resources. As follows we highlight our contributions of this paper.

- We propose and formulate a new problem, named as Joint Computation Partitioning and Resource Allocation Problem (JCPRP). To the best of our knowledge, this work is the first one to study the problem in the mobile edge computing.
- We develop a novel heuristic, named as Multiple Dimensional Search and Adjust (MDSA) to solve the problem. We compare MDSA with the benchmark methods through extensive numeric simulations. The results show MDSA outperforms the benchmark methods in terms of the overall application latency.
- We design an online method, i.e., Cooperative Online Scheduling (COS), including the implementation architecture, algorithms and mechanisms. COS can be easily deployed and implemented in practical systems. Our methods for the problem can be used in solving general scheduling problems with multiple types of constrained resources.

2 System Models and Problem Formulation

2.1 An Example of the Real Application

One mobile application that can leverage the edge cloud to accelerate its execution is Augmented Reality (AR), which continuously recognizes the scene in reality from the camera streams, and augments the streams with related information. Fig. 1 shows an AR application scenario in a campus

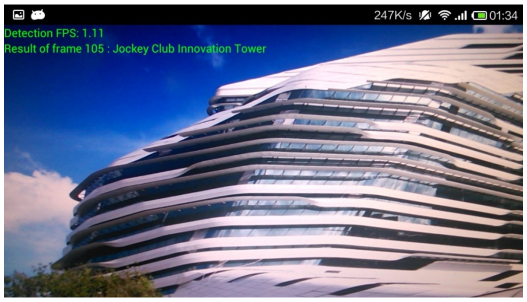


Fig. 1: An Augmented Reality application scenario in the campus environment

environment. The users take pictures of the buildings when they walk around the campus. The application recognizes the building in the campus and displays interesting things on the video streams, e.g., the events happening in the building. The application helps visitors who are not familiar with the campus to easily find their interested places and activities. The core function in AR is object recognition from the video frames. The device usually executes the function periodically while the user moves, aiming to recognize the varying scene of the surrounding environment in time. We measure the execution time of the recognition function on the main-streaming hardware with 1.7 G Hz 4 Core CPU and 2G RAM. It takes at least 60 seconds to process one 1000*800 frame in the video. If the resolution increases, it can take longer time. In our previous work [17], we have implemented a platform to partition the application between the device and cloud and thus significantly speed up the execution. However, the platform lacks the mechanism of resource allocation in case of serving multiple users.

2.2 System Model

We consider the latency sensitive applications running on the mobile device. The application is usually composed of several functional modules which have data dependency between each other. Thus, to simplify the problem, we model the application as a sequential task graph. The application is composed of n modules, and its input data and output data can be abstracted as two virtual modules with no computation load. The **application latency** is defined by the summation of the execution time of the modules and the transmission time on the edges.

Fig.2 shows the system model. The edge cloud is deployed behind the cellular Base Station (BS) or the wireless Access Point (AP). We consider the users who connect to the edge cloud via one BS or AP. The users share the bandwidth provided by the wireless network. When the users start to execute the applications, the execution cost of each module on their mobile device is sent to the edge cloud. These execution costs are usually obtained in advance by offline profiling. With knowing the execution cost of all the users, the edge cloud makes the partitioning decisions for the users on which modules are offloaded to the cloud and which others are executed on the mobile devices, and at the meanwhile allocates the shared bandwidth to the users. The aim is to minimize the average application latency of the users. The edge cloud has a constrained number of servers which can accommodate the offloaded modules,

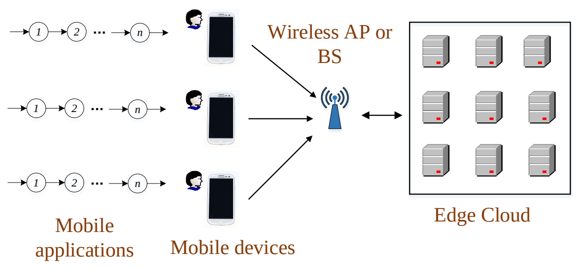


Fig. 2: The system model

TABLE 1: Mathematical notations in this paper

λ	the total number of users;
r	the number of servers on the edge cloud;
B	the total bandwidth of the access network;
n	the total number of modules in the application graph;
T	the length of the whole time period;
i	index of the user (or the mobile device);
j	index of the module in the application graph;
τ	index of the time slot;
D_j	the amount of data that needs to be transmitted from module j to $j+1$;
$w_{i,j}$	the execution cost (time) of the module j on the user device i;
c_j	the execution cost (time) of the module j on the edge cloud server;
$x_{i,j}$	a binary variable indicating whether the module j of the user i is executed on the edge cloud;
$t_{i,j}$	the time when the module j of the user i starts to execute;
$y_{i,j,\tau}$	the bandwidth allocated to the data transmission from the module j to the module $j+1$ for the user i at the τ-th time slot;
N_{ch}	the number of network channels;
R_j	the priority of allocating the module m to an idle resource;
μ_j	estimated average edge server's execution time for module;
φ	the number of modules to be adjusted;
γ	constant ratio to constrain the number of modules adjusted ;
Q_{edge}	the number of modules waiting in the edge cloud queue;
Q_{net}	the number of data flows waiting in the cloud queue;

so the partitioning decision guarantees that at any time the number of offloaded modules in execution from the users would not exceed the number of servers. To simplify the problem, we assume that all the users run the same application. However, our models and proposed solution can be easily extended to the generalized cases where the users run various applications.

2.3 Problem Formulation

Suppose there exists λ users offloading computations to the edge cloud. The edge cloud has a number r of servers. The users access the edge cloud through the commonly shared wireless AP or cellular base stations. The total bandwidth provided by the AP or the BS is B. We assume the users' mobile devices are heterogeneous, while the servers in the edge cloud are the same. The execution time of each module in the application on the mobile devices is denoted by $w_{i,j}$, which in particular indicates the execution of j-th module on the user i-th device, where $1 \leq i \leq \lambda$ and $1 \leq j \leq n$. The execution time of each module j on the edge cloud server is denoted by c_j. Let D_j denote the amount of data required to transmit between the j-th module and the $j+1$ module if the two modules are executed at the different sides.

Decision variables. The problem is to: 1) partition the computations for the user's applications, i.e., to decide whether each module is executed at the mobile side or at the edge cloud side; 2) schedule the offloaded computations onto the edge cloud servers; 3) allocate the bandwidth to the data transmissions arising from the edges in the application graph whose two connective modules are executed at the different sides. We introduce the binary variable $x_{i,j}$ to denote whether the j-th module from the i-th user is executed on the edge cloud. $x_{i,j} = 1$ if the j-th module of the user i is executed on the edge cloud; otherwise $x_{i,j} = 0$. We introduce the variable $t_{i,j}$ to represent the start execution time of the j-th module of the user i.

Suppose the time is divided into small slots. All the time related variables such as $w_{i,j}$, c_j and $t_{i,j}$ are measured in slots. We consider the whole time period with the total number T of slots. Let τ be the index of the time slots, and we have $1 \leq \tau \leq T$. Under the module placement $x_{i,j}$, we define the edge in the application graph with its two connective modules placed at different sides as the *cross edge*. The cross edge actually causes a data transmission (or flow) in the network. Let $\mathbf{f}_{i,j}$ denote the flow from the module j to the module $j+1$ for the user i. We define the *release time* of the flow $\mathbf{f}_{i,j}$ by the completion time of its precedent module, which is formulated by $t_{i,j} + (1 - x_{i,j}) \cdot w_{i,j} + x_{i,j} \cdot c_j$. We define the *deadline* of the flow $\mathbf{f}_{i,j}$ by the start time of its successive module $t_{i,j+1}$. Now we introduce the variable $y_{i,j,\tau}$ to denote the bandwidth allocated to the flow $\mathbf{f}_{i,j}$ at the time τ.

Objectives. The objective is to minimize the average application delay of all the users. For the convenience of formulating the objective, we add two virtual modules into the application graph including the start module denoted by $j = 0$, and the end module denoted by $j = n+1$. Since the application normally gets the input data from the mobile device and is required to output the result to the mobile device as well, we have $x_{i,0} = 0$ and $x_{i,n+1} = 0$ for each user i. With the two virtual modules added, $t_{i,0}$ indicates the start execution time of the application of user i, which is usually equals to the release time of the user's application. $t_{i,n+1}$ denotes the end time of application. Therefore, the objective can be formulated by

$$\min \frac{1}{\lambda} \sum_{i=1}^{\lambda} (t_{i,n+1} - t_{i,0}), \tag{1}$$

Constraint on the dependency of the modules. The execution time of the modules should satisfy the dependency constrained in the application graph. The module can not be executed until its precedent module is finished. We assume that the module's execution is not preemptive. Once it is scheduled on one of the edge cloud servers, it will be finished without suspension. Thus, we can represent the execution interval of the j-th module from the i-th user by $[t_{i,j}, t_{i,j} + (1 - x_{i,j}) w_{i,j} + x_{i,j} c_j]$. The dependency constraint among the modules is formulated by

$$t_{i,j} \geq t_{i,j-1} + (1 - x_{i,j-1}) \cdot w_{i,j-1} + x_{i,j-1} \cdot c_{j-1}, \quad \forall i \in [1, \lambda], \forall j \in [1, n+1]. \tag{2}$$

Constraint on the resources on the edge cloud. Suppose that every server in the edge cloud can not process multiple modules in parallel at the same time. At every time slot, the number of modules that are executed in the edge cloud should be less than the total number r of servers. The constraint is formulated by

$$\forall \tau, \sum_{i=1}^{\lambda} \sum_{j=1}^{n} x_{i,j} \cdot \mathbb{F}(\tau - t_{i,j}) \cdot \mathbb{F}(t_{i,j} + c_j - \tau) \leq r \tag{3}$$

.

Note that $\mathbb{F}(x)$ is the sign function. The value of the function is equal to 1 if the variable $x \geq 0$; otherwise it is zero.

Constraint on the network bandwidth. It is required that at every time slot, the total bandwidth allocated to the flows in the network should be less than B. For a particular time point, we first identify the flows that have the transmission interval covering the time point. We then calculate the total bandwidth according to the bandwidth allocation $y_{i,j,\tau}$, and guarantee that it can not exceed the network bandwidth B. The constraint is formulated by

$$\forall \tau, \sum_{i=1}^{\lambda} \sum_{j=0}^{n} (x_{i,j} - x_{i,j+1})^2 \cdot \mathbb{G}(i,j,\tau) \leq B, \tag{4}$$

where $\mathbb{G}(i,j,\tau)$ is a function to test whether the transmission interval of the flow from the module j to $j+1$ of the user i covers the time point τ. If it covers, then the value of the function is 1; otherwise it equals to zero. We represent $\mathbb{G}(i,j,\tau)$ using the sign functions

$$\mathbb{G}(i,j,\tau) = \mathbb{F}[\tau - t_{i,j} - (1 - x_{i,j}) \cdot w_{i,j} - x_{i,j} \cdot c_j]. \times \mathbb{F}(t_{i,j+1} - \tau), \tag{5}$$

At the meanwhile, the allocated bandwidth to the flow should be enough to transmit the amount of data before the flow's deadline. Thus, we also have the constraint as follows.

$$(x_{i,j} - x_{i,j+1})^2 \cdot [\sum_{\delta = t_{i,j} + (1-x_{i,j}) \cdot w_{i,j} + x_{i,j} \cdot c_j}^{t_{i,j+1}} y_{i,j,\delta} - D_j] = 0, \quad \forall i \in [1, \lambda], \forall j \in [0, n]. \tag{6}$$

Definition 1 *Joint Computation partitioning and Bandwidth allocation Problem (JCBP).* Given the application graph and associated parameters $\{n, D_j\}$, the execution cost of each module on both the local devices and the edge cloud $\{w_{i,j}, c_j\}$, the execution cost on the edge cloud c_j, the number of servers on the edge cloud r, and the network bandwidth B, the problem is to partition the modules between the mobile devices and the edge cloud servers, and meanwhile allocate the bandwidth to the arising data transmissions, such that the average application latency is minimized. The problem is formulated as follows.

$$\min_{x_{i,j}, t_{i,j}, y_{i,j,\tau}} \frac{1}{\lambda} \sum_{i=1}^{\lambda} (t_{i,n+1} - t_{i,0}), \tag{7}$$

s.t. (2), (3), (4), (6).

3 Offline Solution

3.1 A Naive Solution

We introduce a naive solution to the problem. In the solution, we first assume that the resources in the edge cloud are unconstrained, and every single user is allocated with the total bandwidth B. We then obtain the optimal partitioning for every user's application graph by using the algorithm for Single user Computation Partitioning Problem (SCPP) which has been introduced in our previous work [18]. Having determined the places where the modules are executed, we schedule the offloaded modules onto the edge cloud servers and as well as schedule the data transmissions (flows) within the network according to a first-come-first-serve policy.

In the scheduling, we abstract the offloaded modules and data flows as tasks. Each task has a release time which can be calculated according to the initial partitioning and an execution cost. The r servers on the edge and the network channel can be abstracted as $r+1$ machines. The tasks are assigned with priorities based on their release time. The one with an early release time has a high priority. The algorithm schedules the task with the highest priority, and selects the machine which can execute the task at the earliest time. If the actual execution time of the task is later than its release time, which means it has to be delayed due to the resource unavailability, we update the release time of its precedent tasks in the corresponding application graph. The algorithm selects the next task to be scheduled, and determines the machine where it is executed. The task selection and machine determination are done iteratively until all the tasks are scheduled. Note that during the machine determination, we constrain that the offloaded modules should be placed on one of the edge cloud servers, while the data flows should be allocated to the network channel.

3.2 Muti-Dimensional Search and Adjust (MDSA)

We propose a new heuristic algorithm, named Multi-Dimensional Search and Adjust (MDSA), to solve the joint computation partitioning and resource allocation problem. The algorithm divides the whole bandwidth resource into a number of N_{ch} virtual channels. Each channel has the bandwidth $\frac{B}{N_{ch}}$. Assume that multiple flows can not be transmitted concurrently in one channel. N_{ch} is a parameter in the algorithm, and usually much less than the number λ of users. Similar to the naive algorithm, we first neglect the constraints of the servers in the edge cloud and the network bandwidth, and calculate the optimal partitioning for every single user. Note that each user has the bandwidth $\frac{B}{N_{ch}}$ while computing its optimal partitioning.

According to the initial partitioning, we calculate the *execution schedule* of the users, which indicates the execution time of the modules and transmission time of the data flows. From the execution schedule, we can count at which interval and how many modules are executed concurrently in the edge cloud. We search the earliest time when the number of modules being executed exceeds the constraint r, which is defined as the *server critical point*. By the same method, we can also search the earliest time when the number of flows being transmitted concurrently is large than the constraint N_{ch}. We call this time point as *network critical point*. If the server critical point occurs before the network critical point, we will adjust the modules by moving them back to the mobile device and/or delaying the execution of modules at the corresponding servers, such that the constraint violation is resolved at this critical point; otherwise, we will adjust the flows by changing the execution places of their adjacent modules or delaying the transmission of flows. Either the server critical point or the network flow point is resolved, we will update the users' execution schedule and search the next critical point in terms of servers and network utilization.

The searching of critical points and adjustment of modules or flows are done iteratively until no critical point is found. In our previous work, we only consider the constraint of the cloud servers, while neglecting the bandwidth constraints. In our method, we extend the method into the case of constraints in multi-dimensional resources. That is why we name our algorithm as multi-dimensional search and adjust. Algorithm 1 shows the pseudocode of MDSA, in which the related data structures and terminologies are defined as follows.

- **Execution Schedule**. Execution schedule is a data structure to store the time intervals of the execution of modules and transmission of flows for a user. Assuming that the edge cloud has unconstrained servers and network channels, under the initial partitioning $x_{i,j}$ of the user i, we can calculate the start execution time of the module j by

$$t_{i,j} = \sum_{j'=0}^{j-1} [(1 - x_{i,j'})w_{i,j'} + x_{i,j'}c_{j'} + (x_{i,j'} - x_{i,j'+1})^2 \cdot \frac{D_{i,j'}}{\Delta B}], \tag{8}$$

where $\Delta B = \frac{B}{N_{ch}}$. With knowing $t_{i,j}$, we can easily obtain the completion time of the module j and the transmission interval of the flow $\mathbf{f}_{i,j}$. Our algorithm maintains the execution schedule for every user.

- **Servers Utilization List**. Servers utilization list includes a set of time intervals, and the number of servers being utilized by the modules during the intervals. According to the execution schedule of the users, we select the execution intervals of all the modules offloaded to the edge cloud. The bounds of these intervals divide the whole time period into a number of short intervals. For every short interval, we count the number of modules whose execution interval covers this short interval. If the number of an interval exceeds the number r of servers, it means during this interval the server constraint is violated.

- **Network Utilization List**. Network utilization list includes a set of time intervals and the number of network channels being utilized by the data transmissions during the intervals. We can also count the number of data flows being transmitted during each interval according to the user's execution schedule. This number indicates the number of the demanded channels.

- **Servers Critical Point**. Server critical point is defined as the earliest time when the violation of the servers constraint in the edge cloud occurs. We can easily obtain it based on the *server utilization list*.

- **Network Critical Point**. Network critical point is defined as the earliest time when the violation of the network constraint occurs, which means the number of demanded network channels exceeds the maximum value N_{ch}. It can be calculated based on the *network utilization list*.

Algorithm 1: MDSA Algorithm

```
Input  : A set of $\lambda$ users, $r$ servers, network bandwidth $B$
Output: the execution schedules of the users
1  Divide the network bandwidth into $N_{ch}$ channels;
2  Compute the optimal partitioning for the users by
     assuming each user is allocated with bandwidth $\frac{B}{N_{ch}}$;
3  Compute the execution schedule for the users ;
4  Obtain the server occupation list and network occupation
     list ;
5  Search the server critical point $t_{SCP}$ and network critical
     point $t_{NCP}$;
6  if $t_{SCP} < t_{NCP}$ then
7      Find the modules which are executed in the edge
         cloud and have the execution interval covering
         $t_{SCP}$;
8      for each of the modules do
9          Compute the reward of adjusting the module
             including moving the module to the mobile
             device or delaying the execution of module;
10     Select the modules with the $\alpha$ greatest rewards to
         adjust ;
11 else
12     Find the data flows which have the transmission
         interval covering $t_{SCP}$;
13     for each of the flows do
14         Computing the reward of adjusting the flow,
             including changing the execution place of the
             flow's adjacent module or delaying the
             transmission of the flow;
15     Select the modules with the $\beta$ greatest rewards to
         adjust;
16 Update the execution schedule of the selected modules ;
17 Update the server occupation list and network occupation
     list;
18 if search the $t_{SCP}$ or search the $t_{SCP}$ then
19     Go to Line 5 ;
20 return the execution schedule of the users
```

3.2.1 Adjustment of Modules

As shown in Algorithm 1 (**Line** 9), to eliminate the resource violation at the servers critical point, we have two ways to adjust the modules: 1) moving the module from the edge cloud to the mobile device; 2) delaying the execution of the module on the server. Our algorithm compares the reward of the two ways for each candidate module, and chooses the maximum one as the adjustment reward. The algorithm selects modules with the top α rewards to adjust, where α is equal to the number of demanded servers at the critical point minus the constraint r.

We define a **reward function** to evaluate the benefit of doing the adjustment by

$$Reward = t_{server} + t_{net} - t_{delay}, \tag{9}$$

where t_{server} represents the decreased occupation period on the server due to the adjustment, t_{net} represents the decreased occupation period of data transmission in the network due to the adjustment, and t_{delay} is the increase of application delay caused by the adjustment. In calculating

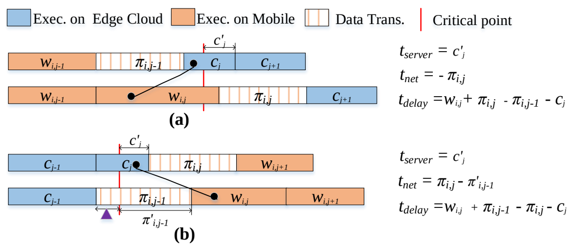


Fig. 3: Module adjustment at the servers critical point

the t_{server} and t_{net}, we compare the execution schedule of a user after the adjustment with the original execution schedule. Starting from the critical point to the end of the user's application, we count by how much the server occupation time is decreased and by how much the data transmission is decreased by. Fig.3 illustrates how to calculate the three parameters.

Under this reward function, if adjusting a module releases much server or network resources, and causes slightly longer application delay, this module will have high reward, and thus has high priority to be adjusted finally. For the way of delaying the module, the reward only depends on t_{delay}, since it does release some resources on the server or the network channel. Now we ask the question to what time the execution of module is delayed when we evaluate the reward of delaying a module. In our algorithm, we delay the execution of module to the nearest time point when the number r of servers are not all occupied.

Note that moving a module to the mobile device may cause the violation of network constraint before a servers critical point. Fig.3(b) illustrates the case as an example, in which the triangle marks the interval that may cause a new network critical point before the current servers critical point. The case should be avoided, because in each iteration of our algorithm, we want to guarantee that the new critical point due to the adjustment should occur after the original one in the time axis. By achieving this, our algorithm is able to converge fast. If moving a module to the mobile device causes new network critical point before the servers critical point, we only consider to delay the module and assign the delay to the module as its adjustment reward.

3.2.2 *Adjustment of Data Flows*

The adjustment of a flow include two ways: 1) changing the execution place of the adjacent module that the flow points to; and 2) delaying the transmission of flow in the network channel. We use the same reward function in Equation (9) to calculate the reward of adjusting a flow. For each flow whose transmission interval covers the *network critical point*, we compute the rewards of both the adjustment ways, and assign the maximum one to the module as its reward. If changing the execution place of the flow's adjacent module leads to the violation of server constraint before the *network critical point*. We will only consider the delaying of flows. Similar to the determination of α, the parameter β in Line 15 is equal to the number of the demanded channels at the critical point minus the constraint N_{ch}.

4 Online Solution

Since MDSA is an offline algorithm, it could be used in real system if the arrival time of the users requests could be predicted. In most cases where the workloads of the users are not predictable at the start time, MDSA will not be used in practical deployment but in theoretical performance analysis. In this section, we will develop an online solution, named by Cooperative Online Scheduling (COS), and its implementation architecture for more general cases. In contrast to the offline solution, the online solution only requires the release time of the current request but does not need the knowledge about the future requests. The partitioning decision for one user's request will not be determined until the request is released.

4.1 COS Architecture

First, we introduce some terminologies frequently used in the description of the online solution.

Job and **Task**. We regard the users request for executing a mobile application as a **job** request. As shown by the application graph in Section 2.2, a job is divided into n modules, and each module is an indivisible unit of computation. Here the module is also called by **task**.

Job Release Time, Task Release Time and **Task Release Place.** The *job release time* is defined as the time when the job request arrives at the system. We define the release time of a task as the completion time of its precedent task. If a task is the first task of a job, the task release time is equivalent to the job release time. The release place of a task is defined as the place where its precedent task is executed. It could either be on the mobile device or on the edge cloud. By default, the first task of a job is released on the mobile device, since we assume that the input data of the application are from the mobile device.

COS is a task oriented online scheduling method. Fig.4 shows the system architecture of COS. It contains two schedulers, i.e., edge cloud scheduler and network scheduler. The edge cloud scheduler is responsible for allocating the computational resources on the edge cloud to the tasks, while the network scheduler takes charge of allocating the network resources to the arising data transmission. The two schedulers work cooperatively to balance the workloads on the two types of resource, aiming to avoid long waiting time on either of the two types of resources. Note that the network resource is modeled as multiple channels, each of which has the same bandwidth.

Trigger Message for the Schedulers. The schedulers are triggered by various messages. The edge cloud scheduler is triggered whenever a task is released. More specifically, the trigger time could be the time when a new job is released or the time when a task is completed. The trigger messages respectively correspond to Job Release Message (JRM) and Module Completion Message (MCM). The network scheduler is triggered when a data transmission (flow) is completed on a network channel. The trigger message is named as Data Flow Completion Message (DFCM).

Furthermore, the two schedulers trigger each other by exchanging messages, which include Edge Cloud Triggered Message (ECTM) and Network Triggered Message (NTM) in Fig.4. The edge cloud scheduler generates a ECTM when it

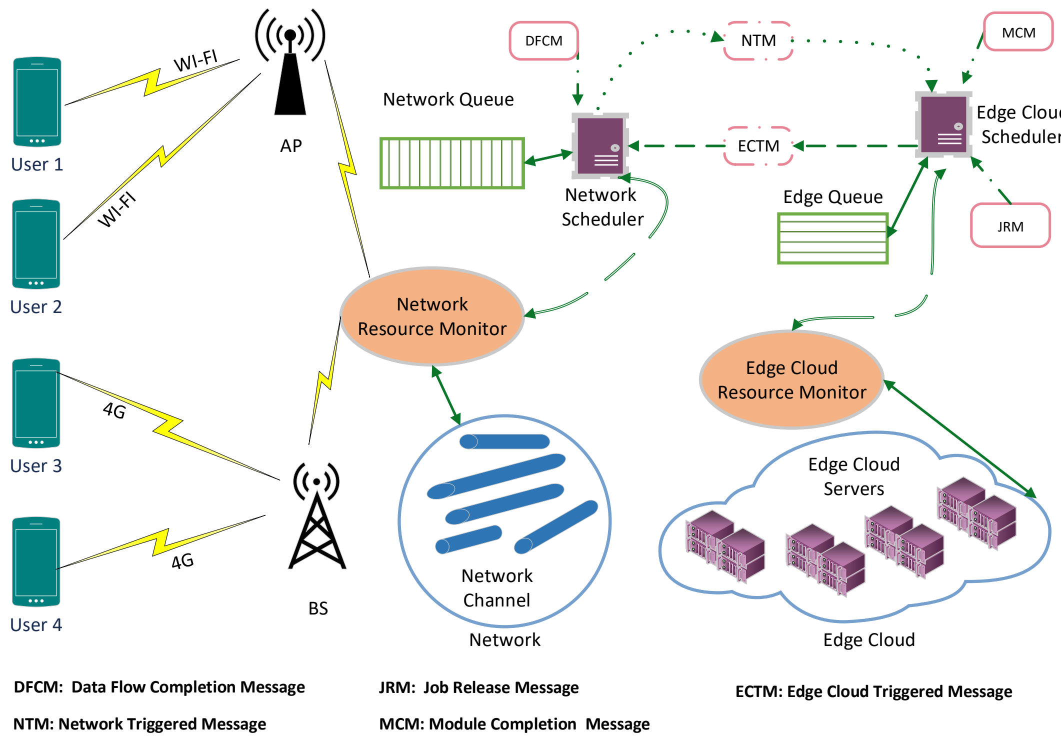


Fig. 4: Architecture of Cooperative Online Scheduling

allocates a task released on the mobile device onto the edge cloud or allocates a task released on the edge cloud onto the mobile device. In both situations, the execution place of a task is different from its release place, which causes additional data transmission overhead. At this moment, the network scheduler is triggered by the message to deal with the data transmission. The network scheduler sends a NTM to the edge cloud scheduler as a data flow is completed. This moment indicates that the successive task of the completed data flow is ready to be executed, and the edge cloud scheduler needs to deal with the task in ready.

The two schedulers have associated queues and resource monitors. The edge cloud queue is implemented at the edge cloud, and it is used to queue the tasks which are ready to be executed. The network queue stores the data flows which are waiting to be transmitted over the network. The two resource monitors detect the idleness of the resources and provide relevant information to the schedulers.

4.2 COS Algorithms and Strategies

When a user initiates an application on the mobile device, a Job Release Message (JRM) arrives at the edge cloud scheduler. The scheduler then considers to determine the execution place of the first task of the job. It compares the cost of executing it on the mobile device and the cost of remote execution by assuming zero waiting time of data transmission on the network. If the execution on mobile device is faster than remote execution, the task is then allocated to the mobile device; otherwise, the task is allocated to the edge cloud. The data needed by the task will be transmitted from the mobile device to the edge cloud at first. The edge cloud scheduler adds a data flow into the network queue, and notifies the network scheduler by sending a ECTM message. When a task is completed on the edge cloud, the edge cloud scheduler directly puts the successor of the completed task into the edge cloud queue if the completed task is not the last task of a job. It then selects a task from the queue and executes it on an idle server of the edge cloud. The edge cloud scheduler frequently check the length of the queue, and moves some of the tasks back to the mobile devices if the workload on the edge cloud is high. We call this phase as *module adjustment*. This strategy is to avoid waiting too long time for the tasks on edge cloud. If a task released on the edge cloud is adjusted to run on the mobile device, the edge cloud scheduler adds the data flow into the network queue, and wakes up the network scheduler by sending an ECTM message. The adjustment in this case reduces the load on edge cloud but would bring additional transmission load on the network.

Upon receiving ECTM message, the network scheduler checks if there exists any idle channels. If it exists, the scheduler allocates a channel to one of the data flow in the queue. The allocation strategy will be described in details in Section 4.2.1. When a data flow is completed on the network channel, the network scheduler selects a data flow from the queue and allocates it to the channel based on the same allocation strategy. The network scheduler also monitors the loads in its queue, and will cancel the data flow by changing the execution place of the task to which the data flow is adjacent. The algorithm only allows the change of the successive task of the data flow. If a data flow to be adjusted is from the edge cloud to the mobile device, the network scheduler adds the successive task of the data flow to the edge cloud queue, and notifies the edge cloud scheduler by sending a NTM message. Although this adjustment alleviates the transmission loads on the network,

Algorithm 2: The scheduling algorithm running in the edge cloud scheduler

```
1  if receive the message MCM then
2      if the completed module is executed on the mobile device then
3          if ($T_{local}$)>(($T_{channel}$)+($T_{server}$)) then
4              add the the module's successive data flow into network queue, and send a message ECTM to the network scheduler;
5          else
6              execute the module on the mobile device;
7          return;
8      if the completed module is the last module of a job then
9          add the module's successive data flow into network queue, and send a message ECTM to the network scheduler;
10         return;
11     add the successive module into edge cloud queue;
12     execute edge cloud server allocation strategy;
13     while the number of modules adjusted is less than $\varphi \times \gamma$ do
14         select a module from the queue with $maf = 0$ to adjust;
15         if the module is released at the mobile device then
16             execute the module on the mobile device immediately;
17         else
18             add the successive data flow of the module into the network queue, and send a message ECTM to the network scheduler;
19         update the module's $maf$ value;
20 else if receive the message NTM then
21     repeat line 12 to 19;
22 else if receive the message JRM then
23     repeat line 3 to 7;
24 return;
```

Algorithm 3: Scheduling algorithm in the network

```
1  if receive the message DFCM then
2      if the completed data flow is from mobile device to the edge cloud then
3          add the data flow's successive module into the edge cloud queue, and send a message NTM to the edge cloud scheduler;
4      else
5          execute the data flow's successive module on the mobile device ;
6      execute the network channel allocation strategy ;
7      select $\phi$ data flows from $B_{adj}$ to execute network channel adjustment strategy;
8      while the number of data flows that have been adjusted is less than $\varphi \times \gamma$ do
9          select a data flow with $maf = 0$ to be adjusted;
10         if the data flow is from the edge cloud to the mobile device then
11             add the data flow's successive module into the edge cloud queue, and send a message NTM to the edge cloud scheduler;
12         else
13             execute the data flow's successive module on the mobile device;
14         update data flow's $maf$ value;
15 else if receive the message EGTM then
16     repeat line 6 to 14;
17 return;
```

it could increase the computation load on the edge cloud. In contrast, if a data flow to be adjusted is from the mobile device to the edge cloud, the adjustment will decrease the loads both on the network and the edge cloud.

The novel idea of COS lies in the cooperation of the two schedulers. The cooperation not only includes the message based notification with each other but also includes the adjustment strategies. As explained above, the adjustment done by either of the schedulers would affect the performance of the other one. Through design of the adjustment strategies, COS will balance the workload on the network resources and the edge cloud resources, and therefore achieves a good trade-off between the waiting time of the tasks for the network transmission and that for the edge cloud execution. Algorithm 2 and Algorithm 3 respectively shows the pseudo-code of the algorithms in the edge cloud scheduler and network scheduler.

4.2.1 *Allocation Strategy*

In the following section, we describe the task allocation strategy in the online solution. Whenever a task is completed on a server in the edge cloud, the edge cloud scheduler selects the module which has the largest R_j from the edge cloud queue, where R_m is defined by

$$R_j = \frac{\mu_j + c_j}{c_j}. \tag{10}$$

In Equation(10), μ_j denotes the waiting time of the task so far in the queue. c_j denotes the execution time of the task on the edge cloud server. The task that has waited for a long time and has short execution time will be scheduled with high priority. R_j is usually named by response ratio in many scheduling systems. In COS, both the edge cloud scheduler and the network scheduler use the high-response-ratio-first policy to allocate the tasks and data flows.

4.2.2 *Adjustment Strategy*

In order to alleviate the workloads of the edge cloud, the edge cloud scheduler will iteratively pick up some of the modules to the mobile device from the edge cloud queue. The number of modules that is returned to the mobile device in each iteration depends on the minimum value of the following three variables: (i) the number of modules in the edge cloud queue minus r (ii) the number of modules whose execution time is greater than the average execution time over all the modules; (iii)the number of modules whose waiting time in the queue is greater than the average waiting time over all the modules. Suppose we use d, e, f respectively to denote the three variables above. The number of modules to be adjusted φ is shown by

TABLE 2: Parameters of the workloads

Parameters	Values
The number of users λ	100
The number of servers r	25
The network bandwidth B	240 Mbps
Modules in the application graph n	4
The average execution time of modules on edge cloud	0.15 s
The average execution time of modules on mobile devices	0.6 s
The average data transmission size	0.75 MB

$$\varphi = \text{Min}\{d, e, f\}. \tag{11}$$

In Equation(11), φ should be positive. When φ is d, we select the first d released modules to return to the mobile devices for execution. If φ is equal to e, the e modules with the greatest execution time will be returned to the mobile devices. If φ is f, the modules whose waiting time μ_j exceeds the average waiting time will be adjusted to the mobile device. For the simplicity of analysis, we use a list φ_{list} to store all the information about these modules to be adjusted. Apart from the adjusted modules, the remaining modules in the edge cloud queue will be still waiting. The pseudocode of module adjustment is described in the algorithm 2.

Cooperative Load Balancing. At each time of adjustment, the edge cloud scheduler determines the number of modules to be adjusted according to Equation (11). The number is then multiplied by a constant ratio γ which is defined by

$$\gamma = \frac{Q_{edge/r}}{Q_{net}/N_{ch}}, \tag{12}$$

where Q_{edge} represents the number of modules waiting in the edge cloud queue, and Q_{net} is the number of data flows waiting in the network queue. r is the number of servers in the edge cloud. N_{ch} is the number of network channels. γ indicates the ratio of load between the edge cloud and network. $\gamma < 1$ represents that the network has relatively higher load than the edge cloud. In this case, the number of modules to be adjusted should be reduced by multiplying a ratio γ. If γ is greater than 1, the edge cloud is busier than the network. In this case, we will constrain the value of γ by 1. The number of modules to be adjusted remains the same with the initial value calculated by Equation (12).

The adjustment in the network follows the same policy with that in the edge cloud. However, the difference is that the constant ratio γ should be replaced by $\frac{1}{\gamma}$. By introducing the constant ratio γ, our adjustment strategy reduces the workload on one of the scheduler, but meanwhile avoids overloading the other scheduler.

4.2.3 *Avoidance of Adjustment Oscillation*

Oscillation is defined as a procedure in which the same workload is adjusted from one queue to the other queue repeatedly, and can not be executed all the time. For instance, if a module released on the edge cloud is adjusted, the associated data flow would be added into the network queue. After waiting some time in the network queue, this data flow is then adjusted by re-allocating the successive task onto the edge cloud. The task in the edge cloud queue is then adjusted after waiting some time. This procedure could continue in our algorithm and the task would not be executed. To avoid the oscillation, we mark each module and data flow in the queues with a count variable which represents the times that it has been adjusted. Each time when a module or data flow is adjusted, we update the count variable. At the time we select the modules or data flows to be adjusted, the ones that have been adjusted for several times will not be adjusted any more. In our algorithms, we define a threshold maf to constrain the times of adjustment of the same module or data flow.

5 PERFORMANCE EVALUATION

In this section, we will respectively evaluate the performance of the proposed offline solutions and online solutions. In the offline solutions, the release time of all the requests are assigned to zero for simplicity. In the online solution, the release time of all the requests are different and random. Through the evaluation of various offline and online solutions, we aim to answer two questions: 1)which solution performs the best; 2)which factors influence the performance in terms of the average application latency; 3)how the application performance varies depending on the factors such as the edge cloud resources, network resources, and the workloads.

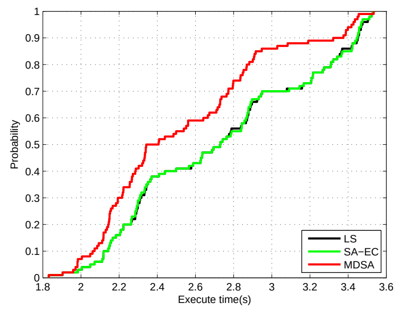
5.1 Evaluations of offline solutions

5.1.1 *Environment Setting*

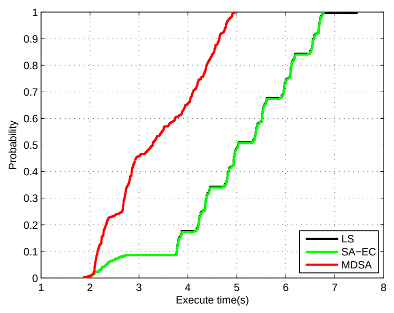
In the simulation, we generate an application graph with 4 modules. The number of users is 100 by default. On the edge cloud, we set the number of servers as 20, which is usually much less than the number of users. The average execution time of the modules at the edge cloud server is 0.15s. The execution time of the modules on the mobile devices are different with the users, since their device have different processing capability. We generate local execution time randomly with the average time of a module being 0.6s. The amount of data transmission on the edges in the user's application graph are generated randomly with the mean of 0.75MB. The total bandwidth of the access network is 240 Mbps. The number of network channels is 30 and each channel has the bandwidth 8 Mbps. Table 2 shows the default values of the environment parameters.

We compare the proposed MDSA algorithm with two benchmark algorithms, which are described as follows.

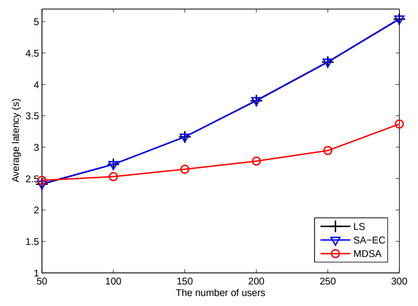
- **List Scheduling (LS)**. List scheduling is a classic method for scheduling tasks on the constrained resources [16]. It contains two steps: task selection and resource assignment. The naive algorithm presented in Section 3.1 pertains to the list scheduling methd, where the tasks including the modules and data transmissions are selected to be scheduled according to a first-come-first-serve policy, and the resource assignment is determined by the initial partitioning. For fairness in the comparison, we also allow the division of the bandwidth into multiple channels as we do in MDSA algorithm.



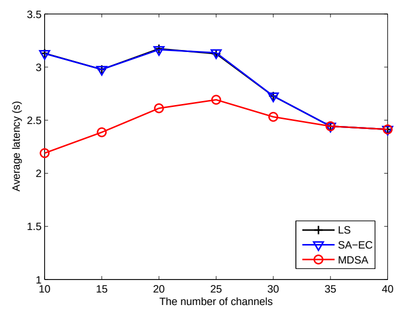
(a) The CDF of application latency ($\lambda = 100$)



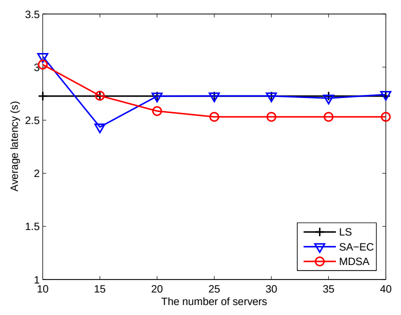
(b) The CDF of application latency ($\lambda = 300$)



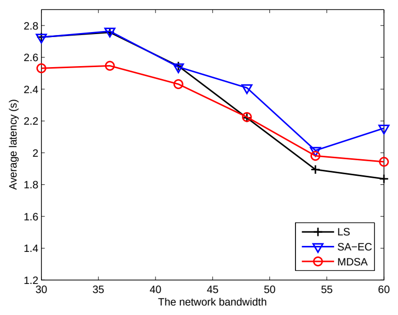
(c) The impact of λ to the average latency



(d) The impact of N_{ch} to the average latency



(e) The impact of r to the average latency



(f) The impact of B to the average latency

Fig. 5: Performance results of the three off-line algorithms

- **SA-EG**. The algorithm does the searching and adjustment of the users' partitioning when scheduling the modules onto the servers in the edge cloud, while uses the list scheduling to deal with the data flows in the network. The method pertains to the single dimension Search and Adjustment, named as Search and Adjust for Edge Clouds (SA-EG), which is introduced in our previous work [18].

The primary metric we concern on is the average latency of the user applications. Fig.5a and Fig.5b shows the cumulative distribution function of the application latency respectively when $\lambda = 100$ and $\lambda = 300$. We can see that in both cases MDSA achieves lower latency than the other benchmark algorithms. Specially when $\lambda = 300$, the average execution time (or latency) is less than the benchmark algorithms by 34%. SA-EG does not have obvious better performance than the list scheduling. That means the searching and adjustment only for the resources in the edge cloud does not improve the performance, since the application latency is affected by both the resources in the edge cloud and the bandwidth.

Fig.5d indicates the impact of the number of users onto the average application latency. It is obvious that the average latency increases as the number of users increases, since the resources on the edge cloud and network keeps unchanged. It is important to note that MDSA has much better performance over the benchmark algorithms when there exists a large number of users. The reason is that when the number of users increase, MSDA has high chance to avoid the network congestions by adjusting the partitioning of the user applications.

Next, we want to know how the number N_{ch} of channels affects the performance of the algorithms. In this evaluation, we test the performance when the number of channels ranges from 10 to 40. Fig.5d shows that MDSA achieves the shortest average latency when $N_{ch} = 10$. As N_{ch} increases, MDSA has an increasing average latency. It is because the more channels may cause the lower utilization of network bandwidth. If N_{ch} is too large, the users do not have conflict in the data transmission in the networks. In such case, MDSA has the same performance with the benchmark algorithms.

We evaluate the sensitivity of the performance to the number of servers. In the evaluation setting, we change the number r of servers from 10 to 40, and keep the other parameters as the default values. Fig.5e shows that how the performance changes depending on r. The average latency of MDSA decreases as r increases, while the two benchmark algorithms are not sensitive to the change of r. It is because the benchmark algorithms adopt the list scheduling for data transmissions which causes long delay and becomes the bottleneck of the performance. The increasing of edge cloud servers do not lead to the performance enhancement. We can also find that the performance of MDSA is better than the benchmark algorithms in most cases, while as the number of servers is less than 2, MDSA has worse performance.

Fig.5f shows how the performance changes depending on the total bandwidth. The average latency would be reduced as the bandwidth increases. We can find that MDSA has lower average latency than the list scheduling in the case that the total bandwidth is relatively constrained. However, if the bandwidth is abundant, i.e., greater than 48 Mbps, MDSA does not have advantage over the list scheduling, because there exists no competition among the users for the bandwidth in such case. It is not necessary to adjust the partitioning to avoid the network congestions.

5.2 Evaluations of online solutions

We still use the same application with previous simulations, in which the number of modules is $n = 4$. The users' requests for executing the applications do not arrive at the same time. Instead, we assume that the arrival time of requests are uniformly distributed in a certain interval. With the release pattern of these requests, we evaluate the performance of the online methods. In particular, we increase the scale of the simulated system. The number of users ranges from 100 to 500. The total bandwidth of the network is 602 Mbps. The number of network channel is 80 and each channel has different bandwidth in a range of (7, 8) Mbps. Similarly, the amount of data transmission in the user's application graph are generated randomly from 2 Mb to 2.3 Mb. Table 3 shows the default parameter settings for the online simulation.

TABLE 3: Parameters of the workloads

Parameters	Values
The number of users λ	$(100, 500)$
The number of servers r	60
The network bandwidth B	602.5280 Mbps
The number of modules in the application graph n	4
The number of network channels N_{ch}	80
The network bandwidth of each channel	$(7, 8)$ Mb/s
The execution power of each edge cloud server	$(12, 13)$ Mb/s
The execution time of modules on mobile devices	$(0.8, 1)$ s
The data transmission size	$(2, 2.3)$ MB

We compare the proposed COS with two benchmark methods, which are described as follows.

• **Local Execution (Local)**. Local execution means all the computation modules are executed on the mobile device.

• **Edge cloud Scheduling (Ecloud)**. If we delete the adjustment strategies in the Cooperative Online Schedule, we then get a naive scheduling method which is named by Edge cloud Scheduling (Ecloud). In this method, if a task is released on the mobile device, the scheduler compares the cost of offloading it to the edge cloud and the cost of local execution, and chooses the decision with the lower cost. If a task is released on the cloud, the scheduler directly allocates the task onto the edge cloud. The tasks allocated to the edge cloud are scheduled on the servers according to a First-Come-First-Serve (FCFS) policy. Meanwhile, the arising data flows are allocated to the network channels based on the FCFS policy. In Ecloud, the scheduling of the tasks and data transmissions are independent. There exists no interactions and knowledge sharing between the two schedulers.

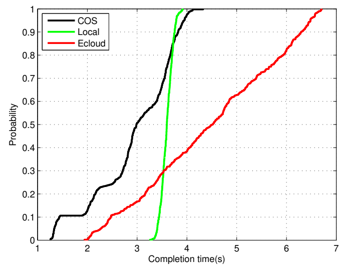
The major performance metric we concern is the average completion time of applications of all users. Fig.6a shows the results of the cumulative distribution of the users completion time when the user number λ = 500. The completion time of the users by using COS is located in the interval (1.2, 4.0) with an average around 2.7s. The completion time by using the local execution method is from 3.2s to 3.9s with an average of 3.6s. Our online algorithm COS has 25 % lower completion time over the local execution method. It is also shown that Ecloud has the worst performance compared with the other two methods, because Ecloud without the adjustment strategy would cause long waiting time at the edge cloud.

The completion time of a user's application contains five parts, i.e., the execution time on the mobile device, the waiting time for data transmission, data transmission time, the waiting time for the edge cloud execution, and the execution time on the edge cloud. Fig.6b and Fig.6c show how much time is spent on each of the five procedures, which are respectively labeled by local, net wait, net, edge wait, and edge in the figures. When $\lambda = 100$, by using COS algorithm, the time in data transmission contributes the most to the total completion time, which is slightly more than the time in edge cloud execution and mobile execution. When λ increases to 500, by using the COS algorithm, the most time-consuming procedure changes to be the mobile execution, because COS allocates most of the modules onto the mobile devices when the workload is high. Through comparing COS and Ecloud, we found that COS algorithm has significantly less waiting time for the edge cloud execution than Ecloud. The reason is that COS adjusts the tasks in the edge cloud queue back to the mobile device if the workload in the edge cloud is high. Accordingly it leads to more time spent in the local execution as shown by the first bar in Fig.6b. In particular, when λ = 500, as shown in Fig.6c, COS has much less time in both the waiting time for the network transmission and edge cloud execution than Ecloud. Fig.6d and Fig.6e compares COS and Ecloud in terms of the waiting time in network transmission. Fig.6f and Fig.6g compares COS and Ecloud in terms of the waiting time in edge cloud execution. This is because COS can balance the workloads of the network and the edge cloud by the cooperation of the network scheduler and edge cloud scheduler.

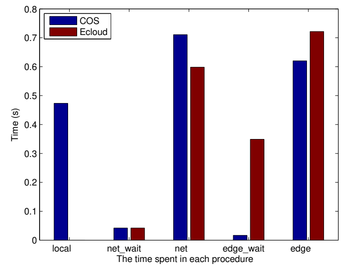
Next, we want to know how the network bandwidth influences the performance of the algorithms. In the e-valuation settings, we change the total bandwidth from 361Mbps to 1084Mbps, and keep the other parameters as the default values in Table 3. Fig.6h shows that as the network bandwidth increases, the average completion time decreases. COS per- forms better than Ecloud under different bandwidths. When the network bandwidth exceeds a large value, the completion time would not decrease obviously with the increase of bandwidth. In this case, the network bandwidth is no longer the bottleneck resource that affects the performance.

Fig.6i shows how the release rate of jobs affects the performance. By default in the evaluation setting, we have 500 requests from users which are released randomly during a time interval of 1 second. Now, we change the length of the time interval from 1 second to 7 seconds, and keep the number of the users as the default value. We can see that as the time interval increases, which means the release rate of jobs gets slow, the average completion time for both COS and Ecloud deceases. This is because the load on the network and edge cloud gets low as the jobs arrive with a low rate. The performance of the local execution method is not affected by the release rate because no competition for resources exists among users if all the modules are executed locally.

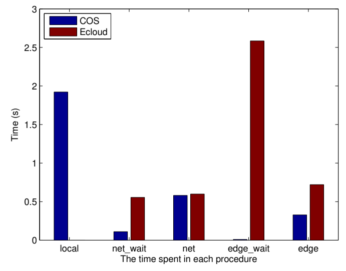
Then, we will evaluate how the performance varies depending on the number of edge cloud servers. In this e-valuation, the number of servers ranges from 50 to 80. Fig.6j shows as the number of servers r increases, the average



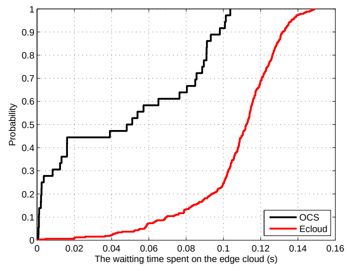
(a) The CDF of average completion time(λ=500)



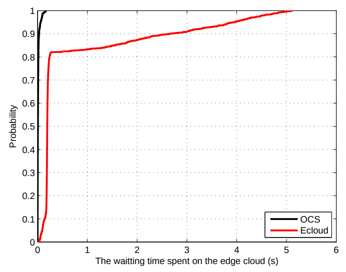
(b) The time spent in the five procedures(λ=100)



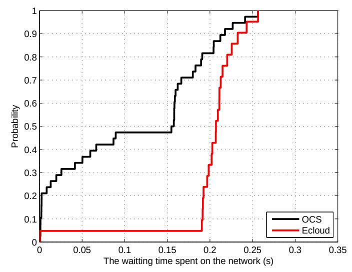
(c) The time spent in the five procedures(λ=500)



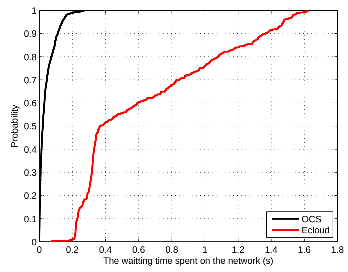
(d) The waiting time for the edge cloud execution(λ=100)



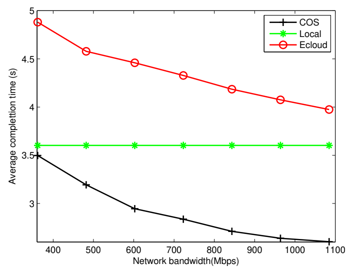
(e) The waiting time for the edge cloud execution(λ=500)



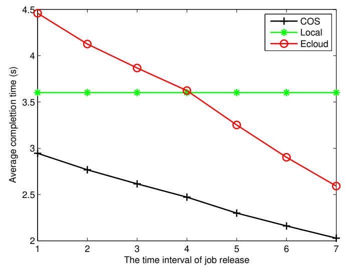
(f) The waiting time for the network transmission(λ=100)



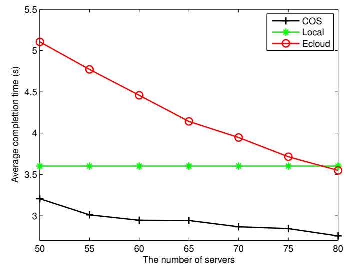
(g) The waiting time for the network transmission(λ=500)



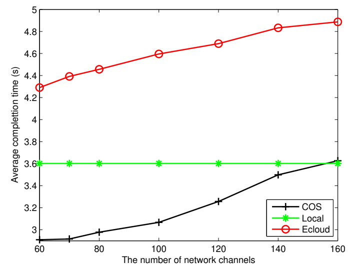
(h) The impact of the bandwidth to the average completion time



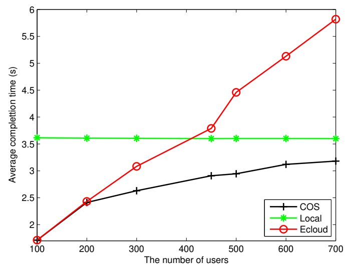
(i) The impact of the job release rate to the average completion time



(j) The impact of the number of servers to the average completion time



(k) The impact of the number of network channels to the average completion time



(l) The impact of the number of users to the average completion time

Fig. 6: Performance results of the online algorithms

completion time for COS decreases. COS achieves the least average completion time when $r = 80$. It is shown that COS has obviously better performance than Ecloud under all the values of r. The local execution method does not change with the variance of the edge cloud resources.

Now, we evaluate the sensitivity of the performance to the number of N_{ch}, which is shown in Fig.6k. We set N_{ch} as various values located in the range of (60,160) without changing the other parameters. COS has the optimal performance when $N_{ch} = 60$. As N_{ch} increases from 60 to 160, the performance of COS degrades. The reason is that the number of network channels is relatively abundant when the number of users is 500 by default. Increasing N_{ch} in this case leads to the decrease of the bandwidth that a channel has, and thus the data transmission in one single channel gets slower. Meanwhile, it is possible that some of the channels may be idle all the time, which causes a low utilization of the network resources. Moreover, in our model we assume that the virtual channels almost have the same bandwidth. We also evaluate the performance of COS when the channels have various bandwidth. The data flow being ready for transmission is always allocated with the channel with the maximum bandwidth. We find that the performance degrades greatly if the variance of the channels' bandwidth is high. When the variance changes with a threshold, the performance will be not obviously affected.

Finally, we compare the performance of the three algorithms under the changing number of users. Fig.6l shows how the performance changes with the increase of λ. When λ =100 and λ=200, COS and Ecloud almost have the same performance. It is because that when the number of users is small, network resources and edge cloud resources are relatively adequate. In such case, the adjustment strategy in COS does not have much benefit. However, when λ increases to 440 or even more, Ecloud performs the worst among the three methods, while our proposed COS algorithm still has better performance than the other two benchmark algorithms. The reason is that the adjustment strategies in COS reduce the loads on the network and edge cloud. It takes effect in improving the performance especially when the system has a high workload.

6 Related Works

We first present an overview of the related works in Mobile Edge Cloud (MEC), and then present the research on computation partitioning and resource allocations in MEC.

Mobile Edge Clouds. One fundamental research in MEC is about the architecture design [11] [12] [22] [24]. Tong et al proposes a tree-based topology for the MEC system, in which the edge clouds are geo-distributed and form a hierarchical architecture [11]. Chen et al developed a D2D Crowd system model in mobile edge computing and solved the energy-efficient D2D Crowd task assignment problem [22]. The architecture can efficiently migrate the peak workload from the mobile users among the edge clouds. Ceselli et al studies the placement of edge cloud and corresponding access points among the candidates sites [12].

According to the functionality that the edge cloud supports in MEC applications, related researchs are categorized into computation offloading [9] [10] [20] [23], content caching [13] [14] [27] [28], content adaption [21]. Rodrigues et al presents a method for minimizing service delay in two-cloudlet scenario [20]. Zeydan et al studies the models and methods of caching the big data on the edge cloud in order to avoid long latency in accessing the data from remote Internet [13]. Yang et. al study the optimal data placement problem on the geo-distributed edge cloud [14]. Yan et al proposes a hybrid edge cloud and application framework for HTTP adaptive streaming [21]. Other functionalities by the edge cloud include network transmission controlling [30] [31], content aggregation [29], and virtual machine migration [25] [26].

Computation Partitioning. Computation partitioning has been studied a lot in the past years. The earliest works consider the single user computation partitioning model. The problem is to decide for a single user which parts of an application should be executed locally and which parts are executed remotely. MAUI [1] and CloneCloud [2] are representative systems that support the partitioning of mobile applications to reduce the execution time and/or save energy consumption. The work in [5] [7] partitions the execution of mobile data stream applications and aim to achieve high throughput. Yang et al developed a method for achieving the optimal partitioning in dynamic mobile cloud environment [8].

By combining the computation partitioning with the workload scheduling within the cloud, researchers propose the multi-user computation partitioning model. The aim is to solve the issue arising from the competition among multiple users for the resources in the cloud [18] [19]. Yang et al [18] proposed a novel multi-user computation partitioning algorithm to minimizing the average application latency for the mobile users. Besides the competition for cloud resources, existing works on multi-user computation partitioning also considers the users' competition for physical layer communication resources. Chen et al [9] studied a multi-user computation offloading game in a multi-channel wireless interference environment. You jointly considers the offloading and the communication resource allocation in a TDMA model [10]. [23] propose a computation offloading model by considering multiple objectives including energy consumption, execution delay, payment cost and so on.

However, the work above neglects the competition for bandwidth among the users at the wireless access points or base station. The bandwidth allocation impacts the overall application performance of the users. Our paper jointly optimizes the computation partitioning and the two dimensional resource allocation which includes both the computing resources at the cloud and bandwidth resources in the network. Although recent work studies the joint optimization of the two dimensional resource allocations in MEC, it does not do the partitioning decision for the users [10].

7 Conclusion

In this paper, we study the joint computation partitioning and resource allocation for latency sensitive application in mobile edge cloud. We developed an efficient heuristic, named by Multi-Dimensional Search and Adjust (MDSA), to solve the problem. On basis of the MDSA, we further develop an online solution, named by Cooperative Online Scheduling (COS). Through extensive evaluation, we conclude that both MDSA and COS has better performance over the benchmark methods. As massive data are now generated by the mobile devices, edge clouds will be the primary places for provisioning the application services for the purpose of low latency. The problem solved in the paper is related to the service deployment which is one of the fundamental problem in service computing, and aims to optimize the binding of the computational services to the physical resources including the edge clouds and mobile devices. Our proposed models and solutions will provide new insight for solving the service deployment in a mobile edge cloud environment.

Acknowledgments

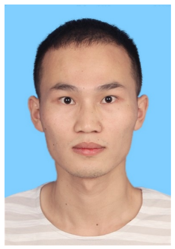
This work is supported by the National Natural Science Foundation of China under Grant 61502312, Hong Kong RGC General Research Fund under Grant PolyU 152244/15E, Science and Technology Fund of Guangzhou, China, under Key Grant 201802010025, Guangzhou Education Bureau-University Innovation and Entrepreneurship Platform Construction Key Project 2019PT103 and Fundamental Research Funds for the Central Universities.

References

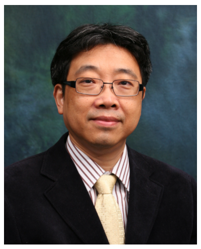
[1] E. Cuervoy, A. Balasubramanianz, and D. Cho. Maui: Making smartphones last longer with code offload. In *Proc. of MobiSys*, pp.277-289, ACM Press, 2010.
[2] B. Chun, S. Ihm, et.al. Clonecloud: Elastic execution between mobile device and cloud. In *Proc. of EuroSys*, pp.301-314, 2011.
[3] I. Giurgiu, O. Riva, et.al. Calling the cloud: Enabling mobile phones as interfaces to cloud applications. In *Proc. of Middleware*, pp.1-20, 2009.
[4] S. Kosta, A. Aucinas, et.al. Thinkair: Dynamic resource allocation and parallel execution in cloud for mobile code offloading. In *Proc. of INFOCOM*, pp.945-953, 2012.
[5] L. Yang, J. Cao, et.al. A framework for partitioning and execution of data stream applications in mobile cloud computing. In *ACM SigMetrics PER*, vol. 40, no. 4, pp.23-32, 2013.
[6] M. Gordon, D. Jamshidi, et.al. Comet: code offload by migrating execution transparently. In *Proc. of OSDI*, pp.93-106,2012
[7] M. Ra, A. Sheth, et.al. Odessa: enabling interactive perception applications on mobile devices. In *Proc. of MobiSys*, pp.43-56, 2011.

[8] L. Yang, J. Cao, et.al. Runtime application repartitioning in dynamic mobile cloud environment. In *IEEE Transactions on Cloud Computing*, vol. 4, no. 3, pp.336-348, 2016.
[9] X. Chen, L. Jiao, et.al. Efficient Multi-User Computation Offloading for Mobile-Edge Cloud Computing. In *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp.2795-2808, 2016.
[10] C. You, K. Huang, et.al. Energy-efficient resource allocation for mobile-edge computation offloading. In *IEEE Transactions on Wireless Communications*, vol. 16, no. 3, pp.1397-1411, 2017.
[11] L. Tong, Y. Li, et.al. A hierarchical edge cloud architecture for mobile computing. In *Proc. of INFOCOM*, pp.1-9, 2016.
[12] C. You, K. Huang, et.al. Mobile edge cloud network design optimization. In *IEEE/ACM Transactions on Networking*, vol. PP, no. 99, pp.1-14, 2017.
[13] E. Zeydan, E. Bastug, et.al. Big data caching for networking: moving from cloud to edge. In *IEEE Communication Magazine*, vol. 54, no. 9, pp.36-42, 2016.
[14] L. Yang, J. Cao, et.al. Cost aware service placement and load dispatching in mobile cloud systems. In *IEEE Transactions on Computers*, vol. 65, no. 5, pp.1440-1452, 2016.
[15] Z. Yan, J. Xue, et.al. Prius: hybrid edge cloud and client adaptation for HTTP adaptive streaming in cellular networks. In *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, no. 1, pp.209-222, 2017.
[16] H. Topcuoglu, S. Hariri, and M. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. In *IEEE Trans. on Para. and Dist. Sys.*, vol. 13, no. 3, pp.260273, 2002.
[17] H. Liu, F. Eldarrat, et.al. Mobile edge cloud system: architectures, challenges, and approaches. In *IEEE Systems Journal*, vol. PP, no. 99, pp.114, 2017.
[18] L. Yang, J. Cao, et.al. Multi-user computation partitioning for latency sensitive applications in mobile cloud computing. In *IEEE Transactions on Computers*, vol. 65, no. 5, pp.1440-1452, 2016.
[19] S. Guo, B. Xiao, et.al. Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing. In *Proc. of Infocom*, pp.1-9, 2016.
[20] T. G. Rodrigues, K. Suto, et.al. Hybrid Method for Minimizing Service Delay in Edge Cloud Computing Through VM Migration and Transmission Power Control. In *IEEE Transactions on computers*, vol.66, no.5, pp.810-819, 2017.
[21] Z. Yan, J. Xue, et.al. Prius: hybrid edge cloud and client adaptation for HTTP adaptive streaming in cellular networks. In *IEEE Transactions on Circuits and Systems for Video Technology*, vol.27, no.1, pp.209-222, 2017.
[22] X. Chen, L. Pu, et.al. Exploiting Massive D2D Collaboration for Energy-Efficient Mobile Edge Computing. In *IEEE Wireless Communications*, vol. 24, no. 4, pp.64-71, Aug. 2017.
[23] L. Liu, X. Guo, et.al. Multi-objective Optimization for Computation Offloading in Fog Computing. In *IEEE Internet of Things Journal*, DOI: 10.1109/JIOT.2017.2780236, 2018.
[24] C. Sarros, S. Diamantopoulos, et.al. Connecting the Edges: A Universal, Mobile-Centric, and Opportunistic Communications Architecture. In *IEEE Communications Magazine*, vol. 56 , no. 2 , pp. 136-143, 2018.
[25] A. Machen, S. Wang, et.al. Live service migration in mobile edge clouds. In *IEEE Communications Magazine*, vol. 25 , no. 1 , pp. 140-147, 2018.
[26] L. Chaufournier, P. Sharma, et.al. Fast transparent virtual machine migration in distributed edge clouds. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, 2017.
[27] Q. Yuan, H. Zhou, et.al. Toward Efficient Content Delivery for Automated Driving Services: An Edge Computing Solution. In *IEEE Network*, vol. 32, no. 1, pp. 80-86, 2018.
[28] D. Zhang, Y. Zhou, et.al. A Multi-Level Cache Framework for Remote Resource Access in Transparent Computing. In *IEEE Network*, vol. 32, no. 1, pp. 140-145, 2018.
[29] E. Fitzgerald, M. Piro, et.al. Energy-Optimal Data Aggregation and Dissemination for the Internet of Things. In *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 955-969, 2018.
[30] X. Li, D. Li, et.al. Adaptive Transmission Optimization in SDN-based Industrial Internet of Things with Edge Computing. In *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 1351-1360, 2018.
[31] G. Aujla, N. Kumar, et.al. Optimal Decision Making for Big Data Processing at Edge-cloud Environment: An SDN Perspective. In *IEEE Transactions on Industrial Informatics*, vol. 14, no. 2, pp. 778-789, 2018.

Lei Yang is currently an associate professor at the School of Software Engineering, South China University of Technology, China. He received the BSc degree from Wuhan University, in 2007, the MSc degree from the Institute of Computing Technology, Chinese Academy of Sciences, in 2010, and the PhD degree from the Department of Computing, Hong Kong Polytechnic University, in 2014. He has been a visiting scholar at Technique University Darmstadt, Germany from Nov. 2012 to Mar. 2013. His research interests include edge computing, mobile cloud computing and Internet of Things with particular focus on task scheduling and resource management.



Liu Bo is a 2nd-year master student in the School of Software Engineering, South China University of Technology, China. He got his BSc degree from Hengyang Normal University, China, 2015. Since September 2016 he has been a post-graduate student working in the laboratory of mobile cloud computing. His research interests include cloud computing, edge computing, and Internet of Things.



Jiannong Cao is a Chair Professor of Distributed and Mobile Computing of the Department of Computing at The Hong Kong Polytechnic University. He is also the director of the Internet and Mobile Computing Lab in the department and the director of University Research Facility in Big Data Analytics. He received the B.Sc. degree in computer science from Nanjing University, China, in 1982, and the M.Sc. and Ph.D. degrees in computer science from Washington State University, USA, in 1986 and 1990 respectively. His research interests include parallel and distributed computing, wireless networks and mobile computing, big data and cloud computing, pervasive computing, and fault tolerant computing. He has co-authored 5 books in Mobile Computing and Wireless Sensor Networks, co-edited 9 books, and published over 500 papers in major international journals and conference proceedings. He is a fellow of IEEE, a member of ACM, a senior member of China Computer Federation (CCF).

Yuvraj Sahni is a Ph.D. student at the Department of Computing, The Hong Kong Polytechnic University, Hong Kong. He received B.E. (Hons) degree in Electrical and Electronics Engineering from Birla Institute of Technology and Science, Pilani, India in 2015. His research interests include wireless sensor network, middle-ware, and Internet of Things.

Zhengyu Wang is a professor and the dean of the School of Software Engineering, South China University of Technology, China. He received the BSc degree from Xiamen University, China, in 1987, and the MSc and the PhD degrees from Harbin Institute of Technology, China, in 1990 and 1993, all in computer science. His research interests include distributed computing and SOA, operating systems, software engineering, and large-scale application design and development.